Algebraformer: A Neural Approach to Linear Systems

Pietro Sittoni
Gran Sasso Science Institute
pietro.sittoni@gssi.it

Francesco Tudisco
University of Edinburgh & Miniml.AI
f.tudisco@ed.ac.uk

Abstract

Recent work in deep learning has opened new possibilities for solving classical algorithmic tasks using end-to-end learned models. In this work, we investigate the fundamental task of solving linear systems, particularly those that are ill-conditioned. Existing numerical methods for ill-conditioned systems often require careful parameter tuning, preconditioning, or domain-specific expertise to ensure accuracy and stability. In this work, we propose Algebraformer, a Transformer-based architecture that learns to solve linear systems end-to-end, even in the presence of severe ill-conditioning. Our model leverages a novel encoding scheme that enables efficient representation of matrix and vector inputs, with a memory complexity of $\mathcal{O}(n^2)$, supporting scalable inference. We demonstrate its effectiveness on application-driven linear problems, including interpolation tasks from spectral methods for boundary value problems and acceleration of the Newton method. Algebraformer achieves competitive accuracy with significantly lower computational overhead at test time, demonstrating that general-purpose neural architectures can effectively reduce complexity in traditional scientific computing pipelines.

1 Introduction

Linear systems are ubiquitous in the physical sciences. Many fundamental tasks in physics, from solving inverse problems [29] to performing interpolation [46], or from discretizing and simulating differential equations [10, 69], ultimately reduce to solving a system of linear equations. Yet, while the theory behind linear solvers is well established, the practical numerical solution of *ill-conditioned* systems remains notoriously challenging. These systems are extremely sensitive to input perturbations and often lead to numerical instability, making their resolution highly dependent on problem-specific choices such as solver type, preconditioner design, and regularization strategy. As a result, even standard scientific computing libraries may yield poor results unless the system structure is explicitly accounted for and, in general, approaching these problems requires substantial expertise in numerical linear algebra, creating a barrier to accessible and robust solutions.

Over the decades, three main classes of methods have been developed. *Direct methods* (LU, Cholesky, QR [16]) are accurate but computationally prohibitive for large or sparse systems. *Iterative solvers* (Krylov methods [70], Jacobi, Gauss-Seidel, SOR [58]) scale better but require tuning and often fail under ill-conditioning. *Randomized algorithms* [49, 64] offer scalability but remain sensitive to conditioning. In all cases, the condition number is central, motivating preconditioning [13, 51, 3, 7] and regularization [67, 52, 17, 60, 26, 57, 8, 62], which improve stability but demand careful design.

In this work, we propose a new machine learning-based paradigm for solving linear systems that is simple, general, and effective even in the presence of severe ill-conditioning. Inspired by recent advances in machine learning, end-to-end solution for scientific computing, from solving PDEs [1, 6] to learning classical algorithms [72, 74, 77], we introduce **Algebraformer**, a Transformer-based architecture [71] that can solve linear systems via a single forward pass.

Machine Learning and the Physical Sciences Workshop, NeurIPS 2025.

Overall, **our main contributions are as follows**:

- We design a novel matrix encoding scheme that scales quadratically with the system size ($\mathcal{O}(n^2)$ memory), enabling Transformers to process linear systems of practical size;
- We evaluate Algebraformer on a broad suite of problem instances, highlighting its robustness to ill-conditioning and noise, as well as its ability to quickly solve linear systems.
- We demonstrate strong generalization across two use cases: spectral interpolation problems arising in solving boundary value problems and acceleration of the Newton method (we want to remark that the Newton method is widely used in physics, for example, in inverse problems [54, 59]). In the BVP experiment, we further demonstrated the adaptability of Algebraformer to previously unseen equations. The model was first pre-trained on a diffusion equation and then fine-tuned on a more complex one, consistently outperforming a model trained from scratch. This highlights its ability to transfer relevant knowledge and effectively generalize to datasets outside its original training distribution. The experiment concerning the Newton method is provided in Appendix 6.

While we do not aim to outperform finely tuned, task-specific numerical methods, our goal is to provide a general-purpose, plug-and-play solver that offers a compelling trade-off between accuracy, robustness, and ease of use. By sidestepping the need for handcrafted solver pipelines and extensive tuning, Algebraformer reduces the entry barrier to solving challenging linear systems, opening the door to broader adoption of ML-based numerical solvers in scientific and engineering workflows.

2 Related Work

Learning to accelerate numerical algorithms. There has been a surge of interest in learning-based approaches for improving classical solvers. Several works focus on accelerating iterative methods such as Conjugate Gradient for symmetric positive definite systems [42, 37, 81], or GMRES-type solvers for specific applications like the Poisson equation [45]. Others focus on learned preconditioning strategies: neural networks have been used to construct preconditioners that speed up convergence [28, 48, 66], or to optimize heuristics such as Jacobi and ILU variants [21, 63]. NeurKItt [47], for example, employs a neural operator to predict the invariant subspace of the system matrix and accelerate solution convergence. However, these approaches are primarily designed to enhance classical numerical pipelines. In contrast, our work takes a fundamentally different perspective: we aim to learn an end-to-end solver that directly outputs the solution to a linear system, bypassing the traditional iterative or decomposition-based steps altogether. We leave for the appendix an extended related works section appendix A.

3 Method

In this section, we present the design of Algebraformer, highlighting how matrices can be efficiently encoded into a sequence-to-sequence model that can be applied out of the box to solve a wide range of linear systems.

3.1 Algebraformer

Transformers have become a cornerstone in many areas of machine learning, excelling at processing one-dimensional sequences of token embeddings. However, adapting them to handle inherently two-dimensional data poses significant challenges. In [11], the authors address this by flattening a matrix of size $n \times n$ into a sequence, representing each matrix entry with symbolic tokens. They explore various encoding schemes: P10, P1000, P1999, and FP15, which correspond to using 5, 3, 2, and 1 token(s) per matrix entry, respectively. While this approach is innovative and has inspired further research, it also comes with notable limitations. Most critically, the memory complexity of the self-attention mechanism scales as $\mathcal{O}(n^4)$, making it infeasible to train even on moderate-to-small matrices. Additionally, the symbolic encoding introduces significant computational overhead during inference. Finally, the method reduces numerical precision, capping accuracy at 10^{-2}, which may be inadequate for applications requiring finer resolution.

Rather than flattening the matrix $A \in \mathbb{R}^{n\times n}$ into a one-dimensional sequence of length n^2, we encode the matrix and vector inputs more efficiently. Specifically, we represent each column of A separately and associate it with the corresponding entry in the right-hand side vector $b \in \mathbb{R}^n$. The resulting input

sequence consists of n tokens, where the i-th token is defined as $[a_i, b_i] \in \mathbb{R}^{n+1}$, with a_i denoting the i-th column of A, and b_i the i-th component of b. This structured encoding enables the model to process the linear system in a format that preserves its two-dimensional nature without incurring the excessive memory costs of full flattening, in fact, the requirements grow as $\mathcal{O}(n^2)$. A graphical overview of the model architecture is provided in fig. 1.

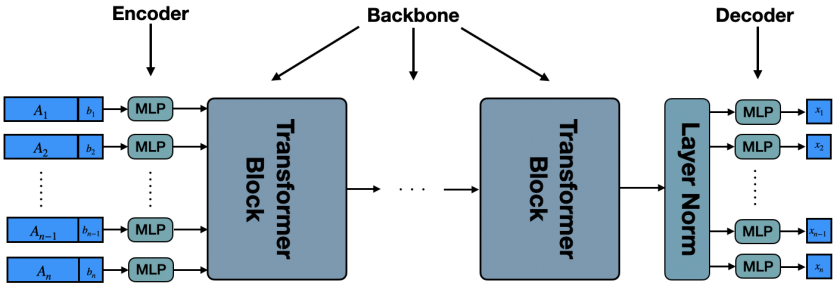

Figure 1: **Model overview.** We split the matrix A into column patches, to each patch we attach one component of the vector b, then we embed each patch into a decoder-only Transformer backbone. At the end, we decode the output of the backbone to output the vector solution x

For the backbone, we choose a decoder-only transformer, akin to the one used in [55]. The l-th block is represented by the equation eq. (1), where n_1 and n_2 denote the normalization layers. For normalization, we employ Layer Normalization [2], and we use a pre-norm strategy. Additionally, the normalization layer is applied to the output of the backbone:

$$\hat{x}_l = x_l + \mathrm{Attn}(n_1(x_l)), \qquad x_{l+1} = \hat{x}_l + \mathrm{MLP}(n_2(\hat{x}_l)). \tag{1}$$

To train our model, we use the following loss

$$\min_{\theta} \mathbb{E}\left[\mathcal{L}\left(\mathrm{Model}_\theta\left(\bar{A}\right), x\right)\right] \tag{2}$$

where x is the solution of the linear system, $\bar{A} = [A, b]$, and $\mathcal{L}(x, y) = \sum_i (x_i - y_i)^2$ is the MSE between the ground truth solution vector and the predicted vector.

4 Models details

We are interested in approximating a function defined as follows $(A, b) \mapsto x = A^{-1}b$. This mapping possesses two key properties: nonlinearity and smoothness. While the context differs from ours, [24] demonstrates that specific classes of nonlinear smooth functions can be learned using in-context examples. Their findings show that Transformers with approximately 10M parameters can effectively approximate function classes with comparable characteristics. Another motivation to keep the model size small is that we aim to have fast inference and be competitive with standard numerical solvers. To this end, we train Algebraformer with 12 Transformer blocks. Each block uses an embedding dimension of 256 and 8 attention heads. In the MLP sublayers, we project the embedding dimension by a factor of 4 and use the `GELU` as activation function [34]. Our model has 9.5M parameters.

We train two baselines: bidirectional LSTM and GRU. [11] showed that for linear algebra tasks, these RNNs perform comparably to transformers, though their study used very small matrices and did not focus on linear systems. To investigate further, we adopt an Algebraformer-like encoding for both, using 4 layers with 384-dimensional embeddings, yielding 12M parameters for the LSTM and 9M for the GRU. All three models employ a linear layer as encoder and decoder.

5 Experiments with spectral method for BVPs

In this section, we evaluate Algebraformer on linear systems arising from spectral methods for boundary value problems. We pre-train Algebraformer on a diffusion equation and, on this dataset, compare it with bidirectional LSTM and GRU models, as well as with standard numerical solvers (a direct method using LU decomposition with partial pivoting and least-squares solvers using SVD

and QR decomposition) under noisy conditions. Furthermore, we show that Algebraformer can be easily fine-tuned on more complex equations in a low-data regime, specifically, a reaction-diffusion equation and an advection-diffusion equation, comparing the pre-trained model with the one trained from scratch. We want to highlight the fact that for all the equations analyzed in this dataset, the resulting matrices have a condition number on the order of 10^5.

We defer the introduction to boundary value problems to appendix B, spectral methods to appendix C, the equations used to appendix D, data generation to appendix E, and training details to appendix F.

5.1 Results

In table 1, we can notice that Algebraformer outperforms both the LSTM and GRU baseline, on the dataset of the diffusion equation eq. (5).

Model	MSE
LSTM	0.00048211
GRU	0.00031371
Algebraformer	**0.00024131**

Table 1: MSE comparison on the diffusion equation.

The first two panels on the left of fig. 2 show performance on the fine-tuning task. In both cases, the fine-tuned model outperforms the model trained from scratch, demonstrating that pretraining on the simpler diffusion equation improves the model's ability to adapt to new and more complex equations. For eq. (6) (left), the fine-tuned model achieves an order-of-magnitude lower test error by epoch 50, a level that the scratch-trained model does not reach even after 1000 epochs. For eq. (7) (middle), the fine-tuned model matches the scratch-trained model's performance at epoch 1000 within just 100 epochs, and continues to improve until around epoch 500.

The right panel of fig. 2 assesses robustness to noise on the diffusion equation. Algebraformer maintains low relative MSE even with noisy test data, outperforming classical solvers such as direct methods and least-squares approaches (SVD, QR).

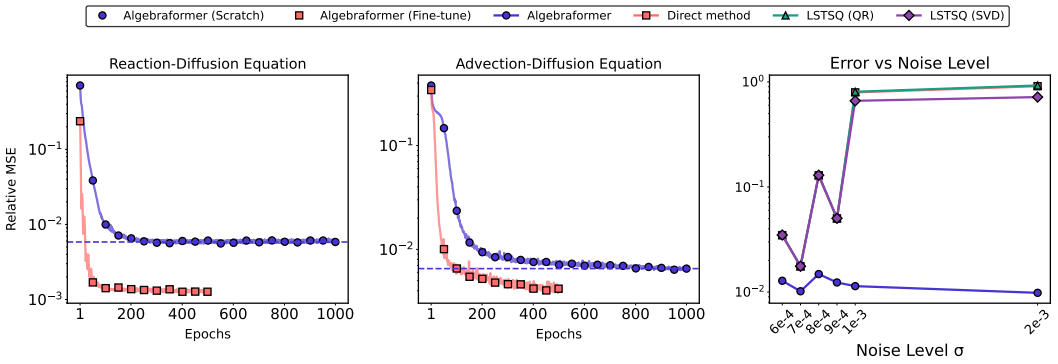

Figure 2: The two plots on the left show the relative MSE on the test set during training for (6) and (7). The plot on the right displays the relative MSE on the test set with noisy data for (5), where we compare Algebraformer with 3 different numerical methods.

6 Experiments with Newton method for nonlinear optimization

In this section, we focus on one more key setting where solving a linear system plays a crucial role: second-order optimization. In particular, we look at the Newton method [39]. One of the main drawbacks of Newton's method for the unconstrained optimization problem $\min_x f(x)$ is that, at each iteration, it requires solving the linear system $p_k = \mathcal{H}_f(x_k)^{-1}\nabla f(x_k)$ to compute the update $x_{k+1} = x_k + p_k$, from the current approximation x_k. Here $\mathcal{H}_f(x_k)$ is the Hessian and $\nabla f(x_k)$ is the gradient of the objective function f in x_k. While the update direction p_k is locally optimal, as it guarantees very fast quadratic local convergence, this procedure remains

computationally prohibitive in most applications as it requires computing the Hessian matrix and solving the associated linear system at each step. Approximated or inexact Newton alternatives are popular alternatives that significantly reduce the cost per iteration by approximating the Hessian and its inverse, albeit sacrificing on the convergence speed side [19, 20, 39]. Here we show that the method can be drastically sped up by using Algebraformer to replace the standard linear system solver step.

As a reference problem, we consider the following minimization task:

$$\min_{x \in \mathbb{R}^n} f(x) = \|Ax - b\|_p,$$

where $p \neq 2$, $A \in \mathbb{R}^{n \times m}$, and $b \in \mathbb{R}^m$. Choosing $p \neq 2$ ensures nonlinearity, and this is a well-studied problem in the literature [41, 14, 9]. A detailed technical introduction to Newton's method and the problem formulation is provided in appendix G.

As previously mentioned, constructing the Hessian and solving the corresponding linear system is costly. To address this, we propose using Algebraformer to take as input the current iterate x_k and the problem data (A and b), and directly predict the update direction p_k.

We conduct an experiment using matrices with 10^4 rows and 60 columns, with A and b uniformly distributed and normalized to unit norm. We conduct the experiments with $p = 6$, which is typically employed to reduce the influence of outliers. To reduce memory demand, due to the high amount of rows in the matrix A, instead of encoding the full matrix A and vector b, we encode the vector $A^\top b$ concatenated with the current iterate x_k. We generate 1,250 Newton iteration trajectories for the training set. By *trajectory*, we mean a sequence of Newton method steps $[x_0, x_1, \ldots, x_k]$ until convergence, i.e., until the norm of the residual $|f(x_k) - f(x_{k+1})|$ reaches a tolerance of 10^{-5}. We use 125 trajectories as the test set.

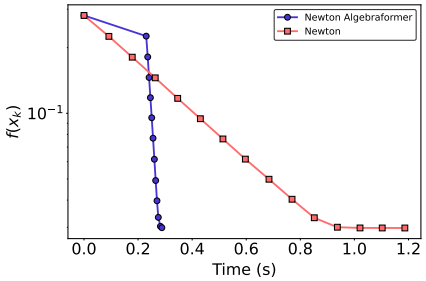

Figure 3: Time to convergence for Newton and accelerated Newton methods.

For the training of our model, we use the AdamW optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.95$, without any warm-up, and apply cosine decay to the learning rate, starting from 10^{-4} and decaying to 10^{-5}, and we train our model for 50 epochs. The architecture hyperparameters remain unchanged across experiments.

6.1 Results

After 50 epochs, our model can solve the linear system with a mean squared error (MSE) of 1×10^{-4} on the test set.

As shown in fig. 3, although the updates produced by our model are inexact, they enable convergence approximately four times faster than the classical Newton method. The plot also shows that both methods do converge; however, the objective function does not reach zero, as the linear system $Ax = b$ has no exact solution. In fact, the minimum value achieved is approximately 2×10^{-2}. Additionally, we observe that the first model evaluation takes noticeably longer than the subsequent ones. This initial latency is likely due to internal mechanisms within `PyTorch`.

7 Conclusion

We introduced Algebraformer, a Transformer-based architecture for solving linear systems. We show that Transformers can effectively solve linear algebra problems, and we propose a more scalable encoding framework than existing approaches in the literature. Experiments on BVP interpolation and Newton's method (section 6) demonstrate that Algebraformer offers greater robustness and speed-ups compared to standard numerical methods, highlighting its practical potential.

References

[1] Kamyar Azizzadenesheli, Nikola Kovachki, Zongyi Li, Miguel Liu-Schiaffini, Jean Kossaifi, and Anima Anandkumar. Neural operators for accelerating scientific simulations and design. *Nature Reviews Physics*, 6(5):320–328, 2024.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[3] Michele Benzi. Preconditioning techniques for large linear systems: a survey. *Journal of Computational Physics*, 182(2):418–477, 2002.

[4] Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales. In *International Conference on Machine Learning*, pages 936–945. Pmlr, 2021.

[5] Wilfried Bounsi, Borja Ibarz, Andrew Joseph Dudzik, Jessica B Hamrick, Larisa Markeeva, Alex Vitvitskyi, Razvan Pascanu, and Petar Veličković. Transformers meet neural algorithmic reasoners, 2025.

[6] Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural pde solvers. *arXiv preprint arXiv:2202.03376*, 2022.

[7] Achi Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation*, 31(138):333–390, 1977.

[8] Kristian Bredies, Karl Kunisch, and Thomas Pock. Total generalized variation. *SIAM Journal on Imaging Sciences*, 3(3):492–526, 2010.

[9] Alessandro Buccini and Lothar Reichel. An lp-lq minimization method with cross-validation for the restoration of impulse noise contaminated images. *Journal of Computational and Applied Mathematics*, 375:112824, 2020.

[10] John C Butcher. Implicit runge-kutta processes. *Mathematics of computation*, 18(85):50–64, 1964.

[11] Francois Charton. Linear algebra with transformers. *Transactions on Machine Learning Research*, 2022.

[12] François Charton, Amaury Hayat, and Guillaume Lample. Learning advanced mathematical computations from examples. *arXiv preprint arXiv:2006.06462*, 2020.

[13] Ke Chen. *Matrix preconditioning techniques and applications*. Number 19. Cambridge University Press, 2005.

[14] Julianne Chung and Silvia Gazzola. Flexible krylov methods for \ell_p regularization. *SIAM Journal on Scientific Computing*, 41(5):S149–S171, 2019.

[15] Alex Davies, Petar Veličković, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, Richard Tanburn, Peter Battaglia, Charles Blundell, András Juhász, et al. Advancing mathematics by guiding human intuition with ai. *Nature*, 600(7887):70–74, 2021.

[16] Timothy A Davis, Sivasankaran Rajamanickam, and Wissam M Sid-Lakhdar. A survey of direct methods for sparse linear systems. *Acta Numerica*, 25:383–566, 2016.

[17] Elisa Davoli, Irene Fonseca, and Pan Liu. Adaptive image processing: first order pde constraint regularizers and a bilevel training scheme. *arXiv preprint arXiv:1902.01122*, 2019.

[18] Andreea Deac, Petar Veličković, Ognjen Milinković, Pierre-Luc Bacon, Jian Tang, and Mladen Nikolic. Neural algorithmic reasoners are implicit planners. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[19] Ron S Dembo, Stanley C Eisenstat, and Trond Steihaug. Inexact newton methods. *SIAM Journal on Numerical Analysis*, 19(2):400–408, 1982.

[20] Stanley C Eisenstat and Homer F Walker. Choosing the forcing terms in an inexact newton method. *SIAM Journal on Scientific Computing*, 17(1):16–32, 1996.

[21] Goran Flegar, Hartwig Anzt, Terry Cojean, and Enrique S. Quintana-Ortí. Adaptive precision block-jacobi for high performance preconditioning in the ginkgo linear algebra software. *ACM Trans. Math. Softw.*, 47(2), April 2021.

[22] Simon Frieder, Julius Berner, Philipp Petersen, and Thomas Lukasiewicz. Large language models for mathematicians. *arXiv preprint arXiv:2312.04556*, 2023.

[23] Simon Frieder, Luca Pinchetti, Ryan-Rhys Griffiths, Tommaso Salvatori, Thomas Lukasiewicz, Philipp Petersen, and Julius Berner. Mathematical capabilities of chatgpt. *Advances in neural information processing systems*, 36:27699–27744, 2023.

[24] Shivam Garg, Dimitris Tsipras, Percy S Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598, 2022.

[25] Dobrik Georgiev Georgiev, JJ Wilson, Davide Buffelli, and Pietro Lio. Deep equilibrium algorithmic reasoning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[26] Guy Gilboa and Stanley Osher. Nonlocal operators with applications to image processing. *Multiscale Modeling & Simulation*, 7(3):1005–1028, 2009.

[27] Somdatta Goswami, Aniruddha Bora, Yue Yu, and George Em Karniadakis. Physics-informed deep neural operator networks. In *Machine learning in modeling and simulation: methods and applications*, pages 219–254. Springer, 2023.

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

[29] Charles Groetsch. *Linear Inverse Problems*, pages 3–41. Springer New York, New York, NY, 2011.

[30] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[31] Jayesh K Gupta and Johannes Brandstetter. Towards multi-spatiotemporal-scale generalized pde modeling. *arXiv preprint arXiv:2209.15616*, 2022.

[32] Christopher Hahn, Frederik Schmitt, Jens U Kreber, Markus N Rabe, and Bernd Finkbeiner. Teaching temporal logics to neural networks. *arXiv preprint arXiv:2003.04218*, 2020.

[33] Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, and Jun Zhu. Gnot: A general neural operator transformer for operator learning. In *International Conference on Machine Learning*, pages 12556–12569. PMLR, 2023.

[34] Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. 2016.

[35] Maximilian Herde, Bogdan Raonic, Tobias Rohner, Roger Käppeli, Roberto Molinaro, Emmanuel de Bézenac, and Siddhartha Mishra. Poseidon: Efficient foundation models for pdes. *Advances in Neural Information Processing Systems*, 37:72525–72624, 2024.

[36] Jonas Jürß and Dulhan Hansaja Jayalath. Recursive reasoning with neural networks, 2023.

[37] Ayano Kaneda, Osman Akar, Jingyu Chen, Victoria Alicia Trevino Kala, David Hyde, and Joseph Teran. A deep conjugate direction method for iteratively solving linear systems. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 15720–15736. PMLR, 23–29 Jul 2023.

[38] Subhash Kantamneni and Max Tegmark. Language models use trigonometry to do addition. *arXiv preprint arXiv:2502.00873*, 2025.

[39] C. T. Kelley. *Solving Nonlinear Equations with Newton's Method*. Society for Industrial and Applied Mathematics, Philadelphia, 2003.

[40] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. *arXiv preprint arXiv:1912.01412*, 2019.

[41] Alessandro Lanza, Serena Morigi, Lothar Reichel, and Fiorella Sgallari. A generalized krylov subspace method for \ell_p-\ell_q minimization. *SIAM Journal on Scientific Computing*, 37(5):S30–S50, 2015.

[42] Yichen Li, Peter Yichen Chen, Tao Du, and Wojciech Matusik. Learning preconditioner for conjugate gradient pde solvers, 2023.

[43] 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.

[44] Chih-Jen Lin, Rong-En Weng, and S Sathiya Keerthi. Trust region newton methods for large-scale logistic regression. *Journal of Machine Learning Research*, 9:627–650, 2008.

[45] Kevin Luna, Katherine Klymko, and Johannes P. Blaschke. Accelerating gmres with deep learning in real-time, 2021.

[46] Alessandra Lunardi. *Interpolation theory*, volume 16. Springer, 2018.

[47] Jian Luo, Jie Wang, Hong Wang, huanshuo dong, Zijie Geng, Hanzhu Chen, and Yufei Kuang. Neural krylov iteration for accelerating linear system solving. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[48] Ilay Luz, Meirav Galun, Haggai Maron, Ronen Basri, and Irad Yavneh. Learning algebraic multigrid using graph neural networks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 6489–6499. PMLR, 13–18 Jul 2020.

[49] Per-Gunnar Martinsson and Joel A Tropp. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numerica*, 29:403–572, 2020.

[50] Michael McCabe, Bruno Régaldo-Saint Blancard, Liam Holden Parker, Ruben Ohana, Miles Cranmer, Alberto Bietti, Michael Eickenberg, Siavash Golkar, Geraud Krawezik, Francois Lanusse, et al. Multiple physics pretraining for physical surrogate models. *arXiv preprint arXiv:2310.02994*, 2023.

[51] J.A. Meijerink and H.A. Van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric m-matrix. *Mathematics of Computation*, 31(137):148–162, 1977.

[52] David L. Phillips. A technique for the numerical solution of certain integral equations of the first kind. *J. ACM*, 9(1):84–97, January 1962.

[53] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.

[54] Aleksey I Prilepko, Dmitry G Orlovsky, Igor A Vasin, et al. *Methods for solving inverse problems in mathematical physics*. CRC Press, 2000.

[55] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners.

[56] Bogdan Raonic, Roberto Molinaro, Tobias Rohner, Siddhartha Mishra, and Emmanuel de Bezenac. Convolutional neural operators. In *ICLR 2023 Workshop on Physics for Machine Learning*, 2023.

[57] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268, 1992.

[58] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[59] Pierre C Sabatier. Past and future of inverse problems. *Journal of Mathematical Physics*, 41(6):4082–4124, 2000.

[60] Matti Lassas Saksman, Samuli Siltanen, et al. Discretization-invariant bayesian inversion and besov space priors. *arXiv preprint arXiv:0901.4220*, 2009.

[61] Feng Shi, Chonghan Lee, Mohammad Khairul Bashar, Nikhil Shukla, Song-Chun Zhu, and Vijaykrishnan Narayanan. Transformer-based machine learning for fast sat solvers and logic synthesis. *arXiv preprint arXiv:2107.07116*, 2021.

[62] Zakhar Shumaylov, Jeremy Budd, Subhadip Mukherjee, and Carola-Bibiane Schönlieb. Weakly convex regularisers for inverse problems: Convergence of critical points and primal-dual optimisation. In *Forty-first International Conference on Machine Learning*.

[63] Rita Stanaityte. *ILU and Machine Learning Based Preconditioning For The Discretized Incompressible Navier-Stokes Equations*. PhD thesis, University of Houston, 2020.

[64] Thomas Strohmer and Roman Vershynin. A randomized kaczmarz algorithm with exponential convergence. *Journal of Fourier Analysis and Applications*, 15(2):262–278, 2009.

[65] Shashank Subramanian, Peter Harrington, Kurt Keutzer, Wahid Bhimji, Dmitriy Morozov, Michael W. Mahoney, and Amir Gholami. Towards foundation models for scientific machine learning: characterizing scaling and transfer behavior. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

[66] Ali Taghibakhshi, Scott MacLachlan, Luke Olson, and Matthew West. Optimization-based algebraic multigrid coarsening using reinforcement learning. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc.

[67] Andrei N Tikhonov. Solution of incorrectly formulated problems and the regularization method. *Sov Dok*, 4:1035–1038, 1963.

[68] Alasdair Tran, Alexander Mathews, Lexing Xie, and Cheng Soon Ong. Factorized fourier neural operators. *arXiv preprint arXiv:2111.13802*, 2021.

[69] Lloyd N Trefethen. *Spectral methods in MATLAB*. SIAM, 2000.

[70] Henk A Van der Vorst. *Iterative Krylov methods for large linear systems*. Number 13. Cambridge University Press, 2003.

[71] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[72] Petar Veličković, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Misha Dashevskiy, Raia Hadsell, and Charles Blundell. The CLRS algorithmic reasoning benchmark. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 22084–22102. PMLR, 17–23 Jul 2022.

[73] Petar Veličković, Alex Vitvitskyi, Larisa Markeeva, Borja Ibarz, Lars Buesing, Matej Balog, and Alexander Novikov. Amplifying human performance in combinatorial competitive programming, 2024.

[74] Petar Veličković, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms, 2019.

[75] Richard A Waltz, Jorge L Morales, Jorge Nocedal, and Dominique Orban. An interior algorithm for nonlinear optimization that combines line search and trust region steps. *Mathematical Programming*, 107(3):391–408, 2006.

[76] Gege Wen, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, and Sally M Benson. U-fno—an enhanced fourier neural operator-based deep-learning model for multiphase flow. *Advances in Water Resources*, 163:104180, 2022.

[77] Kaijia Xu and Petar Velickovic. Recurrent aggregators in neural algorithmic reasoning. *CoRR*, abs/2409.07154, 2024.

[78] Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. *arXiv preprint arXiv:2311.12424*, 2023.

[79] Zhanhong Ye, Xiang Huang, Leheng Chen, Hongsheng Liu, Zidong Wang, and Bin Dong. Pdeformer: Towards a foundation model for one-dimensional partial differential equations. *arXiv preprint arXiv:2402.12652*, 2024.

[80] Shizhuo Dylan Zhang, Curt Tigges, Stella Biderman, Maxim Raginsky, and Talia Ringer. Can transformers learn to solve problems recursively? *arXiv preprint arXiv:2305.14699*, 2023.

[81] Xuan Zhang, Limei Wang, Jacob Helwig, Youzhi Luo, Cong Fu, Yaochen Xie, Meng Liu, Yuchao Lin, Zhao Xu, Keqiang Yan, Keir Adams, Maurice Weiler, Xiner Li, Tianfan Fu, Yucheng Wang, Alex Strasser, Haiyang Yu, YuQing Xie, Xiang Fu, Shenglong Xu, Yi Liu, Yuanqi Du, Alexandra Saxton, Hongyi Ling, Hannah Lawrence, Hannes Stärk, Shurui Gui, Carl Edwards, Nicholas Gao, Adriana Ladera, Tailin Wu, Elyssa F. Hofgard, Aria Mansouri Tehrani, Rui Wang, Ameya Daigavane, Montgomery Bohde, Jerry Kurtin, Qian Huang, Tuong Phung, Minkai Xu, Chaitanya K. Joshi, Simon V. Mathis, Kamyar Azizzadenesheli, Ada Fang, Alán Aspuru-Guzik, Erik Bekkers, Michael Bronstein, Marinka Zitnik, Anima Anandkumar, Stefano Ermon, Pietro Liò, Rose Yu, Stephan Günnemann, Jure Leskovec, Heng Ji, Jimeng Sun, Regina Barzilay, Tommi Jaakkola, Connor W. Coley, Xiaoning Qian, Xiaofeng Qian, Tess Smidt, and Shuiwang Ji. Artificial intelligence for science in quantum, atomistic, and continuum systems, 2023.

A Extended related works

Transformers for mathematical tasks and algorithmic reasoning. Transformers have been extensively explored for symbolic manipulation and mathematical reasoning tasks, including equation solving and symbolic integration [40, 53, 32, 61, 4, 12]. More recently, large language models (LLMs) have demonstrated remarkable capabilities in arithmetic and logical reasoning [15, 23, 22, 30]. Studies have begun to shed light on the internal mechanisms behind this behavior: for example, [38] suggests that LLMs internally use trigonometric structures to perform operations like addition. In [80], it was shown that Transformers can emulate structurally recursive functions from input-output data, while [24, 78] examined how Transformers trained from scratch generalize to function classes in context. Another active line of work investigates how Transformers can emulate classical algorithms, leveraging recursion [25, 36], or integrating graph-based inductive biases [5]. These methods have been applied in diverse settings, from competitive programming tasks [73] to fixed-point iteration problems in reinforcement learning [18].

Neural PDE solvers. A growing body of work applies neural networks to solve partial differential equations (PDEs). Among the most prominent frameworks are neural operators [1], which aim to learn mappings between function spaces in a resolution-invariant fashion. Notable examples include the Fourier Neural Operator [43], Factorized FNO [68], U-Net FNO [76], Physics-Informed Neural Operator [27], GNOT [33], and Convolutional Neural Operator [56]. These models are particularly suited for modeling the infinite-dimensional solution operators of PDEs. Beyond operator learning, architectures like message-passing neural networks (MPNNs) and U-Nets have also proven effective in discretized PDE settings [6, 31]. More recently, foundation models for PDEs have emerged [50, 35, 79, 65], aiming to generalize across families of equations that arise in diverse physical domains.

Learning to accelerate numerical algorithms. There has been a surge of interest in learning-based approaches for improving classical solvers. Several works focus on accelerating iterative methods such as Conjugate Gradient for symmetric positive definite systems [42, 37, 81], or GMRES-type solvers for specific applications like the Poisson equation [45]. Others focus on learned preconditioning strategies: neural networks have been used to construct preconditioners that speed up convergence [28, 48, 66], or to optimize heuristics such as Jacobi and ILU variants [21, 63]. NeurKItt [47], for example, employs a neural operator to predict the invariant subspace of the system matrix and accelerate solution convergence.

B Boundary value problems and spectral methods

A boundary value problem is given by a differential equation of the form

$$\begin{cases} \mathcal{L}\,u(x) = f(x), & x \in \Omega, \\ B[u(x)] = 0, & x \in \partial\Omega, \end{cases} \tag{3}$$

where $\Omega \subset \mathbb{R}^d$ is a bounded domain with boundary $\partial\Omega$. The differential operator $\mathcal{L} : \mathcal{X} \to \mathcal{Y}$ acts between suitable functional spaces, with $u : \Omega \to \mathbb{R}$ denoting the unknown solution and $f : \Omega \to \mathbb{R}$ the prescribed source term. The boundary operator B imposes conditions on $\partial\Omega$.

Spectral methods are a widely used class of numerical techniques known for their exponential convergence and global, basis-coefficient representation of solutions [69]. In contrast, finite difference schemes (and related discretizations) typically offer only polynomial convergence and provide local approximations. However, finite difference methods often yield structured matrices, which are generally easier to handle than the dense matrices that arise in spectral methods. These observations motivate our approach: our model is agnostic to the matrix structure and can efficiently solve the resulting linear system, while preserving the key advantages of spectral methods, namely, exponential convergence and a polynomial representation of the solution.

Among spectral methods, a common choice involves Chebyshev differentiation matrices [69] (see appendix C). This formulation is particularly attractive, as it interprets the model as an oracle that outputs the node of a polynomial. Polynomials are mathematically tractable and interpretable objects, an essential advantage for applications in physics.

C Spectral Method and chebyshev differentiation matrix

This section is based on the book [69]. For a smooth function $u : [-1, 1] \to \mathbb{R}$ we employ a Chebyshev spectral interpolant

$$u_N(x) = \sum_{k=0}^{N} a_k T_k(x), \qquad T_k(\cos\theta) = \cos(k\theta),$$

sampling at the *Chebyshev–Gauss–Lobatto* nodes

$$x_j = \cos\Big(\tfrac{\pi j}{N}\Big), \qquad j = 0, \dots, N,$$

with data $u_j = u(x_j)$. The cardinal functions $\ell_j(x)$ satisfy $\ell_j(x_m) = \delta_{jm}$, so the derivative of the interpolant is

$$u'_N(x) = \sum_{j=0}^{N} u_j \, \ell'_j(x).$$

Evaluating at every node x_i yields the *Chebyshev differentiation matrix* $D \in \mathbb{R}^{(N+1)\times(N+1)}$ through

$$u'_N(x_i) = \sum_{j=0}^{N} D_{ij} u_j, \qquad D_{ij} := \ell'_j(x_i).$$

Closed-form entries Let $c_0 = c_N = 2$ and $c_j = 1$ for $1 \le j \le N-1$. Then

$$D_{ij} = \begin{cases} \dfrac{c_i}{c_j} \dfrac{(-1)^{i+j}}{x_i - x_j}, & i \neq j, \\ -\dfrac{x_i}{2(1-x_i^2)}, & 1 \le i \le N-1, \\ \dfrac{2N^2+1}{6}, & i = j = 0, \\ -\dfrac{2N^2+1}{6}, & i = j = N. \end{cases}$$

Spectral convergence theorem (Bernstein–Jackson–Clenshaw–Curtis). If u is analytic in a Bernstein ellipse E_ρ ($\rho > 1$) with foci ± 1, then for any integer $m \ge 0$

$$\max_{x\in[-1,1]} \big|u^{(m)}(x) - u_N^{(m)}(x)\big| \;\le\; C_m \, \frac{\rho^{-N}}{(\rho-1)^m}, \tag{4}$$

where C_m depends on u and ρ.

D Equations

We consider a sequence of three differential equations on the one-dimensional domain $[0, 7.5]$ with Dirichlet boundary conditions, aiming to model diffusion-dominated processes and extend to more complex physical phenomena.

We first train our model on a classical second-order elliptic equation describing diffusion in a heterogeneous medium:

$$-\nabla(K(x)\nabla u(x)) = f(x), \tag{5}$$

Commonly used in heat conduction, groundwater flow, and diffusion through materials with spatially varying diffusivity $K(x)$.

To assess generalization to more complex scenarios, we fine-tune the model on two variants of eq. (5). The first introduces an *reaction* term:

$$-\nabla(K(x)\nabla u(x)) + q(x)u(x) = f(x), \tag{6}$$

modelling phenomena such as heat loss, chemical reactions, or decay processes. The second variant adds an *advection* term:

$$-\nabla(K(x)\nabla u(x)) + \nabla(v(x)u(x)) = f(x), \tag{7}$$

capturing the transport of u by a velocity field $v(x)$, relevant in fluid dynamics and pollutant dispersion.

In all cases, the resulting matrices have a condition number on the order of 10^5. For the training details, we refer to appendix F and for the details regarding the dataset we refer to appendix E.

E Datasets for BVP

For eq. (5) the function K is

$$K(x) = 1 + \alpha \cos(2\pi\omega x),$$

where $\alpha \sim \mathrm{U}[0.25, 0.75]$ and $\omega \sim \mathrm{U}[0.01, 0.75]$, the function $K(x)$ models a spatially varying medium with periodic structure. The parameters α and ω control the amplitude and frequency of variation, allowing tunable complexity. It ensures smoothness and positivity, making it ideal for testing PDE solvers in heterogeneous settings.

On the other for, the source term $f(x)$ often represents external forcing, such as heat sources,

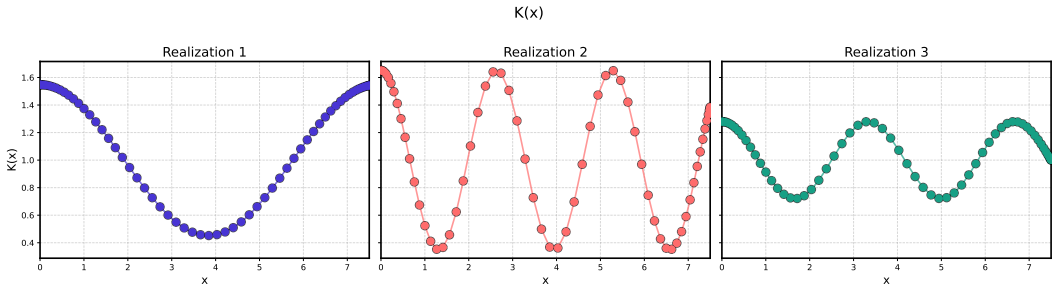

Figure 4: Three different samples of the function K

chemical reactions, or applied loads, distributed across space. The function

$$f(x) = (1 - \alpha) + \alpha \cdot r(x)$$

Can be interpreted as a blend between a uniform background source (the constant $1 - \alpha$) and a spatial random fluctuation $\alpha \cdot r(x)$, where $r(x)$ is a random field with unit average.

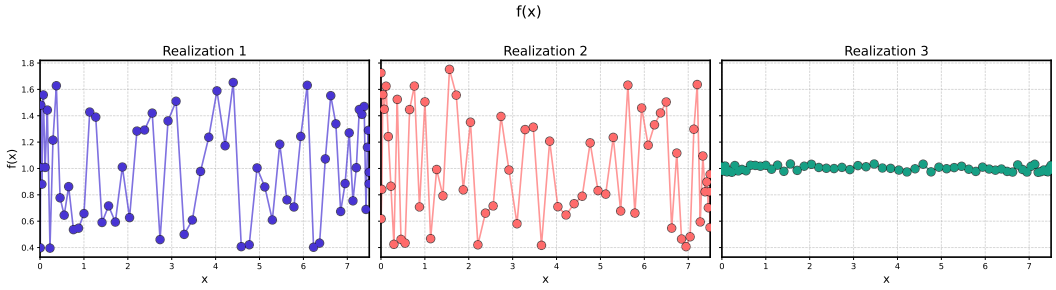

Figure 5: Three different samples of the function f

For the eq. (6) we choose the absorption term to be not null only in the interval $[3, 4.5]$ where we choose a constant $1/3$, thus as an attractive force that pushes to 0, the solution u. Instead, for the velocity field of eq. (7), we choose and uniform velocity field $v(x) = \alpha$ where $\alpha \sim U(-2, 2)$, based on the sign of the velocity field, the maximum of the solution is moved toward right or left.
We plot the three different solutions appendix E

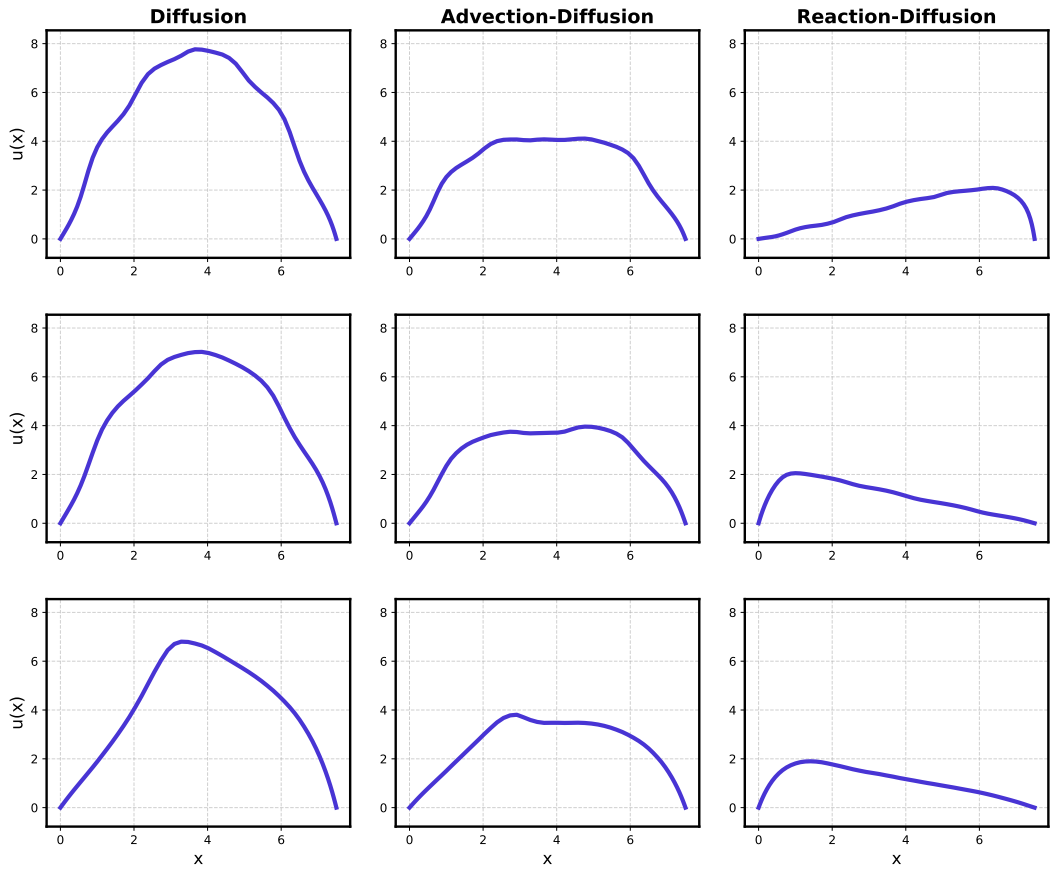

Figure 6: Three different samples for each equation

F Training details

For eqs. (5) to (7), we aim to predict the approximate solution passing through a basis of functions. The model receives two inputs: for eq. (5), the Chebyshev differentiation matrix based on the spatially variable coefficient K, and for the other two equations, the inputs are dependent on their respective coefficient terms. Furthermore, we concatenate the right-hand side $f(x)$ into the input matrix, as described in section 3.1. The training dataset consists of 50,000 samples, and the test set contains $5,000$ samples; further details on sample construction can be found in appendix E. For the more complex equations eq. (6) and eq. (7), we train with fewer samples to simulate real-world scenarios where the data for the simpler equations is more abundant. Specifically, we used 250 training observations for eq. (6) and $1,500$ for eq. (7), finding that the latter is the minimal number of observations for a good generalisation due to the complexity of the equation. We test the model on 5,000 samples for both equations.

For eq. (5), we train all models for 400 epochs using the AdamW optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.95$, applying cosine decay to the learning rate from 10^{-4} to 10^{-5}. For fine-tuning tasks on eqs. (6) and (7), we use the same learning rate schedule and train for 1,000 epochs when starting from scratch. For fine-tuned models, we use a fixed learning rate of 5×10^{-5} and train for half the epochs. In all experiments, we used a base of 64 elements, resulting in a matrix 64×64.

G Newton Method for $\min_x ||Ax - b||_p^p$

The Newton method is a classical iterative algorithm for solving nonlinear optimization problems [39]. It is particularly attractive due to its quadratic convergence rate, and it finds widespread application in areas such as interior-point methods for constrained optimization [75] and large-scale machine learning [44].

Consider the unconstrained optimization problem:

$$\min_{x \in \mathbb{R}^n} f(x).$$

At each iteration, the Newton method computes the next iterate as:

$$x_{k+1} = x_k - \mathcal{H}_f(x_k)^{-1} \nabla f(x_k),$$

where $\nabla f(x_k)$ is the gradient and $\mathcal{H}_f(x_k)$ is the Hessian matrix evaluated at x_k.

In practice, rather than computing the inverse of the Hessian explicitly, it is common to solve the linear system:

$$\mathcal{H}_f(x_k)\, p_k = \nabla f(x_k),$$

and update using $x_{k+1} = x_k - p_k$. This approach avoids direct inversion but still incurs high computational cost.

Both strategies, explicit matrix inversion and linear system solving, are expensive in terms of both time and resources, especially in high-dimensional settings. This motivates the search for techniques to accelerate the Newton method while preserving its fast convergence properties.

Algorithm 1 Newton's Method

procedure NEWTONMETHOD($f, \nabla f, \mathcal{H}_f, x_0$, tol, max_iter)
  $x \leftarrow x_0$
  for $k = 0$ to max_iter **do**
    $g \leftarrow \nabla f(x)$
    if $\|g\| <$ tol **then**
      return x
    end if
    $H \leftarrow \mathcal{H}_f(x)$
    Solve $Hp = g$ for p
    $x \leftarrow x - p$
  end for
  return x
end procedure

As discussed, Newton's method is a powerful tool for solving nonlinear optimization problems. We now consider applying it to the following objective:

$$f(x) = \|Ax - b\|_p^p,$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $p \geq 1$. This function arises in robust regression and data fitting applications, especially when $p \neq 2$.

Unlike the ℓ_2-norm case (where $p = 2$), the ℓ_p^p-norm leads to a nonlinear, non-quadratic objective for general p. As a result, the optimization problem becomes nonlinear and requires iterative methods like Newton's method for efficient minimization.

Let us define the residual vector $r(x) = Ax - b$. Then the objective can be written as:

$$f(x) = \sum_{i=1}^{m} |r_i(x)|^p.$$

Gradient. The gradient of $f(x)$ is given by:

$$\nabla f(x) = pA^\top \left(|r(x)|^{p-1} \odot \operatorname{sign}(r(x)) \right),$$

where $\odot$ denotes elementwise multiplication, and the operations $|\cdot|^{p-1}$ and $\operatorname{sign}(\cdot)$ are applied elementwise.

Hessian. The Hessian $\mathcal{H}_f(x)$ is:

$$\mathcal{H}_f(x) = p(p-1)A^\top \operatorname{diag}\left(|r(x)|^{p-2}\right) A,$$

which is valid for $p > 1$ and $r_i(x) \neq 0$. For values near zero, regularization or smoothing techniques are often needed to ensure numerical stability.