One-Shot Transfer Learning for Nonlinear PDEs with Perturbative PINNs

Samuel Auroy
ENSTA Paris
samuel.auroy@ensta.fr

Pavlos Protopapas
Harvard University
pprotopapas@g.harvard.edu

Abstract

We propose a framework for solving nonlinear partial differential equations (PDEs) by combining perturbation theory with one-shot transfer learning in Physics-Informed Neural Networks (PINNs). Nonlinear PDEs with polynomial terms are decomposed into a sequence of linear subproblems, which are efficiently solved using a Multi-Head PINN. Once the latent representation of the linear operator is learned, solutions to new PDE instances with varying perturbations, forcing terms, or boundary/initial conditions can be obtained in closed form without retraining.

We validate the method on KPP–Fisher and wave equations, achieving errors on the order of 10^{-3} while adapting to new problem instances in under 0.2 seconds; comparable accuracy to classical solvers but with faster transfer. Sensitivity analyses show predictable error growth with ϵ and polynomial degree, clarifying the method's effective regime.

Our contributions are: (i) extending one-shot transfer learning from nonlinear ODEs to PDEs; (ii) deriving a closed-form solution for adapting to new PDE instances; and (iii) demonstrating accuracy and efficiency on canonical nonlinear PDEs. We conclude by outlining extensions to derivative-dependent nonlinearities and higher-dimensional PDEs.

1 Introduction

Partial differential equations (PDEs) are central to modeling in biology, physics, and engineering. Unlike ordinary differential equations (ODEs), which govern single-variable dynamics, PDEs describe coupled spatial-temporal processes and are harder to solve due to stability, dimensionality, and complex boundaries. Classical ODE methods like Runge–Kutta [1] extend only partially, leaving finite element and related methods [2] as workhorses, but at high computational cost.

Recent machine learning approaches, especially Physics-Informed Neural Networks (PINNs), provide an appealing alternative [3, 4, 5, 6, 7]. By embedding physical laws into neural networks, PINNs bypass discretization and yield differentiable solutions, but each new instance typically requires retraining. Multi-instance strategies such as bundle training [8] or Multi-Head PINNs [9] help, yet generalization across PDE families remains limited.

One-shot transfer learning shows strong results for linear equations [10], but nonlinear cases still need iterative optimization. Building on [11], which addressed nonlinear ODEs with perturbative terms, we extend this approach to PDEs with polynomial perturbations. We validate it on KPP–Fisher equations and argue for broader applicability across nonlinear PDEs. Extending to higher-dimensional PDEs (for example, Navier–Stokes) is future work; here we validate on canonical PDEs with well-understood dynamics to isolate the method's behavior.

Contributions. This work makes the following contributions:

- We extend one-shot transfer learning, previously demonstrated only for nonlinear ODEs, to nonlinear PDEs with polynomial perturbations.
- We derive a perturbative framework that reduces nonlinear PDEs into a sequence of linear subproblems, enabling efficient reuse of a shared latent representation learned by a Multi-Head PINN.
- We demonstrate the method on canonical PDEs (KPP–Fisher and wave equations), showing accuracy comparable to classical solvers while adapting more efficiently to new forcing terms, boundary conditions, and perturbations.
- We analyze sensitivity to perturbation size and polynomial degree, and discuss extensions to related operators and non-perturbative terms.

2 Methodology

We study a class of scalar nonlinear PDEs of order n,

$$\mathcal{D}u + \epsilon P(u) = f(x,t), \tag{1}$$

$$\text{where} \quad \mathcal{D}u = \sum_{|\alpha|=0}^{n} a_\alpha(x,t)\,\partial^\alpha u \text{ for } \alpha \in \mathbb{N}^2, \quad P(u) = \sum_{l=0}^{m} P_l u^l, \text{ and } \forall l \in [0,m],\ P_l \le 1. \tag{2}$$

Here, $\epsilon < 1$ ensures that the polynomial term $\epsilon P(u)$ acts as a perturbation to the linear PDE defined by $\mathcal{D}$. Because all coefficients P_l are bounded by 1, each nonlinear contribution $\epsilon P_l u^l$ remains small relative to the linear term. Appropriate boundary and initial conditions (Dirichlet or Neumann) are imposed so that the problem is well-posed.

2.1 Expansion of the solution

Intuitively, we separate the PDE into a dominant linear operator $\mathcal{D}$ and a smaller nonlinear correction $\epsilon P(u)$. This allows us to write the solution as a series of corrections of increasing order in ϵ. Each correction is governed by a linear PDE, so the nonlinear problem is reduced to a sequence of linear problems.

Following [11] with the technique from [12], assume $u(x,t) = \sum_{i=0}^{\infty} \epsilon^i u_i(x,t)$ and truncate,

$$u(x,t) \approx \sum_{i=0}^{p} \epsilon^i u_i(x,t). \tag{3}$$

For this perturbative expansion to be meaningful, we assume $\epsilon < 1$, so higher-order terms diminish.

Substituting into (1) gives

$$\sum_{i=0}^{p} \epsilon^i \mathcal{D}u_i + \epsilon \sum_{l=0}^{m} P_l \left(\sum_{k_0+k_1+\ldots+k_p=l} \frac{l!}{k_1!k_2!\ldots k_p!} \epsilon^{\sum_{i=0}^{p} ik_i} \prod_{i=0}^{p} u_i^{k_i} \right) = f. \tag{4}$$

The left-hand side is a polynomial in ϵ. Collecting equal powers yields $p+1$ linear PDEs.

At order ϵ^0,

$$\mathcal{D}u_0 = f. \tag{5}$$

For a general order j with $1 \le j \le p$,

$$\mathcal{D}u_j = -\sum_{l=0}^{m} P_l \left(\sum_{\substack{k_0+k_1+\ldots+k_p=l \\ \sum_{i=0}^{p} ik_i = j-1}} \frac{l!}{k_1!k_2!\ldots k_p!} \prod_{i=0}^{p} u_i^{k_i} \right) := f_j. \tag{6}$$

Thus we obtain linear PDEs of the form

$$\mathcal{D}u_j(x,t) = f_j(x,t), \tag{7}$$

where each f_j depends only on u_i with $i < j$. We solve them iteratively to construct

$$u(x,t) \approx \sum_{i=0}^{p} \epsilon^i u_i(x,t). \tag{8}$$

To satisfy ICs and BCs, a common choice is to impose the full problem's ICs/BCs on $\mathcal{D}u_0 = f$ and homogeneous ICs/BCs on higher orders.

2.2 Multi-Head Physics-Informed Neural Network (MH-PINN)

We solve the linear PDEs $\mathcal{D}u_j = f_j$ using a Multi-Head PINN. The network is fully connected with K heads. The kth output is

$$u_k(x,t) = H(x,t)W_k, \tag{9}$$

where $H(x,t)$ is the last shared layer's activation and W_k are head-specific weights.

We compute a physics-informed loss for each head and average:

$$\mathcal{L} = \tfrac{1}{K}\sum_{k=0}^{K} \mathcal{L}_k, \tag{10}$$

$$\begin{aligned}\mathcal{L}_k &= w_{pde}\mathcal{L}_{k,\text{pde}} + w_{IC}\mathcal{L}_{k,\text{IC}} + w_{BC}\mathcal{L}_{k,\text{BCs}} \qquad (11)\\ &= w_{pde}\left(\mathcal{D}u_k(x,t) - f_k(x,t)\right)^2 + w_{IC}\left(u_k(x,0) - g_k(x)\right)^2 + w_{BC}\sum_{\mu\in\{L,R\}}\left(u_k(\mu,t) - B_{\mu,k}(t)\right)^2.\end{aligned}$$

Here $w_{pde}, w_{IC}, w_{BC} > 0$ balance PDE residual, initial conditions, and boundary conditions. Neumann conditions are handled by replacing u_k with $\partial^\alpha u_k$ in IC/BC terms. The total loss is computed over collocation points and minimized by backpropagation. The key objective is to learn a reusable latent space H tied to $\mathcal{D}$.

2.3 One-shot Transfer Learning

Let (f^*, g^*, B^*) denote the forcing, initial data, and boundary data for a new linear problem. With the body frozen, evaluate H at sampled points $(\hat{x}, \hat{t})$ and write

$$u^*(\hat{x},\hat{t}) = H(\hat{x},\hat{t})W^*. \tag{12}$$

Define the loss

$$\mathcal{L} = w_{pde}\left(\mathcal{D}HW^* - f^*\right)^2 + w_{IC}\left(H_0W^* - g^*\right)^2 + w_{BC}\sum_{\mu\in\{L,R\}}\left(H_\mu W^* - B^*_\mu\right)^2, \tag{13}$$

where H_0 and H_μ denote H at initial and boundary points. Each term is convex in W^*, so $\partial\mathcal{L}/\partial W^* = 0$ yields

$$W^* = M^{-1}\left(w_{pde}\mathcal{D}_H^T f^* + w_{IC}H_0^T g^* + w_{BC}\sum_{\mu\in\{L,R\}} H_\mu^T B_\mu^*\right), \tag{14}$$

with

$$M = w_{pde}\mathcal{D}_H^T\mathcal{D}_H + w_{IC}H_0^T H_0 + w_{BC}\sum_{\mu\in\{L,R\}} H_\mu^T H_\mu. \tag{15}$$

Importantly, M depends only on $\mathcal{D}$ and the sampling strategy, not on (f^*, g^*, B^*), so M^{-1} can be reused across new tasks.

3 Results

We applied our methodology to a family of KPP Fisher equations (details on MH-PINN training are provided in Section 6.4):

$$\frac{\partial u}{\partial t} - D\frac{\partial^2 u}{\partial x^2} - \epsilon u^{n_1}(1 - u^{n_2}) = f(x,t). \tag{16}$$

This is the form in (1) with $\mathcal{D} = \frac{\partial}{\partial t} - D\frac{\partial^2}{\partial x^2}$ and $P(u) = -u^{n_1} + u^{n_1+n_2}$. The equation first appeared in population dynamics [13, 14] and has been applied in chemistry [15]. When $f = 0$, it describes evolution between equilibria 0 and 1. For $n_1 = n_2 = 1$ and the conditions from Table 6.1, we compute a solution under conditions similar to those in [16] and observe the propagation toward the equilibrium value 1.

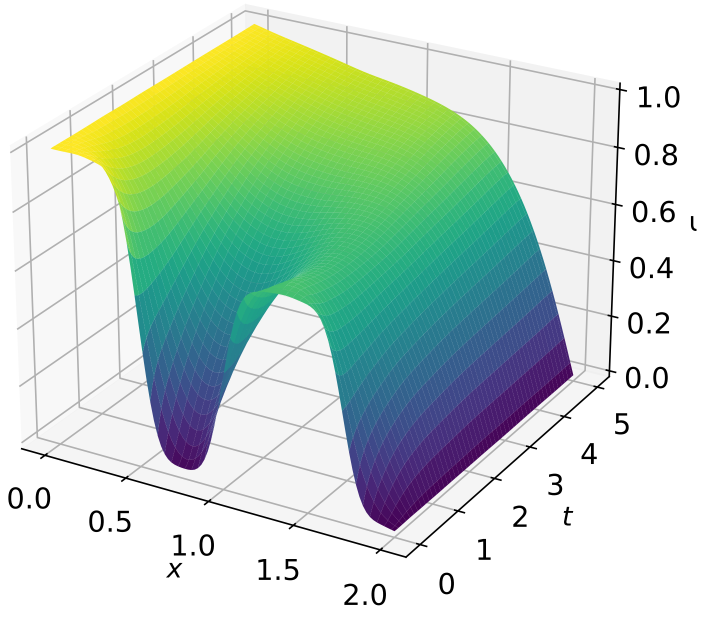

Figure 1: Solution to the nonlinear equation computed by PINNs.

We then compare the error obtained with our approach to that of an approximate solution computed using a classical numerical solver, as a function of the perturbation order p.

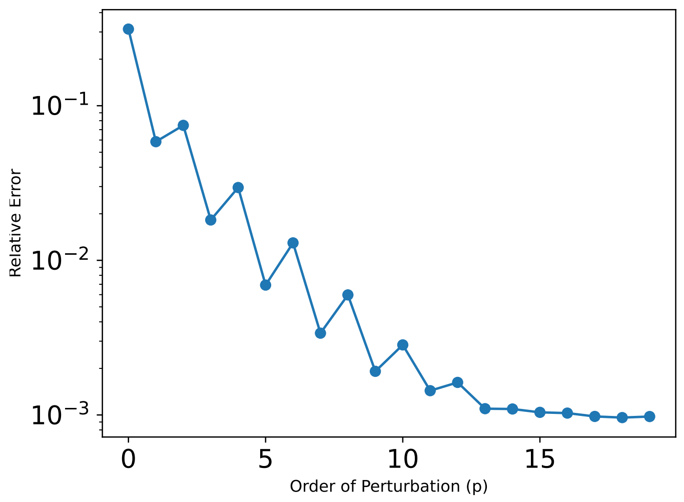

Figure 2: Relative error versus perturbation order p.

Our method solves this equation in $1.489 \cdot 10^{-1}$ seconds for $p = 10$, compared to $1.62 \cdot 10^{-1}$ seconds using a method-of-lines solver with SciPy's `solve_ivp`.

The solution can be computed for various n_1 and n_2. Increasing n_2 accelerates propagation toward equilibrium 1, while larger n_1 slows it. We observe similar precision and runtime for the fourth set of conditions in Table 6.1. The relative errors are $1.1 \cdot 10^{-3}$, $1.2 \cdot 10^{-3}$, and $1.9 \cdot 10^{-2}$ respectively; plots are in Appendix 6.2. The slightly higher error in the third case stems from its larger ϵ (see Section 4).

To test robustness, we trained another Multi-Head PINN for the wave operator $\mathcal{D} = \frac{\partial^2}{\partial t^2} - c^2 \frac{\partial^2}{\partial x^2}$ and obtained similar results (Appendix 6.5).

4 Limitations and Extensions

The method's accuracy is sensitive to the choice of ϵ. As shown in Appendix 6.2, the error exhibits a sharp increase once ϵ exceeds a threshold that depends on solution amplitudes and forcing. Specifically, solutions with larger amplitudes require smaller ϵ values to maintain accuracy. A favorable aspect of the method is that the problematic range of ϵ values is easily identifiable in experiments, as solutions diverge when ϵ becomes too large. For higher-degree polynomials (degree > 10), even smaller ϵ values are required due to the rapid growth of perturbation terms.

This work lays the foundation for numerous extensions:

- The approach can transfer across related differential operators, for instance when the coefficients $a_\alpha(t)$ in Eq. 2 vary slightly, as demonstrated in [17, 11].
- The derivation naturally extends to polynomial terms involving u and its derivatives. In this case, the mathematical extension is straightforward: one defines the multi-variable polynomial and collects terms for each power of epsilon. The functions f_j now depend on previously computed solutions and their derivatives, which represents the additional requirement for the experimental setup. Testing this approach on perturbative Burgers equations constitutes a natural next step.

A key open challenge remains extending one-shot transfer learning to non-perturbative terms, which will likely require the development of new mathematical tools.

5 Conclusion

We developed an efficient method for solving nonlinear PDEs, extending the approach of [11] for ODEs. By combining perturbation theory with one-shot transfer learning, our method attains accuracy comparable to an optimized classical solver while adapting faster to new instances, positioning it between single-task PINNs and full-scale operator learners (e.g. Neural Operators in [18, 19, 20]), our model generalizes to diverse cases with a leaner backbone and minimal fine-tuning. As discussed in Section 4, several extensions are promising, including derivative-dependent nonlinearities and mild operator changes, which could broaden applicability to larger-scale physical systems.

6 Appendix

The code used to generate the results of this study is publicly available on this GitHub: https://github.com/Sam147258369/NeurIPS-2025-Submission-Code

6.1 Equations used

We define a function used as an initial condition:

$$h(x) = \frac{1}{1+e^{20(x-0.5)}} + 0.7\left(\frac{1}{1+e^{-20(x-1)}} + \frac{1}{1+e^{-20(x-1.5)}}\right),$$

a smooth approximation of a discontinuous function equal to 1 for $x < 0.5$, equal to 0.7 for $1 < x < 1.5$, and 0 elsewhere. This is the same type of condition as in [16], but PINNs struggle to learn discontinuous functions.

Figure	ϵ	IC	BC Left	BC Right	Forcing function
1	0.5	$\psi(0,x) = h(x)$	$\psi(t,0) = 1$	$\psi(t,2) = 0$	$f(t,x) = 0$
3	0.1	$\psi(0,x) = h(x)$	$\psi(t,0) = 1$	$\psi(t,2) = 0$	$f(t,x) = 0$
4	0.1	$\psi(0,x) = h(x)$	$\psi(t,0) = 1$	$\psi(t,2) = 0$	$f(t,x) = 0$
5	0.3	$\psi(0,x) = 1 + 0.5x\sin(2\pi x)$	$\psi(t,0) = e^{-t}$	$\psi(t,2) = 1 + \sin(t)$	$f(t,x) = \sin(2t)\cos(3x)$

6.2 Graphs of various solutions

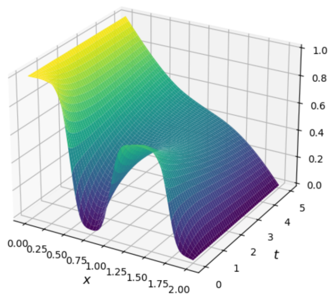
Figure 3: PINNs solution for $n_1 = 10, n_2 = 1$.

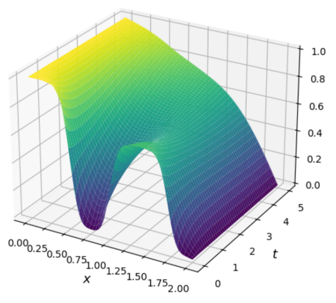
Figure 4: PINNs solution for $n_1 = 1, n_2 = 10$.

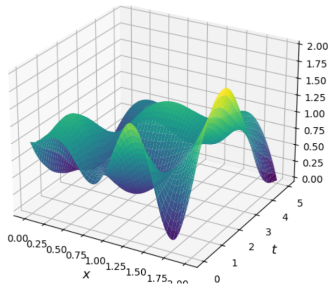
Figure 5: PINNs solution for $n_1 = 2$, $n_2 = 1$ with various conditions.

6.3 Graphs of error

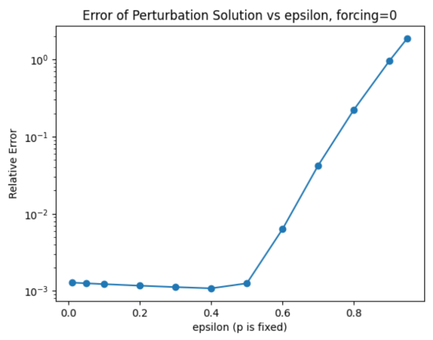

Figure 6: Relative error for different ϵ, $f = 0$.

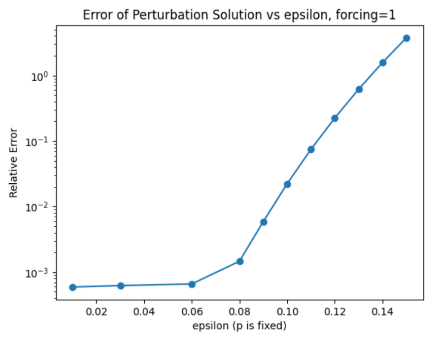

Figure 7: Relative error for different ϵ, $f = 1$.

6.4 Training Details

The following diagram illustrates the overall architecture and training process:

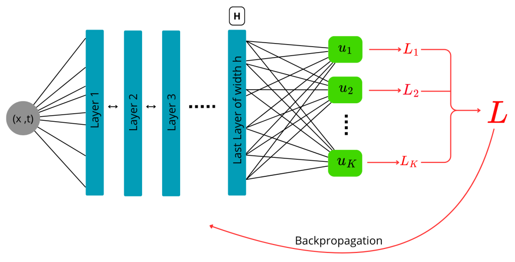

Figure 8: Architecture and process of the MH-PINN.

The model was trained on an **Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz** processor with **8 GB of RAM**. The network was optimized using PyTorch's Adam optimizer for 50,000 epochs with an initial learning rate of 1×10^{-4}. A scheduler reduced the learning rate by a factor of 0.975 every 1,000 iterations. Each iteration used 100 randomly sampled points from the domain, generated via an equally spaced grid with added noise.

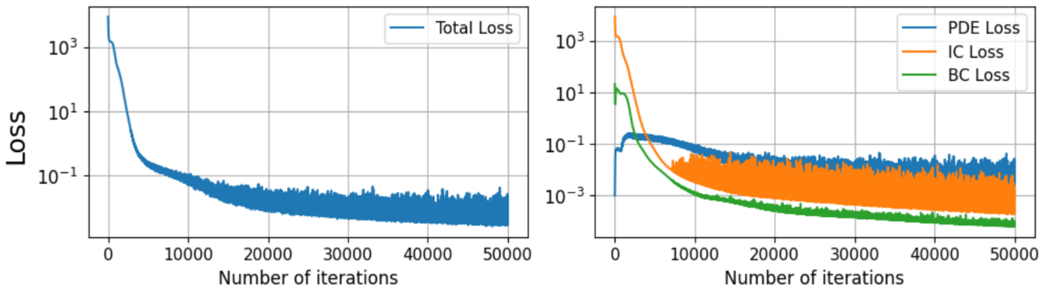

Figure 9: Total loss and components over iterations. The final total loss was 5.7×10^{-3}.

6.5 Results on the wave equation

For a nonlinear equation of the form

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} + \epsilon P(u) = f(x,t)$$

with parameters:

$$\begin{aligned}
&\text{(IC) } u(x,0) = 0, \quad \frac{\partial u}{\partial t}(x,0) = 1, \\
&\text{(BC) } u(0,t) = 0, \quad u(2,t) = 0, \\
&\text{(Forcing) } f(x,t) = 0, \\
&\text{(Polynomial) } P(u) = u - \frac{1}{6}u^3, \quad \epsilon = 0.75, \quad c = 1,
\end{aligned}$$

we obtain the following error versus perturbation order p:

With the corresponding solution for $p = 12$:

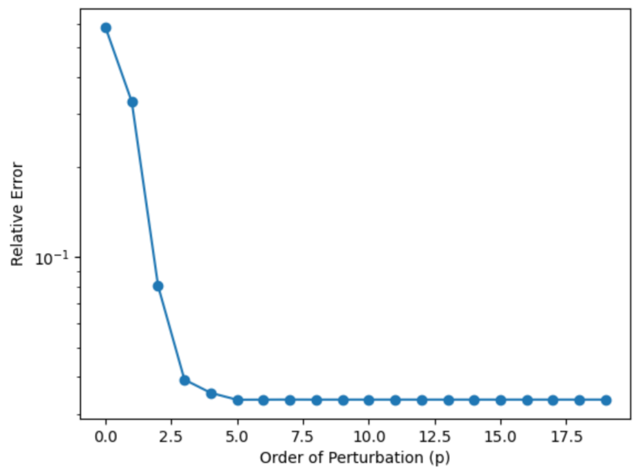

Figure 10: Relative error versus perturbation order for the modified wave equation.

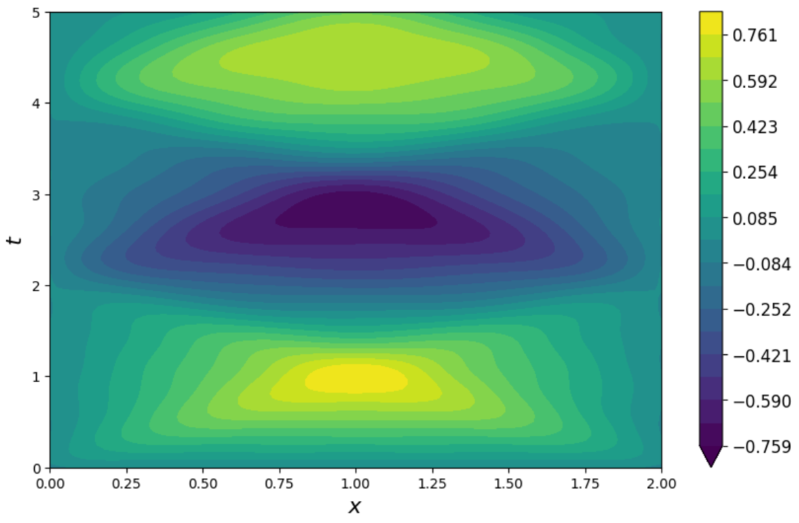

Figure 11: Solution for the perturbed wave equation.

References

[1] J.C. Butcher. “A history of Runge-Kutta methods”. In: *Applied Numerical Mathematics* 20.3 (1996), pp. 247–260. ISSN: 0168-9274. DOI: `https://doi.org/10.1016/0168-9274(95)00108-5`. URL: `https://www.sciencedirect.com/science/article/pii/0168927495001085`.

[2] Patrick Ciarlet and Éric Lunéville. *La méthode des éléments finis : de la théorie à la pratique.* 2009, p. 194. URL: `https://inria.hal.science/hal-04039611`.

[3] I.E. Lagaris, A. Likas, and D.I. Fotiadis. “Artificial neural networks for solving ordinary and partial differential equations”. In: *IEEE Transactions on Neural Networks* 9.5 (1998), pp. 987–1000. ISSN: 1045-9227. DOI: `10.1109/72.712178`. URL: `http://dx.doi.org/10.1109/72.712178`.

[4] Jian Cheng Wong et al. *Evolutionary Optimization of Physics-Informed Neural Networks: Evo-PINN Frontiers and Opportunities*. 2025. arXiv: `2501.06572 [cs.NE]`. URL: `https://arxiv.org/abs/2501.06572`.

[5] 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”. In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN:

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

[6] Justin Sirignano and Konstantinos Spiliopoulos. "DGM: A deep learning algorithm for solving partial differential equations". In: *Journal of Computational Physics* 375 (2018), pp. 1339–1364. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.08.029. URL: https://www.sciencedirect.com/science/article/pii/S0021999118305527.

[7] Aditi S. Krishnapriyan et al. *Characterizing possible failure modes in physics-informed neural networks*. 2021. arXiv: 2109.01050 [cs.LG]. URL: https://arxiv.org/abs/2109.01050.

[8] Cedric Flamant, Pavlos Protopapas, and David Sondak. *Solving Differential Equations Using Neural Network Solution Bundles*. 2020. arXiv: 2006.14372 [cs.LG]. URL: https://arxiv.org/abs/2006.14372.

[9] Zongren Zou and George Em Karniadakis. *L-HYDRA: Multi-Head Physics-Informed Neural Networks*. 2023. arXiv: 2301.02152 [cs.LG]. URL: https://arxiv.org/abs/2301.02152.

[10] Shaan Desai et al. *One-Shot Transfer Learning of Physics-Informed Neural Networks*. 2022. arXiv: 2110.11286 [cs.LG]. URL: https://arxiv.org/abs/2110.11286.

[11] Wanzhou Lei, Pavlos Protopapas, and Joy Parikh. *One-Shot Transfer Learning for Nonlinear ODEs*. 2023. arXiv: 2311.14931 [cs.LG]. URL: https://arxiv.org/abs/2311.14931.

[12] F. A. Howes. "Introduction to Perturbation Techniques (Ali Hasan Nayfeh)". In: *Wiley-Interscience Publication* (1982), pp. 135–137. DOI: 10.1137/1024080. eprint: https://doi.org/10.1137/1024080. URL: https://doi.org/10.1137/1024080.

[13] Grégoire Nadin and Luca Rossi. "Propagation phenomena for time heterogeneous KPP reaction–diffusion equations". In: *Journal de Mathématiques Pures et Appliquées* 98.6 (2012), pp. 633–653. ISSN: 0021-7824. DOI: https://doi.org/10.1016/j.matpur.2012.05.005. URL: https://www.sciencedirect.com/science/article/pii/S0021782412000591.

[14] Matthieu Alfaro. "Fujita blow up phenomena and hair trigger effect: The role of dispersal tails". In: *Annales de l'Institut Henri Poincaré C, Analyse non linéaire* 34.5 (2017), pp. 1309–1327. ISSN: 0294-1449. DOI: https://doi.org/10.1016/j.anihpc.2016.10.005. URL: https://www.sciencedirect.com/science/article/pii/S0294144916300749.

[15] Éric Brunet. "Some aspects of the Fisher-KPP equation and the branching Brownian motion". Accreditation to supervise research. UPMC, Nov. 2016. URL: https://theses.hal.science/tel-01417420.

[16] Lloyd N. Trefethen and Kristine Embree. "THE (UNFINISHED) PDE COFFEE TABLE BOOK". Chapter of an unpublished book on PDEs. Oxford university, 2021. URL: https://people.maths.ox.ac.uk/trefethen/pdectb.html.

[17] Emilien Seiler, Wanzhou Lei, and Pavlos Protopapas. *Stiff Transfer Learning for Physics-Informed Neural Networks*. 2025. arXiv: 2501.17281 [cs.LG]. URL: https://arxiv.org/abs/2501.17281.

[18] Nikola Kovachki et al. "Neural operator: learning maps between function spaces with applications to PDEs". In: *J. Mach. Learn. Res.* (Jan. 2023).

[19] Zongyi Li et al. *Fourier Neural Operator for Parametric Partial Differential Equations*. 2021. arXiv: 2010.08895 [cs.LG]. URL: https://arxiv.org/abs/2010.08895.

[20] Lu Lu et al. "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators". In: *Nature Machine Intelligence* 3.3 (Mar. 2021), pp. 218–229. ISSN: 2522-5839. DOI: 10.1038/s42256-021-00302-5. URL: http://dx.doi.org/10.1038/s42256-021-00302-5.