Towards Theoretical and Empirical Foundations of Machine Learning for Differential Equations

Tanya Marwah

February, 2025

CMU-ML-2025-102

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee

Andrej Risteski	Carnegie Mellon University (co-Chair)
Zachary Chase Lipton	Carnegie Mellon University (co-Chair)
Jianfeng Lu	Duke University
Maxim Raginsky	University of Illinois Urbana-Champaign

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy.

This research was funded by: Defense Advanced Research Projects Agency award FA8702-15-D-0002; Department of the Interior award D17PC00340; National Science Foundation award IIS2211907; grants from Ford, Highmark, the University of California; and a Fellowship from the Siebel Scholars Program.

Keywords: Partial Differential Equations, AI for Science, Machine Learning, Theory, Deep Equilibrium Models, State-Space Models, Graphs

Abstract

Recent advances in machine learning have propelled the use of data-driven methods in scientific discovery. In this we study the application of machine learning techniques to solve Partial Differential Equations (PDEs), which form fundamental building blocks in analyzing and describing various scientific phenomena, ranging from fluid dynamics to climate and weather forecasting and molecular dynamics. However, as the dimensionality of the system increases, the computational cost of simulating PDE solutions grows exponentially with the input dimension. Additionally, every new configuration of a PDE system necessitates rerunning the numerical solver from scratch, adding on to the computational challenges.

This thesis aims to theoretically and empirically investigate the conditions under which data-driven machine learning can effectively solve PDEs. It establishes conditions under which for specific classes of PDEs data driven machine learning techniques provide tangible benefits, especially in terms of reducing computational costs. Furthermore, it explores the architectural design space of using neural networks to approximate PDE solutions and fundamentally understands the choice of architecture that benefits downstream applications.

The thesis is divided into three parts. The first part containing Chapters 2, 3 and 4 introduces theoretical results that establish the representational capacity of neural networks for approximating solutions to complex PDEs. These chapters show that for certain families of PDEs the using a neural network can provably evade the curse of dimensionality. The second part includes Chapters 5, 6 and 7 and explores the architectural design choices of neural operators: neural networks that approximate solutions to an entire family of PDEs. We further use our insights towards designing efficient architectures for multi-physics models that can approximate solutions to multiple families of PDEs at once. The third part includes results on approximating graph structured data—which includes various scientific data such as molecules and PDEs on irregular meshes. In Chapter 8 we show how the state-space models based architectures such as Mamba generalizes to graph structured data. Finally, in Chapter 9 we introduce a theoretical results that establishes representational benefits of maintaining edge embeddings in graph neural networks.

To the journey and the people who make it worthwhile …

Acknowledgements

This thesis was not finished in isolation but with the support of many.

I want to begin by thanking my advisors, Andrej Risteski and Zack Lipton. They took me on as a student interested in working on machine learning theory and mathematics even though it was evident that I had no background on the topic. It is through their encouragement that I could work on a topic that was new for all of us. From them, I have not only learned the technical aspects of being a researcher but importantly what makes a good researcher and a collaborator—patience, perseverance and kindness.

Andrej is perhaps the single most hardworking and tenacious person I know. He has taught me the nuances involved in math and theory and shaped the way I approach problems and scientific method in general. There is lot to be learned from Andrej, for example, if he decides to take on a project he will give his hundred percent, else he will politely refuse to participate. He often comes up with key insights about proofs that totally change the course of a project—although admittedly this usually happens two days before a deadline. However, in the current state of machine learning research he hopes that his students slow down, focus on learning and recognize that research in theory takes time. Finally, no matter what he says I always listen to him—or almost always eventually do.

Zack has helped me build an empirical base for machine learning. He as an intuitive understanding of ML methods and how they work in practice and has the uncanny ability to grasp new things in seconds. From him, I have learned to be bold in choosing my work, to ask tough questions, and to challenge every assumption. He has also emphasized the importance of clear and concise scientific writing—since effective communication is key to conveying research and ideas. Finally, even though my PhD work didn't fully align with Zack's research interests, he never pressured me to change direction and always supported me in pursuing what I found meaningful.

I would also like to thank my committee members—Jianfeng Lu, and Maxim Raginsky. Jianfeng has acted as an oracle for me (and Andrej), often filling in the gaps in my understanding and also my proofs. Much has been said about his contributions across multiple fields, so I won't repeat them here. Instead, I want to highlight what an incredible mentor he is. After I spent nearly a year on Monge-Ampere problem only to realize that it couldn't be solved, he told me not to be discouraged but to recognize that every good math PhD has unsolved problems in their closet—meaning, in a way, I have officially earned my place, even tough I am not a math PhD student. Additionally, he is also very generous with his time, always answering my questions on Slack without fail.

I have greatly benefited from the work done by Max and his group throughout my PhD. His research consistently reflects deep thought, not only in tackling complex problems but also in the clarity with which results are communicated. His dedication to his field is also evident in the effort he puts into his course materials—many of us have benefited from his Stochastic Calculus notes. Beyond his research, Max exemplifies true academia through his blog posts and social media presence, engaging deeply with ideas, writing with clarity, and inspiring everyone by showing that being an academic is more than publishing—it is about learning, and sharing knowledge.

I would also like to thank many mentors that I have had the privilege to work with throughout my PhD journey. I would like to thank Jascha Sohl-Dickstein, Guy Gur-Ari, Behnam Neyshabur, Anders An-

dreassen and Yasaman Bahri for their mentorship during my internship at Google, and giving me an early glimpse of the LLM revolution we are in the midst of. I want to give a special thanks to Jascha, who has always taken the time to talk to me whenever I needed career advise. I also want to thank David Alvarez-Melis, Lester Mackey and Nicolo Fusi for providing me with an amazing internship experience at Microsoft, and for encouraging me to believe in myself and continue to work on more scientific applications of AI. At CMU, I like to specially thank Ameet Talwalkar for supporting me and encouraging me to scale up my research on PDEs, and also for his advise at the tail end of my PhD. Finally, I want to thank Albert Gu who has taught me how to think about various aspects of designing and training large models and also provided me with important guidance and help whenever I needed it.

I am also grateful to my undergraduate mentors—Vineeth N. Balasubramanian, Sumohana Channappayya, and Saumya Jana—who sparked my love for learning and set me on this path. Vineeth sir, in particular, has been a constant source of support and encouragement since my junior year in undergrad, when he took me on as a student and gave me the opportunity to work on a project in generative modeling.

I am grateful to the administrative staff at MLD, especially Diane Stidle and Laura Winter. Every student who graduates from MLD owes Diane a great deal of appreciation for ensuring we complete our forms, register for courses on time, and, most importantly, take care of ourselves. She is undoubtedly the reason for the warm, friendly, and supportive environment in MLD. A special thanks to Diane for also sharing photos of Sierra and Ruby with me. I also want to thank Laura for managing our schedules in the ACMI lab, ensuring rooms are booked on time, and handling the chaos of Zack's calendar.

Research is rarely a solo process, and that has certainly been the case for me. I want to start by thanking Ashwini Pokle, who became my collaborator at a time when I was struggling with motivation. Her enthusiasm kept me going, and through our work together, we learned a lot from each other. She has been the most dependable collaborator one could ask for. I am also grateful to Zhili Feng for working with me during our internship at Microsoft, sharing his deep understanding of LLMs, and always being willing to help whenever I needed it. His grasp of theory and extensive knowledge of the literature continue to inspire me. I want to thank Junhong Shen for collaborating with me on scaling my work—she has an incredible ability to execute research ideas effectively. I also want to thank Shanda Li, and appreciate him for always being up for working with me on exploring interesting but less conventional topics in our field with me, bringing genuine excitement to the process. I want to thank Ricardo Buitrago Ruiz, who helped me turn a small idea into a proper paper, but more importantly for nudging me to give matcha a try, which has now turned into an obsession. I also want to thank Misha Khodak for being such a fantastic collaborator. I can't help but wonder why we didn't start working together sooner. Finally, a huge thanks to Aakash Lahoti. He is hardworking, insightful, and thinks deeply about his research. I've seen his ability to execute ideas and debug complex models grow tremendously over time. I am also grateful to him for taking the lead on our work together towards the end, especially when I was busy with other things.

During my time at CMU, I have had the privilege of befriending so many incredible people—some from the research groups I've been part of and others from different departments. I have truly valued each and every one of them. I apologize in advance for any names I may have unintentionally overlooked. From Andrej's group I would like to thank: Niki Hasarati, Stephen Huan, Yuchen Li, Bingbin Liu, Elan Rosenfeld, and Anna Wei. From ACMI Lab I would like to thank Nil-jana Akpinar, Mel Andrews, Pravav Mani, Pratyush Maini and Danish Pruthi. Michael Feffer, Saurabh Garg, Shantanu Gupta, Daniel Jeong,

Simran Kaur, Divyansh Kaushik, Kundan Krishna, Leqi Liu, Pranav Mani, Pratyush Maini, Zachary Novack, Danish Pruthi, Manley Roberts, Jake Tyo, Tom Yan, and Helen Zhou. Special thanks to Saurabh and Kundan for always giving me wise and pragmatic advice, Pratyush for being willing to always help no matter what, Shantanu and Amrith Setlur for being my partners whenever I wanted a break, DK for interesting take on things and Danish for being the person I can always go to whenever I need clarity on things. I also want to thank my friends in MLD: Dhruv Malik for being my constant support since the beginning of our PhD, Euxhen Hasanaj for patiently explaining to me the details of all the developments in Biology, Youngseog Chung, Sachin Goyal, Daniel Jeong and Tom Yan for always being up for a chat. I also want to thank Lucio Dery for the short chats whenever we saw each other in the corridors, Asher Trockman for listening to be complain about things and Vaishnavh Nagarajan for encouraging me to talk to Andrej when I started my PhD—pretty sure he does not remember this.

I want to give a special thanks to the members of IAM-Lab for always inviting me to their gatherings and making me feel like an honorary member: Alex LaGrassa, Mark Lee, Tabitha Lee, Jacky Liang, Sarvesh Patil, Saumya Saxena, Shivam Vats, Erin Zhang, and Kevin Zhang. A shout out to Oliver Kroemer for bringing together such an incredible group of brilliant and kind students. This acknowledgment wouldn't be complete without a special thanks to Kevin for always showing up whenever I needed help—his support has made so many things easier for me. And of course, thanks for adopting Lychee, Kev!

I want to thank my friends that I made during my internships. Yasaman Razeghi for always being hopeful that I will take a break, and Eric Zelikman for brainstorming ideas. I also want to thank the wonderful cohort of people that I met at Microsoft: Fernanda Del La Torre for inspiring us all through setting a wonderful example through her work and life, Tobias Schroeder for making us laugh, Francisco Vargas for being so smart and answering all my stochastic calculus questions at his midnight, Pearl Yu for always being up for an adventure and Ruqing Xu for being so talented. And of course Zhili, but I have already thanked him!

I also want to thank my friends outside of CMU, who made sure that I stay at CMU and actually finish this thing. Gaurav Mittal for being my first collaborator, for pushing me through my masters and helping me whenever (be it in terms of much needed words of encouragement, or GPUs) I needed it. Aman Jain for being my friend since we were two years old, following me to CMU (seriously dude) and never running away. I want to thank Oshin Paranjape for her visits to Pittsburgh and New York that gave me a much needed break. I want to thank Rakshita Nagalla for always encouraging me to take a chance on myself. Anshi Chitransh for being the sane person who makes sure I am taking care of myself—since we were six years old. Manish Chandra Reddy for providing me a much needed respite from work and making me laugh whenever we talked. I want to thank Himangi Mittal (even though she was in Pittsburgh) for always inspiring me through her hard work. I also want to thank Cyril Zhang, who towards the end of my PhD provided me with much needed perspective towards life and what to aim for. Finally to Annapurna Kala, who was also doing her PhD at Johns Hopkins at the same time, we spoke almost every day, and kept each other going, and remained the one unwavering source of support throughout this journey.

Now comes the time for acknowledging the most special, wonderfully kind, strong, hardworking, inspiring group of women in MLD, that turned this entire PhD journey into a wholesome experience. They have all, in their unique ways pushed me to keep going. I want to begin by thanking the girls from my cohort: Arundhati Bannerjee, Bingin Liu, Yusha Liu, Stephanie Milani and Ashwini Pokle. Arundhati

for laughing at my jokes and getting my humour, Bingbin for being the source of sunshine in my life, Yusha for being Yusha (she is amazing, and I hope she knows that), Steph for perhaps being the wisest of the group and encouraging all of us to speak up when we should, and Ashwini for making me laugh so much (and for also cooking so well!). I want to thank Anna Bair for listening—maybe the only one who does—to me talk about the new thing that I am obsessing about, and Valerie Chen for making sure she checks up on me every time she is in New York. I want to thank Helen Zhou for being sweet, helpful and going on long walks with me. Finally I want to thank Leqi Liu for giving me much needed guidance and advice whenever I needed it. Words can not describe how much all of them have helped me, listened to my disillusioned ramblings and offered their unique perspective.

I could go on endlessly about the incredible people I've met, but no words could truly capture how much they mean to me. Before I thank my family, I want to give a special thanks to Aman Tyagi, who, throughout the pandemic, joined me in all my crazy kitchen experiments and was a willing participant—unlike Mohit who complained the entire time—in my quest to find the best wine. And a big thank you to Divyanshi Tyagi for bringing much needed laughter when it was needed most.

I want to thank my aunts—Neelam Bagga, Banita Bhatia, Sharmila Raheja, and Anju Sood—for showing me what it means to be both strong and compassionate, setting a standard to aspire to. I am also deeply grateful to my grandparents—Brij Mohan Marwah and Shashi Marwah, as well as Madan Lal Rajpal and Kaushalya Rani Rajpal. I try to live up to the example they have set for me and carry their values with me in everything I do.

I want to thank my parents, Sangeeta Marwah and Man Mohan Marwah. I talk to them every day—often twice—sharing everything with them. No matter what, things always feel better after our conversations. Their love and support is all I ever need. Also a huge thanks to Jingle—through her memories—and Nikku for bringing joy to my life. Finally, I want to thank my husband, Mohit Sharma, for being my constant source of encouragement, wisdom, stability, laughter and happiness since the day I met him. I can never thank him enough—but luckily, I have a lifetime!

Contents

List of Figures

List of Tables

Part I

Beginnings

1 Introduction

Recent advancements in Machine Learning (ML) and Artificial Intelligence (AI) have revolutionized modeling of language, vision, and video processing, inspiring a growing body of research leveraging these tools for scientific applications. This thesis takes a step towards building a fundamental understanding of how machine learning techniques apply to Partial Differential Equations (PDEs). Widely used to describe scientific phenomena, PDEs are central to applications in physics, computational chemistry, climate science, and weather modeling.

A partial differential equation (PDE) is a mathematical equation that describes the relationship between a (potentially high-dimensional) function $u \in \mathbb{R}^d$ and its partial derivatives. Unlike ordinary differential equations (ODEs), which involve derivatives with respect to a single variable, PDEs can include partial derivatives with respect to multiple dimensions. This additional complexity often makes PDEs significantly more challenging to solve. For example, many PDEs that model complex phenomena, such as the Navier-Stokes equations [Navier, 1821, 1822], lack closed-form solutions in most practical scenarios. As a result, computational approximation methods are frequently employed to solve these equations. For low-dimensional PDEs, widely used techniques include finite difference and finite element methods LeVeque [2007], which discretize the domain into a mesh. However, these methods scale poorly with increasing dimensionality, suffering from the so-called "curse of dimensionality". This has motivated a rapidly growing area of research in data-driven approaches to solving PDEs, where recent experiments have shown that for some PDEs deep networks can approximate solutions to high dimensional PDEs by seemingly escaping the curse of dimensionality.

A PDE takes the following general form,

$$F\left(t, u, \frac{\partial u}{\partial t}, \frac{\partial^2 u}{\partial t^2}, \cdots, \frac{\partial^n u}{\partial t^n}, \frac{\partial u}{\partial x_i}, \frac{\partial u^2}{\partial x_i^2}, \cdots\right) = 0, \quad t > 0, x \in \Omega \tag{1.1}$$

$$G\left(t, u, \frac{\partial u}{\partial t}, \frac{\partial^2 u}{\partial t^2}, \cdots, \frac{\partial^n u}{\partial t^n}, \frac{\partial u}{\partial x_i}, \frac{\partial u^2}{\partial x_i^2}, \cdots\right) = 0, \quad t > 0, x \in \partial\Omega \tag{1.2}$$

$$u(0, x) = u_0, \quad x \in \Omega, \tag{1.3}$$

where, $\Omega \subseteq \mathbb{R}^d$ refers to the domain over which the PDE is defined, and $x \in \mathbb{R}^d$ is the d-dimensional spatial variable and $t \in \mathbb{R}$ denotes time. Here F in Equation 1.1 represents a general functional relationship between the function $u \in \mathbb{R}^d$ with its partial derivatives over the domain Ω, and is referred to as the *governing equation*. The function G in Equation 1.2 defines the relationship over the boundary and is referred to as the *boundary condition*—and are typically less complex than the governing equation. Finally, if the PDE evolves over time, in order to uniquely define the behavior of the solution we need to define the *initial conditions* (Equation 1.3).

The goal of applying machine learning techniques to PDEs is either to approximate the solution u directly using a neural network or to approximate the operator F, which defines the overall relationship, using a neural network. Approaches that follow the former include Physics-Informed Neural Networks (PINNs) [Raissi et al., 2019] and the Deep Ritz Method [Weinan and Yu, 2018]. These methods take as input points sampled from the domain and train the neural network by minimizing a loss function that enforces the constraints derived from the governing equations, boundary conditions, and initial conditions—typically using the L_2 norm.

While these techniques effectively leverage the function-approximation capabilities of neural networks, they are limited to solving one specific PDE at a time. Recently, operator learning frameworks Chen and Chen [1995], Li et al. [2020a], McCabe et al. [2023] have gained significant attention. These methods use neural networks to approximate an infinite-dimensional operator that maps between functional spaces, enabling them to learn solutions for entire families (or multiple families) of PDEs simultaneously.

In this thesis, we take a step towards building a foundational understanding of the algorithms, methods and benchmarks used in the ML for PDEs literature. We take both the theoretical and empirical lens to answer important question such as representational and computational benefits of using neural networks, as well as empirically exploring the architectural design space of neural operators.

1.1 Theoretical Results

This section aims at developing a theoretical understanding of when and for what families of PDEs, can the solution be represented by a small neural network. Here, we show that for linear elliptic PDEs, if the coefficients of the PDE can be approximated by small neural networks, then the neural network approximating the solution to the PDE will have polynomial in the input dimension number of parameters. We then extend these results for a family of nonlinear elliptic PDEs by adopting the Barron norm of the function as the complexity measure. Here the results show that under certain conditions the Barron norm of the solution will have a polynomial dependence on the input dimension.

This part of the thesis are based on the following papers:

- Tanya Marwah, Zachary C. Lipton, and Anderj Risteski. “Parametric Complexity Bounds for Approximating PDEs with Neural Networks”. In *Advances in Neural Information Processing Systems*, 2021.

- Tanya Marwah, Zachary C. Lipton, Jianfeng Lu, and Anderj Risteski. “Neural Network Approximations of PDEs Beyond Linearity: A Representational Perspective”. In *International Conference on Machine Learning*, 2023.

1.2 Empirical Results

In this part we look into the empirical aspects of using machine learning and deep learning for modeling PDEs and explores the architectural design space of neural networks for approximating PDEs. First

we study the effects of utilizing appropriate inductive biases in the design of neural network architectures for approximating steady-state PDEs. Taking inspiration from classical numerical approaches of fast-converging Newton-like iterative schemes, our results show the benefits of very deep, but heavily weight-tied architectures as the appropriate architectural design choice for steady-state PDEs. We then study the benefits of explicitly maintaining memory for modeling time-dependent PDEs and introduce a framework that we refer to Memory Neural Operator (MemNO) which builds upon state space models and neural operators to effectively model memory. This effect more pronounced under scenarios of partial observability such as low-resolution grids and noise. We also shed light to how existing benchmarks are too simple and can often result in incomplete conclusions in the field. Finally, we will see some of our recent efforts towards building large multiphysics foundation models.

This part of the thesis are based on the following papers:

- Tanya Marwah, Ashwini Pokle, J. Zico Kolter, Zachary C. Lipton, Jianfeng Lu and Andrej Risteski. "Deep Equilibrium Based Neural Operators for Steady-State PDEs". In *Advances in Neural Information Processing Systems*, 2023.

- Ricard Buitrago Ruiz, Tanya Marwah, Albert Gu, and Andrej Risteski. "On the Benefits of Memory for Modeling Time-Dependent PDEs". In *International Conference on Learning Representations*, 2025.

- Junhong Shen, Tanya Marwah and Ameet Talwalkar. "UPS: Towards foundation models for pde solving via cross-modal adaptation". In *Transactions of Machine Learning*, 2024.

1.3 Graphs

In the final part of the thesis we take a detour and study Graph Neural Networks (GNNs) both from architectural and representational perspectives. This is driven by the need to study and design architectures capable of handling heterogeneous data—data sampled from irregular geometry, meshes, or other scientific datasets such as molecules. First, we introduce Chimera, a unified framework that mathematically generalizes state space models to incorporate the topological structure of data in a principled way. Chimera is topology aware and does not require any additional heuristics such as position encoding to incorporate the graph structure of the underlying data. Finally, we theoretically study and empirically validate the benefits of architectures that maintain and update edge embeddings when it comes to modeling graph structured data. We show that architectures that maintain edge embeddings almost always improve on their node-based counterparts—frequently significantly so in topologies that have "hub" nodes.

This part of the thesis are based on the following papers:

- Aakash Sunil Lahoti*, Tanya Marwah*, Ratish Pudupully, Albert Gu. "Chimera: State Space Models Beyond Sequences". *In Submission*.

- Dhruv Rahotgi, Tanya Marwah, Zachary C. Lipton, Jianfeng Lu, Ankur Moitra and Andrej Risteski. "Towards Characterizing the Value of Edge Embeddings in Graph Neural Networks". *In Submission*.

Part II

Machine Learning for Partial Differential Equations: Theoretical Underpinnings

2 Parametric Complexity Bounds for Approximating PDEs with Neural Networks

Abstract: *Recent experiments have shown that deep networks can approximate solutions to high-dimensional PDEs, seemingly escaping the curse of dimensionality. However, questions regarding the theoretical basis for such approximations, including the required network size remain open. In this paper, we investigate the representational power of neural networks for approximating solutions to linear elliptic PDEs with Dirichlet boundary conditions. We prove that when a PDE's coefficients are representable by small neural networks, the parameters required to approximate its solution scale polynomially with the input dimension d and proportionally to the parameter counts of the coefficient networks. To this end, we develop a proof technique that simulates gradient descent (in an appropriate Hilbert space) by growing a neural network architecture whose iterates each participate as sub-networks in their (slightly larger) successors, and converge to the solution of the PDE. We bound the size of the solution showing a polynomial dependence on d and no dependence on the volume of the domain.*

2.1 Introduction

A partial differential equation (PDE) relates a multivariate function defined over some domain to its partial derivatives. Typically, one's goal is to solve for the (unknown) function, often subject to additional constraints, such as the function's value on the boundary of the domain. PDEs are ubiquitous in both the natural and social sciences, where they model such diverse processes as heat diffusion [Crank and Nicolson, 1947, Özişik et al., 2017], fluid dynamics [Anderson and Wendt, 1995, Temam, 2001], and financial markets [Black and Scholes, 1973, Ehrhardt and Mickens, 2008]. Because most PDEs of interest lack closed-form solutions, computational approximation methods remain a vital and an active field of research [Ames, 2014]. For low-dimensional functions, dominant approaches include the finite differences and finite element methods [LeVeque, 2007], which discretize the domain. After partitioning the domain into a *mesh*, these methods solve for the function value at its vertices. However, these techniques scale exponentially with the input dimension, rendering them unsuitable for high-dimensional problems.

Following breakthroughs in deep learning for approximating high-dimensional functions in such diverse domains as computer vision [Krizhevsky et al., 2012, Radford et al., 2015] and natural language processing [Bahdanau et al., 2014, Devlin et al., 2018, Vaswani et al., 2017b], a burgeoning line of research leverages neural networks to approximate solutions to PDEs. This line of work has produced promising empirical results for common PDEs such as the Hamilton-Jacobi-Bellman and Black-Scholes equations

[Han et al., 2018, Grohs et al., 2018, Sirignano and Spiliopoulos, 2018]. Because they do not explicitly discretize the domain, and given their empirical success on high-dimensional problems, these methods *appear* not to suffer the curse of dimensionality. However, these methods are not well understood theoretically, leaving open questions about when they are applicable, what their performance depends on, and just how many parameters are required to approximate the solution to a given PDE.

Over the past three years, several theoretical works have investigated questions of representational power under various assumptions. Exploring a variety of settings, Kutyniok et al. [2019], Grohs et al. [2018], and Jentzen et al. [2018], proved that the number of parameters required to approximate a solution to a PDE exhibits a less than exponential dependence on the input dimension for some special parabolic PDEs that admit straightforward analysis. Grohs and Herrmann [2020] consider elliptic PDEs with Dirichlet boundary conditions. However, their rate depends on the volume of the domain, and thus can have an implicit exponential dependence on dimension (e.g., consider a hypercube with side length greater than one).

In this paper, we focus on linear elliptic PDEs with Dirichlet boundary conditions, which are prevalent in science and engineering (*e.g.,* , the Laplace and Poisson equations). Notably, linear elliptic PDEs define the steady state of processes like heat diffusion and fluid dynamics. Our work asks:

Question. *How many parameters suffice to approximate the solution to a linear elliptic PDE up to a specified level of precision using a neural network?*

We show that when the coefficients of the PDE are expressible as small neural networks (note that PDE coefficients are functions), the number of parameters required to approximate the PDE's solution is proportional to the number of parameters required to express the coefficients. Furthermore, we show that the number of parameters depends polynomially on the dimension and does not depend upon the volume of the domain.

2.2 Overview of Results

To begin, we formally define linear elliptic PDEs.

Definition 1 (Linear Elliptic PDE [Evans, 1998])**.** *Linear elliptic PDEs with Dirichlet boundary condition can be expressed in the following form:*

$$\begin{cases} (Lu)(x) \equiv (-div(A\nabla u) + cu)(x) = f(x), \forall x \in \Omega, \\ u(x) = 0, \forall x \in \partial\Omega, \end{cases}$$

where $\Omega \subset \mathbb{R}^d$ is a bounded open set with a boundary $\partial\Omega$. Further, for all $x \in \Omega$, $A : \Omega \to \mathbb{R}^{d\times d}$ is a matrix-valued function, s.t. $A(x) \succ 0$, and $c : \Omega \to \mathbb{R}$, s.t. $c(x) > 0$. [1]

We refer to A and c as the *coefficients* of the PDE. The divergence form in Definition 1 is one of two canonical ways to define a linear elliptic PDE [Evans, 1998] and is convenient for several technical reasons

[1] Here, div denotes the divergence operator. Given a vector field $F : \mathbb{R}^d \to \mathbb{R}^d$, $\text{div}(F) = \nabla \cdot F = \sum_{i=1}^{d} \frac{\partial F_i}{\partial x_i}$

(see Section 2.4). The Dirichlet boundary condition states that the solution takes a constant value (here 0) on the boundary $\partial\Omega$.

Our goal is to express the number of parameters required to approximate the solution of a PDE in terms of those required to approximate its coefficients A and c. Our key result shows:

Theorem (Informal)**.** *If the coefficients A, c and the function f are approximable by neural networks with at most N parameters, the solution $u^\star$ to the PDE in Definition 1 is approximable by a neural network with $O(poly(d)N)$ parameters.*

This result, formally expressed in Section 2.5, may help to explain the practical efficacy of neural networks in approximating solutions to high-dimensional PDEs with boundary conditions [Sirignano and Spiliopoulos, 2018, Li et al., 2020a]. To establish this result, we develop a constructive proof technique that simulates gradient descent (in an appropriate Hilbert space) through the very architecture of a neural network. Each iterate, given by a neural network, is subsumed into the (slightly larger) network representing the subsequent iterate. The key to our analysis is to bound both (i) the growth in network size across consecutive iterates; and (ii) the total number of iterates required.

2.3 Prior Work

Among the first papers to leverage neural networks to approximate solutions to PDEs with boundary conditions are Lagaris et al. [1998], Lagaris et al. [2000], and Malek and Beidokhti [2006]. However, these methods discretize the input space and thus are not suitable for high-dimensional input spaces. More recently, mesh-free neural network approaches have been proposed for high-dimensional PDEs [Han et al., 2018, Raissi et al., 2017, 2019], achieving impressive empirical results in various applications. Sirignano and Spiliopoulos [2018] design a loss function that penalizes failure to satisfy the PDE, training their network on minibatches sampled uniformly from the input domain. They also provide a universal approximation result, showing that for sufficiently regularized PDEs, there exists a multilayer network that approximates its solution. However, they do not comment on the complexity of the neural network or how it scales with the input dimension. Khoo et al. [2017] also prove universal approximation power, albeit with networks of size exponential in the input dimension. Recently, Grohs et al. [2018], Jentzen et al. [2018] provided a better-than-exponential dependence on the input dimension for some special parabolic PDEs, for which the simulating a PDE solver by a neural network is straightforward.

Several recent works [Bhattacharya et al., 2020, Kutyniok et al., 2019, Li et al., 2020b,a] show (experimentally) that a single neural network can solve for an entire family of PDEs. They approximate the map from a PDE's parameters to its solution, potentially avoiding the trouble of retraining for every set of coefficients. Among these, only Kutyniok et al. [2019] provides theoretical grounding. However, they assume the existence of a finite low-dimensional space with basis functions that can approximate this parametric map—and it is unclear when this would obtain. Our work proves the existence of such maps, under the assumption that the family of PDEs has coefficients described by neural networks with a fixed architecture (Section 2.7).

In the work most closely related to ours, Grohs and Herrmann [2020] provides approximation rates polynomial in the input dimension d for the Poisson equation (a special kind of linear elliptic PDE) with Dirichlet boundary conditions. They introduce a walk-on-the-sphere algorithm, which simulates a stochastic differential equation that can be used to solve a Poisson equation with Dirichlet boundary conditions (see, e.g., Oksendal [2013]'s Theorem 9.13). The rates provided in Grohs and Herrmann [2020] depend on the volume of the domain, and thus depend, implicitly, exponentially on the input dimension d. Our result considers the boundary condition for the PDE and is independent of the volume of the domain. Further, we note that our results are defined for a more general linear elliptic PDE, of which the Poisson equation is a special case.

2.4 Notation and Definitions

We now introduce several key concepts from PDEs and some notation. For any open set $\Omega \subset \mathbb{R}^d$, we denote its boundary by $\partial\Omega$ and denote its closure by $\bar{\Omega} := \Omega \cup \partial\Omega$. By $C^0(\Omega)$, we denote the space of real-valued continuous functions defined over the domain Ω. Furthermore, for $k \in \mathbb{N}$, a function g belongs to $C^k(\Omega)$ if all partial derivatives $\partial^\alpha g$ exist and are continuous for any multi-index α, such that $|\alpha| \leq k$. Finally, a function $g \in C^\infty(\Omega)$ if $g \in C^k(\Omega)$ for all $k \in \mathbb{N}$. Next, we define several relevant function spaces:

Definition 2. *For any $k \in \mathbb{N} \cup \{\infty\}$, $C_0^k(\Omega) := \{g : g \in C^k(\Omega), \overline{supp(g)} \subset \Omega\}$.*

Definition 3. *For a domain Ω, the function space $L^2(\Omega)$ consists of all functions $g : \Omega \to \mathbb{R}$, s.t. $\|g\|_2 < \infty$ where $\|g\|_2 = \left(\int_\Omega |g(x)|^2 dx\right)^{\frac{1}{2}}$. This function space is equipped with the inner product*

$$\langle g, h\rangle_2 = \int_\Omega g(x)h(x)dx.$$

Definition 4. *For a domain Ω and a function $g : \Omega \to \mathbb{R}$, the function space $L^\infty(\Omega)$ is defined analogously, where $\|g\|_{L^\infty(\Omega)} = \inf\{c \geq 0 : |g(x)| \leq c$ for almost all $x \in \Omega\}$.*

Definition 5. *For a domain Ω and $m \in \mathbb{N}$, we define the Hilbert space $H^m(\Omega)$ as*

$$H^m(\Omega) := \{g : \Omega \to \mathbb{R} : \partial^\alpha g \in L^2(\Omega),\ \forall \alpha \text{ s.t. } |\alpha| \leq m\}$$

Furthermore, $H^m(\Omega)$ is equipped with the inner product, $\langle g, h\rangle_{H^m(\Omega)} = \sum_{|\alpha|\leq m} \int_\Omega (\partial^\alpha g)(\partial^\alpha h)dx$ and the corresponding norm

$$\|g\|_{H^m(\Omega)} = \left(\sum_{|\alpha|\leq m} \|\partial^\alpha g\|^2_{L^2(\Omega)}\right)^{\frac{1}{2}}.$$

Definition 6. *The closure of $C_0^\infty(\Omega)$ in $H^m(\Omega)$ is denoted by $H_0^m(\Omega)$.*

Informally, $H_0^m(\Omega)$ is the set of functions belonging to $H^m(\Omega)$ that can be approximated by a sequence of functions $\phi_n \in C_0^\infty(\Omega)$. This also implies that if a function $g \in H_0^m(\Omega)$, then $g(x) = 0$ for all $x \in \partial\Omega$. This space (particularly with $m = 1$) is often useful when analyzing elliptic PDEs with Dirichlet boundary conditions.

Definition 7 (Weak Solution)**.** *Given the PDE in Definition 1, if $f \in 2$, then a function $u : \Omega \to \mathbb{R}$ solves the PDE in a weak sense if $u \in H_0^1(\Omega)$ and for all $v \in H_0^1(\Omega)$, we have*

$$\int_\Omega (A\nabla u \cdot \nabla v + cuv)dx = \int_\Omega fvdx \tag{2.1}$$

The left hand side of Equation 2.1 is also equal to $\langle Lu, v\rangle_2$ for all $u, v \in H_0^1(\Omega)$ (see Lemma 27), whereas, following the definition of the 2 norm, the right side is simply $\langle f, v\rangle_2$. Having introduced these preliminaries, we now introduce some important facts about linear PDEs that feature prominently in our analysis.

Proposition 1. *For the PDE in Definition 1, if $f \in 2$ the following hold:*

1. *The solution to Equation Equation 2.1 exists and is unique.*
2. *The weak solution is also the unique solution of the following minimization problem:*

$$u^\star = \operatorname*{argmin}_{v \in H_0^1(\Omega)} J(v) := \operatorname*{argmin}_{v \in H_0^1(\Omega)} \left\{ \frac{1}{2}\langle Lv, v\rangle_2 - \langle f, v\rangle_2 \right\}. \tag{2.2}$$

This proposition is standard (we include a proof in the Appendix, Section 10.1.1 for completeness) and states that there exists a unique solution to the PDE (referred to as $u^\star$), which is also the solution we get from the variational formulation in Equation 2.2. In this work, we introduce a sequence of functions that minimizes the loss in the variational formulation.

Definition 8 (Eigenvalues and Eigenfunctions, Evans [1998])**.** *Given an operator L, the tuples $(\lambda, \varphi)_{i=1}^\infty$, where $\lambda_i \in \mathbb{R}$ and $\varphi_i \in H_0^1(\Omega)$ are (eigenvalue, eigenfunction) pairs that satisfy $L\varphi = \lambda\varphi$, for all $x \in \Omega$. Since $\varphi \in H_0^1(\Omega)$, we know that $\varphi_{|\partial\Omega} = 0$. The eigenvalue can be written as*

$$\lambda_i = \inf_{u \in X_i} \frac{\langle Lu, u\rangle_2}{\|u\|_2^2}, \tag{2.3}$$

where $X_i := span\{\varphi_1, \ldots, \varphi_i\}^\perp = \{u \in H_0^1(\Omega) : \langle u, \varphi_j\rangle_2 = 0 \ \forall j \in \{1, \cdots, i\}\}$ and $0 < \lambda_1 \leq \lambda_2 \leq \cdots$. Furthermore, we define by Φ_k the span of the first k eigenfunctions of L, i.e., $\Phi_k := \operatorname{span}\{\varphi_1, \cdots, \varphi_k\}$.

We note that since the operator L is self-adjoint and elliptic (in particular, L^{-1} is compact), the eigenvalues are real and countable. Moreover, the eigenfunctions form an orthonormal basis of $H_0^1(\Omega)$ (see Evans [1998], Section 6.5).

2.5 Main Result

Before stating our results, we provide the formal assumptions on the PDEs of interest: **Assumptions:**

1. **Smoothness**: We assume that $\partial\Omega \in C^\infty$. We also assume that the coefficient $A \in \Omega \to \mathbb{R}^{d\times d}$ is a symmetric matrix-valued function, i.e., $A = (a_{ij}(x))$ and $a_{ij}(x) \in L^\infty(\Omega)$ for all $i, j \in [d]$ and the function $c \in L^\infty(\Omega)$ and $c(x) \geq \zeta > 0$ for all $x \in \Omega$. Furthermore, we assume that $a_{ij}, c \in C^\infty$. We define a constant

$$C := (2d^2+1)\max\left\{\max_{\alpha:|\alpha|\leq 3}\max_{i,j}\|\partial^\alpha a_{ij}\|_{L^\infty(\Omega)}, \max_{\alpha:|\alpha|\leq 2}\|\partial^\alpha c\|_{L^\infty(\Omega)}\right\}.$$

   Further, the function $f \in 2$ is also in C^∞ and the projection of f onto $\text{span}\{\varphi_1, \cdots, \varphi_k\}$ which we denote f_{span} satisfies for any multi-index α: $\|\partial^\alpha f - \partial^\alpha f_{\text{span}}\|_2 \leq \epsilon_{\text{span}}$. [2]

2. **Ellipticity**: There exist constants $M \geq m > 0$ such that, for all $x \in \Omega$ and $\xi \in \mathbb{R}^d$,

$$m\|\xi\|^2 \leq \sum_{i,j=1}^{d} a_{ij}(x)\xi_i\xi_j \leq M\|\xi\|^2.$$

3. **Neural network approximability**: There exist neural networks $\tilde{A}$ and $\tilde{c}$ with $N_A, N_c \in \mathbb{N}$ parameters, respectively, that approximate the functions A and c, i.e., $\|A - \tilde{A}\|_{L^\infty(\Omega)} \leq \epsilon_A$ and $\|c - \tilde{c}\|_{L^\infty(\Omega)} \leq \epsilon_c$, for small $\epsilon_A, \epsilon_c \geq 0$. We assume that for all $u \in H_0^1(\Omega)$ the operator $\tilde{L}$ defined as,

$$\tilde{L}u = -\text{div}_x(\tilde{A}\nabla u) + \tilde{c}u. \tag{2.4}$$

   is elliptic with $(\tilde{\lambda}_i, \tilde{\varphi}_i)_{i=1}^\infty$ (eigenvalue, eigenfunction) pairs. We also assume that there exists a neural network $f_{\text{nn}} \in C^\infty$ with $N_f \in N$ parameters such that for any multi-index α, $\|\partial^\alpha f - \partial^\alpha f_{\text{nn}}\|_2 \leq \epsilon_{\text{nn}}$. By Σ, we denote the set of all (infinitely differentiable) activation functions used by networks $\tilde{A}$, $\tilde{c}$, and f_{nn}. By Σ', we denote the set that contains all the n-th order derivatives of the activation functions in Σ, $\forall n \in \mathbb{N}_0$

Intuitively, ellipticity of L in a linear PDE $Lu = f$ is analogous to positive definiteness of a matrix $Q \in \mathbb{R}^d$ in a linear equation $Qx = k$, where $x, k \in \mathbb{R}^d$.

In (iii), we assume that the coefficients A and c, and the function f can be approximated by neural networks. While this is true for any smooth functions given sufficiently large N_A, N_c, N_f, our results are most interesting when these quantities are small (e.g. subexponential in the input dimension d). For many PDEs used in practice, approximating the coefficients using small neural networks is straightforward. For example, in heat diffusion (whose equilibrium is defined by a linear elliptic PDE) $A(x)$ defines the conductivity of the material at point x. If the conductivity is constant, then the coefficients can be written as neural networks with $O(1)$ parameters.

[2] Since $\partial\Omega \in C^\infty$ and the functions a_{ij}, c and f are all in C^∞, it follows from Nirenberg [1955] (Theorem, Section 5) the eigenfuntions of L are also C^∞. Hence, the function f_{span} is in C^∞ as well.

The part of assumption (i) that stipulates that f is close to f_{span} can be thought of as a smoothness condition on f. For instance, if $L = -\Delta$ (the Laplacian operator), the Dirichlet form satisfies $\frac{\langle Lu,u\rangle_2}{\|u\|_2^2} = \frac{\|\nabla u\|_2}{\|u\|_2}$, so eigenfunctions corresponding to higher eigenvalues tend to exhibit a higher degree of spikiness. The reader can also think of the eigenfunctions corresponding to larger k as Fourier basis functions corresponding to higher frequencies.

Finally, in (i) and (iii), while the requirement that the function pairs (f, f_{nn}) and (f, f_{span}) are close not only in their values, but their derivatives as well is a matter of analytical convenience, our key results do not necessarily depend on this precise assumption. Alternatively, we could replace this assumption with similar (but incomparable) conditions: e.g., we can also assume closeness of the values and a rapid decay of the L^2 norms of the derivatives. We require control over the derivatives because our method's gradient descent iterations involve repeatedly applying the operator L to f—which results in progressively higher derivatives.

We can now formally state our main result:

Theorem 1 (Main Theorem)**.** *Consider a linear elliptic PDE satisfying Assumptions (i)-(iii), and let $u^\star \in H_0^1(\Omega)$ denote its unique solution. If there exists a neural network $u_0 \in H_0^1(\Omega)$ with N_0 parameters, such that $\|u^\star - u_0\|_2 \leq R$, for some $R < \infty$, then for every $T \in \mathbb{N}$ such that $T \leq \frac{1}{20\min(\lambda_k,1)\delta}$, there exists a neural network u_T with size*

$$O\big(d^{2T}(N_0 + N_A) + T(N_f + N_c)\big)$$

such that $\|u^\star - u_T\|_2 \leq \epsilon + \tilde{\epsilon}$ where

$$\epsilon := \left(\frac{\tilde{\lambda}_k - \tilde{\lambda}_1}{\tilde{\lambda}_k + \tilde{\lambda}_1}\right)^T R,$$

$$\tilde{\epsilon} := \frac{\epsilon_{\text{span}}}{\lambda_1} + \frac{\delta}{\lambda_1}\frac{\|f\|_2}{\gamma - \delta} + \delta\|u^\star\|_2 + (\max\{1, T^2 C\eta\})^T\left(\epsilon_{\text{span}} + \epsilon_{nn} + 4\left(1 + \frac{\delta}{\gamma - \delta}\right)\lambda_k^T\|f\|_2\right),$$

and $\eta := \frac{2}{\tilde{\lambda}_1 + \tilde{\lambda}_k}$, $\delta := \max\left\{\frac{\epsilon_A}{m}, \frac{\epsilon_c}{\zeta}\right\}$. Furthermore, the activation functions used in u_T belong to the set $\Sigma \cup \Sigma' \cup \{\rho\}$ where $\rho(y) = y^2$ for all $y \in \mathbb{R}$ is the square activation function.

This theorem shows that given an initial neural network $u_0 \in H_0^1(\Omega)$ containing N_0 parameters, we can recover a neural network that is ϵ close to the unique solution $u^\star$. The number of parameters in u_ϵ depend on how close the initial estimate u_0 is to the solution $u^\star$, and N_0. This results in a trade-off, where better approximations may require more parameters, compared to a poorer approximation with fewer parameters.

Note that $\epsilon \to 0$ as $T \to \infty$, while $\tilde{\epsilon}$ is a "bias" error term that does not go to 0 as $T \to \infty$. The first three terms in the expression for $\tilde{\epsilon}$ result from bounding the difference between the solutions to the equations $Lu = f$ and $\tilde{L}u = f_{\text{span}}$, whereas the third term is due to difference between f and f_{nn} and the fact that our proof involves simulating the gradient descent updates with neural networks. Further, if the constant ζ is equal to 0 then the error term ϵ_c will also be 0, in which case the term δ will equal ϵ_A/m.

The fact that $\epsilon := \left(\frac{\tilde{\lambda}_k - \tilde{\lambda}_1}{\tilde{\lambda}_k + \tilde{\lambda}_1}\right)^T R$ comes from the fact that we are simulating T steps of a gradient descent-like procedure on a strongly convex loss. The parameters $\tilde{\lambda}_k$ and $\tilde{\lambda}_1$ can be thought of as the effective Lipschitz and strong-convexity constants of the loss. Finally, to give a sense of what R looks like, we show in Lemma 1 that if u_0 is initialized to be identically zero then $R \leq \frac{\|f\|_2}{\lambda_1}$.

Lemma 1. *If $u_0 = 0$, then $R \leq \frac{\|f\|_2}{\lambda_1}$.*

Proof. Given that u_0 is identically 0, the value of R in Theorem 1 equals $\|u^\star - u_0\|_2 = \|u^\star\|_2$ Using the inequality in (2), we have,

$$\begin{aligned} \|u^\star\|_2^2 &\leq \frac{\langle Lu^\star, u^\star \rangle}{\lambda_1} \\ &\leq \frac{1}{\lambda_1} \langle f, u^\star \rangle_2 \\ &\leq \frac{1}{\lambda_1} \|f\|_2 \|u^\star\|_2 \\ \implies \|u^\star\|_2 &\leq \frac{1}{\lambda_1} \|f\|_2 \end{aligned}$$

□

We make few remarks about the theorem statement:

Remark 1. *While we state our convergence results in 2 norm, our proof works for the $H_0^1(\Omega)$ norm as well. This is because in the space defined by the top-k eigenfunctions of the operator L, 2 and $H_0^1(\Omega)$ norm are equivalent (shown in Proposition 13). Further, note that even though we have assumed that $u^\star \in H_0^1(\Omega)$ is the unique solution of (2.1) from the boundary regularity condition, we have that $u^\star \in H^2(\Omega)$ (see Evans [1998], Chapter 6, Section 6.3). This ensures that the solution $u^\star$ is twice differentiable as well.*

Remark 2. *To get a sense of the scale of λ_1 and λ_k, when $L = -\Delta$ (the Laplacian operator), the eigenvalue $\lambda_1 = \inf_{u \in H_0^1(\Omega)} \frac{\|\nabla u\|_2}{\|u\|_2} = \frac{1}{C_p}$, where C_p is the Poincaré constant (see Theorem 14 in Appendix). For geometrically well-behaved sets Ω (e.g. convex sets with a strongly convex boundary, like a sphere), C_p is even dimension-independent. Further from the Weyl's law operator (Evans [1998], Section 6.5) we have*

$$\lim_{k \to \infty} \frac{\lambda_k^{d/2}}{k} = \frac{(2\pi)^d}{\text{vol}(\Omega)\alpha(d)}$$

where $\alpha(d)$ is the volume of a unit ball in d dimensions. So, if $\text{vol}(\Omega) \geq 1/\alpha(d)$, λ_k grows as $O(k^{2/d})$, which is a constant so long as $\log k L^2(\Omega) d$.

2.6 Proof of Main Result

First, we provide some intuition behind the proof, via an analogy between a uniformly elliptic operator and a positive definite matrix in linear algebra. We can think of finding the solution to the equation

$Lu = f$ for an elliptic L as analogous to finding the solution to the linear system of equations $Qx = k$, where Q is a $d \times d$ positive definite matrix, and x and k are d-dimensional vectors. One way to solve such a linear system is by minimizing the strongly convex function $\|Qx - b\|^2$ using gradient descent. Since the objective is strongly convex, after $O(\log(1/\epsilon))$ gradient steps, we reach an ϵ-optimal point in an l_2 sense.

Our proof uses a similar strategy. First, we show that for the operator L, we can define a sequence of functions that converge to an ϵ-optimal function approximation (in this case in the 2 norm) after $O(\log(1/\epsilon)$ steps—similar to the rate of convergence for strongly convex functions. Next, we inductively show that each iterate in the sequence can be approximated by a small neural network. More precisely, we show that given a bound on the size of the t-th iterate u_t, we can, in turn, upper bound the size of the $(t+1)$-th iterate u_{t+1} because the update transforming u_t to u_{t+1} can be simulated by a small neural network (Lemma 7). These iterations look roughly like $u_{t+1} \leftarrow u_t - \eta(Lu_t - f)$, and we use a "backpropagation" lemma (Lemma 8) which bounds the size of the derivative of a neural network.

2.6.1 Defining a Convergent Sequence

The rough idea is to perform gradient descent in 2 [Neuberger, 2009, Faragó and Karátson, 2001, 2002] to define a convergent sequence whose iterates converge to $u^\star$ in 2 norm (and following Remark 1, in $H_0^1(\Omega)$ as well). However, there are two obstacles to defining the iterates as simply $u_{t+1} \leftarrow u_t - \eta(Lu_t - f)$: (1) L is unbounded—so the standard way of choosing a step size for gradient descent (roughly the ratio of the minimum and maximum eigenvalues of L) would imply choosing a step size $\eta = 0$, and (2) L does not necessarily preserve the boundary conditions, so if we start with $u_t \in H_0^1(\Omega)$, it may be that $Lu_t - f$ does not even lie in $H_0^1(\Omega)$.

We resolve both issues by restricting the updates to the span of the first k eigenfunctions of L. More concretely, as shown in Lemma 2, if a function u in $\operatorname{span}\{\varphi_1, \cdots, \varphi_k\}$, then the function Lu will also lie in $\operatorname{span}\{\varphi_1, \cdots, \varphi_k\}$. We also show that within the span of the first k eigenfunctions, L is bounded (with maximum eigenvalue λ_k), and can therefore be viewed as an operator from $\operatorname{span}\{\varphi_1, \cdots, \varphi_k\}$ to $\operatorname{span}\{\varphi_1, \cdots, \varphi_k\}$. Further, we use f_{span} instead of f in our updates, which now have the form $u_{t+1} \leftarrow u_t - \eta(Lu_t - f_{\text{span}})$. Since f_{span} belongs to $\operatorname{span}\{\varphi_1, \cdots, \varphi_k\}$, for a u_t in $\operatorname{span}\{\varphi_1, \cdots, \varphi_k\}$ the next iterate u_{t+1} will now remain in $\operatorname{span}\{\varphi_1, \cdots, \varphi_k\}$. Continuing the matrix analogy, we can choose the usual step size of $\eta = \frac{2}{\lambda_1 + \lambda_k}$. Precisely, we show:

Lemma 2. *Let L be an elliptic operator. Then, for all $v \in \Phi_k$ it holds:*

1. $Lv \in \operatorname{span}\{\varphi_1, \cdots, \varphi_k\}$.
2. $\lambda_1 \|v\|_2 \leq \langle Lv, v\rangle_2 \leq \lambda_k \|v\|_2$
3. $\left\| \left(I - \frac{2}{\lambda_k + \lambda_k} L\right) u \right\|_2 \leq \frac{\lambda_k - \lambda_1}{\lambda_k + \lambda_1} \|u\|_2$

Proof. Writing $u \in \operatorname{span}\{\varphi_1, \cdots, \varphi_k\}$ as $u = \sum_i d_i \varphi_i$ where $d_i = \langle u, \varphi_i \rangle_2$, we have $Lu = \sum_{i=1}^k \lambda_i d_i \varphi_i$. Therefore $Lu \in \tilde{\Phi}_K$ and Lu lies in $H_0^1(\Omega)$, proving (1.).

Since $v \in \Phi_k$, we use the definition of eigenvalues in (2.3) to get,

$$\frac{\langle Lv, v\rangle_2}{\|v\|_2} \leq \sup_v \frac{\langle Lv, v\rangle_2}{\|v\|_2} = \lambda_k$$
$$\implies \langle Lv, v\rangle_2 \leq \lambda_k \|v\|_2^2$$

and similarly

$$\frac{\langle Lv, v\rangle_2}{\|v\|_2} \geq \inf_v \frac{\langle Lv, v\rangle_2}{\|v\|_2} = \lambda_1$$
$$\implies \langle Lv, v\rangle_2 \geq \lambda_1 \|v\|_2^2$$

In order to prove (2.) let us first denote $\bar{L} := \left(I - \frac{2}{\lambda_k+\lambda_1}L\right)$. Note if φ is an eigenfunction of L with corresponding eigenvalue λ, it is also an eigenfunction of $\bar{L}$ with corresponding eigenvalue $\frac{\lambda_k+\lambda_1-2\lambda}{\lambda_k+\lambda_1}$.

Hence, writing $u \in \Phi_k$ as $u = \sum_{i=1}^k d_i\varphi_i$, where $d_i = \langle u, \varphi_i\rangle$, we have

$$\|\bar{L}u\|_2^2 = \left\|\sum_{i=1}^k \frac{\lambda_k+\lambda_1-2\lambda_i}{\lambda_k+\lambda_1} d_i\varphi_i\right\|_2^2 \leq \max_{i\in k}\left(\frac{\lambda_k+\lambda_1-2\lambda_i}{\lambda_k+\lambda_1}\right)^2 \left\|\sum_{i=1}^k d_i\varphi_i\right\|_2^2 \tag{2.5}$$

By the orthogonality of $\{\varphi_i\}_{i=1}^k$, we have

$$\left\|\sum_{i=1}^k d_i\varphi_i\right\|_2^2 = \sum_{i=1}^k d_i^2 = \|u\|_2^2$$

Since $\lambda_1 \leq \lambda_2 \cdots \leq \lambda_k$, we have $\lambda_k + \lambda_1 - 2\lambda_i \geq \lambda_1 - \lambda_k$ and $\lambda_k + \lambda_1 - 2\lambda_i \leq \lambda_k - \lambda_1$, so $|\lambda_k + \lambda_1 - 2\lambda_i| \leq \lambda_k - \lambda_1$. This implies $\max_{i\in k}\left(\frac{\lambda_k+\lambda_1-2\lambda_i}{\lambda_k+\lambda_1}\right)^2 \leq \left(\frac{\lambda_1-\lambda_k}{\lambda_1+\lambda_k}\right)^2$. Plugging this back in Equation 2.5, we get the claim we wanted. □

In fact, we will use a slight variant of the updates and instead set $u_{t+1} \leftarrow u_t - \eta(\tilde{L}u - \tilde{f}_{\text{span}})$ as the iterates of the convergent sequence, where $\tilde{f}_{\text{span}}$ is the projections of f onto $\tilde{\Phi}_K$. This sequence satisfies two important properties: (1) The convergence point of the sequence and $u^\star$, the solution to the original PDE, are not too far from each other; (2) The sequence of functions converges exponentially fast. In Section 2.6.2, we will see that updates defined thusly will be more convenient to simulate via a neural network.

The first property is formalized as follows:

Lemma 3. *Assume that $\tilde{u}^\star_{\text{span}}$ is the solution to the PDE $\tilde{L}u = \tilde{f}_{\text{span}}$, where $\tilde{f}_{\text{span}} : H_0^1(\Omega) \to \mathbb{R}$ is the projections of f onto $\tilde{\Phi}_K$. Given Assumptions (i)-(iii), we have $\|u^\star - \tilde{u}^\star_{\text{span}}\|_2 \leq \epsilon$, such that $\epsilon = \frac{\epsilon_{\text{span}}}{\lambda_1} + \frac{\delta}{\lambda_1}\frac{\|f\|_2}{\gamma-\delta} + \delta\|\tilde{u}^\star_{\text{span}}\|_2$, where $\gamma = \frac{1}{\lambda_k} - \frac{1}{\lambda_{k+1}}$ and $\delta = \max\left\{\frac{\epsilon_A}{m}, \frac{\epsilon_c}{\zeta}\right\}$.*

The proof for Lemma 3 is provided in the Appendix (Section 10.2.1). Each of the three terms in the final error captures different sources of perturbation: the first term comes from approximating f by f_{span}; the second term comes from applying Davis-Kahan [Davis and Kahan, 1970] to bound the "misalignment" between the eigenspaces $\text{span}\{\varphi_1, \cdots, \varphi_k\}$ and $\tilde{\Phi}_K$ (hence, the appearance of the eigengap between the k and $(k+1)$-st eigenvalue of L^{-1}); the third term is a type of "relative" error bounding the difference between the solutions to the PDEs $Lu = \tilde{f}_{\text{span}}$ and $\tilde{L}u = \tilde{f}_{\text{span}}$.

The "misalignment" term can be characterized through the following lemma:

Lemma 4 (Bounding distance between f_{span} and $\tilde{f}_{\text{span}}$)**.** *Given Assumptions (i)-(iii)and denoting the projection of f onto $\tilde{\Phi}_K$ by $\tilde{f}_{\text{span}}$ we have:*

$$\|f_{\text{span}} - \tilde{f}_{\text{span}}\|_2 \leq \frac{\|f\|_2 \delta}{\gamma - \delta} \tag{2.6}$$

where $\delta = \max\left\{\frac{\epsilon_A}{m}, \frac{\epsilon_c}{\zeta}\right\}$.

Proof. Let us write $f_{\text{span}} = \sum_{i=1}^k f_i \varphi_i$ where $f_i = \langle f, \varphi_i \rangle_2$. Further, we can define a function $\tilde{f}_{\text{span}} \in \tilde{\Phi}_K$ such that $\tilde{f}_{\text{span}} = \sum_{i=1}^k \tilde{f}_i \tilde{\varphi}_i$ such that $\tilde{f}_i = \langle f, \tilde{\varphi}_i \rangle_2$.

If $P_k g := \sum_{i=1}^k \langle g, \varphi_i \rangle_2 \varphi_i$ and $\tilde{P}_k g := \sum_{i=1}^k \langle g, \tilde{\varphi}_i \rangle_2 \tilde{\varphi}_i$ denote the projection of a function g onto $\text{span}\{\varphi_1, \cdots, \varphi_k\}$ and $\tilde{\Phi}_K$, from Lemma 28, we have:

$$\begin{aligned}
\|f_{\text{span}} - \tilde{f}_{\text{span}}\|_2 &= \left\| \sum_{i=1}^k \langle f, \varphi_i \rangle_2 \varphi_i - \langle f, \tilde{\varphi}_i \rangle_2 \tilde{\varphi}_i \right\|_2 \\
&= \left\| P_k f - \tilde{P}_k f \right\|_2 \\
&\leq \|P_k - \tilde{P}_k\| \|f\|_2 \\
&\leq \frac{\delta}{\gamma - \delta} \|f\|_2
\end{aligned}$$

where $\gamma = \frac{1}{\lambda_k} - \frac{1}{\lambda_{k+1}}$, and $\delta = \max\left\{\frac{\epsilon_A}{m}, \frac{\epsilon_c}{\zeta}\right\}$. □

The main technical tool for bounding the difference between the operators L and $\tilde{L}$ can be formalized through the lemma below. Note, the "relative" nature of the perturbation is because L and $\tilde{L}$ are not bounded operators.

Lemma 5 (Relative operator perturbation bound)**.** *Consider the operator $\tilde{L}$ defined in Equation 2.4, then for all $u \in H_0^1(\Omega)$ we have the following:*

1. $\langle (\tilde{L} - L)u, u \rangle \leq \delta \langle Lu, u \rangle$
2. $\langle (L^{-1}\tilde{L} - I)u, u \rangle_2 \leq \delta \|u\|_2^2$

where $\delta = \max\left\{\frac{\epsilon_A}{m}, \frac{\epsilon_c}{\zeta}\right\}$.

Proof.

$$\begin{aligned}\langle(\tilde{L}-L)u,u\rangle &= \int_\Omega \Big((\tilde{A}-A)\nabla u\cdot\nabla u + (\tilde{c}-c)u^2\Big)dx \\ &\leq \left(\max_{ij}\|\tilde{A}_{ij}-A_{ij}\|_{L^\infty(\Omega)}\right)\|\nabla u\|_2^2 + \|\tilde{c}-c\|_{L^\infty(\Omega)}\|u\|_2^2 \\ &\leq \epsilon_A\|\nabla u\|_2^2 + \epsilon_c\|u\|_2^2 \end{aligned} \tag{2.7}$$

Further, note that

$$\begin{aligned}\langle Lu,u\rangle &= \int_\Omega A\nabla u\cdot\nabla u + cu^2 dx \\ &\geq m\|\nabla u\|_2^2 + \zeta\|u\|_2^2 \end{aligned} \tag{2.8}$$

Using the inequality $\frac{a+b}{c+d} \geq \min\{\frac{a}{c}, \frac{b}{d}\}$ from Equation 2.7 and Equation 2.8, we have

$$\frac{m\|\nabla u\|_2^2 + \zeta\|u\|_2^2}{\epsilon_A\|\nabla u\|_2^2 + \epsilon_c\|u\|_2^2} \geq \min\left\{\frac{m}{\epsilon_A}, \frac{\zeta}{\epsilon_c}\right\} \tag{2.9}$$

Hence this implies that

$$\langle(\tilde{L}-L)u,u\rangle \leq \delta\langle Lu,u\rangle$$

where $\delta = \max\left\{\frac{\epsilon_A}{m}, \frac{\epsilon_c}{\zeta}\right\}$ proving part (1.).

Further, for part (2.) we have for all $u \in H_0^1(\Omega)$,

$$\begin{aligned} &\langle(\tilde{L}-L)u,u\rangle_2 \leq \delta\langle Lu,u\rangle_2 \\ \implies &\langle(\tilde{L}L^{-1}-I)Lu,u\rangle_2 \leq \delta\langle Lu,u\rangle_2 \\ \implies &\langle(\tilde{L}L^{-1}-I)v,u\rangle_2 \leq \delta\langle v,u\rangle_2 \\ \implies &\langle(\tilde{L}L^{-1})v,u\rangle_2 \leq (1+\delta)\langle v,u\rangle_2 \end{aligned} \tag{2.10}$$

where $v = Lu$. Therefore using Equation 2.10 the following holds for all $u \in H_0^1(\Omega)$,

$$\begin{aligned} &\langle(\tilde{L}L^{-1})u,u\rangle_2 \leq (1+\delta)\|u\|_2^2 \\ \overset{(1)}{\implies} &\langle u,(L^{-1}\tilde{L})u\rangle_2 \leq (1+\delta)\|u\|_2^2 \\ \overset{(2)}{\implies} &\langle(L^{-1}\tilde{L}-I)u,u\rangle_2 \leq \delta\|u\|_2^2 \end{aligned} \tag{2.11}$$

where we use the fact that the operators $\tilde{L}$ and L^{-1} are self-adjoint to get (1) and then bring the appropriate terms to the LHS in (2). □

The second property of the sequence of functions is that they converge exponentially fast. Namely, we show:

Lemma 6 (Convergence of gradient descent in L^2)**.** *Let $\tilde{u}^\star_{\text{span}}$ denote the unique solution to the PDE $\tilde{L}u = \tilde{f}_{\text{span}}$, where $\tilde{f}_{\text{span}} \in \tilde{\Phi}_K$, and the operator $\tilde{L}$ satisfies the conditions in Lemma 2. For any $u_0 \in H^1_0(\Omega)$ such that $u_0 \in \tilde{\Phi}_K$, we define the sequence*

$$u_{t+1} \leftarrow u_t - \frac{2}{\tilde{\lambda}_1 + \tilde{\lambda}_k}(\tilde{L}u_t - \tilde{f}_{\text{span}}) \quad (t \in \mathbb{N}) \tag{2.12}$$

where for all $t \in \mathbb{N}$, $u_t \in H^1_0(\Omega)$. Then for any $t \in \mathbb{N}$, we have

$$\|u_t - \tilde{u}^\star_{\text{span}}\|_2 \leq \left(\frac{\tilde{\lambda}_k - \tilde{\lambda}_1}{\tilde{\lambda}_k + \tilde{\lambda}_1}\right)^{t-1} \|u_0 - \tilde{u}^\star_{\text{span}}\|_2$$

The proof is essentially the same as the the analysis of the convergence time of gradient descent for strongly convex losses. Namely, we have:

Proof. Given that $u_0 \in H^1_0(\Omega)$ and $u_0 \in \tilde{\Phi}_K$ the function $\tilde{L}u_0 \in H^1_0(\Omega)$ and $\tilde{L}u_0 \in \tilde{\Phi}_K$ as well (from Lemma 2).

As $\tilde{f}_{\text{span}} \in \tilde{\Phi}_K$, all the iterates in the sequence will also belong to $H^1_0(\Omega)$ and will lie in the $\tilde{\Phi}_K$.

Now at a step t the iteration looks like,

$$\begin{aligned} u_{t+1} &= u_n - \frac{2}{\tilde{\lambda}_k + \tilde{\lambda}_1}\left(\tilde{L}u_t - \tilde{f}_{\text{span}}\right) \\ u_{t+1} - \tilde{u}^\star_{\text{span}} &= \left(I - \frac{2}{\tilde{\lambda}_k + \tilde{\lambda}_1}\tilde{L}\right)(u_t - \tilde{u}^\star_{\text{span}}) \end{aligned}$$

Using the result from Lemma 2, part 3. we have,

$$\begin{aligned} \|u_{t+1} - \tilde{u}^\star_{\text{span}}\|_2 &\leq \left(\frac{\tilde{\lambda}_k - \tilde{\lambda}_1}{\tilde{\lambda}_k + \tilde{\lambda}_1}\right)\|u_t - \tilde{u}^\star_{\text{span}}\|_2 \\ \implies \|u_{t+1} - \tilde{u}^\star_{\text{span}}\|_2 &\leq \left(\frac{\tilde{\lambda}_k - \tilde{\lambda}_1}{\tilde{\lambda}_k + \tilde{\lambda}_1}\right)^t \|u_0 - \tilde{u}^\star_{\text{span}}\|_2 \end{aligned}$$

This finishes the proof. □

Combining the results from Lemma 3 and Lemma 6 via triangle inequality, we have:

$$\|u^\star - u_T\|_2 \leq \|u^\star - \tilde{u}^\star_{\text{span}}\|_2 + \|\tilde{u}^\star_{\text{span}} - u_T\|_2$$

and the first term on the RHS subsumes the first three summands of $\tilde{\epsilon}$ defined in Theorem 1.

2.6.2 Approximating iterates by neural networks

In Lemma 6, we show that there exists a sequence of functions (2.12) which converge fast to a function close to $u^\star$. The next step in the proof is to approximate the iterates by neural networks.

The main idea is as follows. Suppose first the iterates $u_{t+1} = u_t - \eta(\tilde{L}u_t - \tilde{f}_{\text{span}})$ are such that $\tilde{f}_{\text{span}}$ is exactly representable as a neural network. Then, the iterate u_{t+1} can be written in terms of three operations performed on u_t, a and f: taking derivatives, multiplication and addition. Moreover, if g is representable as a neural network with N parameters, the coordinates of the vector ∇g can be represented by a neural network with $O(N)$ parameters. This is a classic result (Lemma 8), essentially following from the backpropagation algorithm. Finally, addition or multiplication of two functions representable as neural networks with sizes N_1, N_2 can be represented as neural networks with size $O(N_1 + N_2)$ (see Lemma 9).

Using these facts, we can write down a recurrence upper bounding the size of neural network approximation u_{t+1}, denoted by $\hat{u}_{t+1}$, in terms of the number of parameters in $\hat{u}_t$ (which is the neural network approximation to u_t). Formally, we have:

Lemma 7 (Recursion Lemma)**.** *Given the Assumptions (i)-(iii), consider the update equation*

$$\hat{u}_{t+1} \leftarrow \hat{u}_t - \frac{2}{\tilde{\lambda}_1 + \tilde{\lambda}_k}\left(\tilde{L}\hat{u}_t - f_{\text{nn}}\right) \tag{2.13}$$

If at step t, $\hat{u}_t : \mathbb{R}^d \to \mathbb{R}$ is a neural network with N_t parameters, then the function $\hat{u}_{t+1}$ is a neural network with $O(d^2(N_A + N_t) + N_t + N_{\tilde{f}} + N_c)$ parameters.

Proof. Expand the update $\hat{u}_{t+1} \leftarrow \hat{u}_t - \eta\left(\tilde{L}\hat{u}_t - f_{\text{nn}}\right)$ as follows:

$$\hat{u}_{t+1} \leftarrow \hat{u}_t - \eta\left(\sum_{i,j=1}^{d} \tilde{a}_{ij}\partial_{ij}\hat{u}_t + \sum_{j=1}^{d}\left(\sum_{i=1}^{d}\partial_i\tilde{a}_{ij}\right)\partial_j\hat{u}_t + \tilde{c}\hat{u}_t - f_{\text{nn}}\right).$$

Using Lemma 8, $\partial_{ij}\hat{u}_t$, $\partial_j\hat{u}_t$ and $\partial_i\tilde{a}_{ij}$ can be represented by a neural network with $O(N_t)$, $O(N_t)$ and $O(N_A)$ parameters, respectively. Further, $\partial_i\tilde{a}_{ij}\partial_j u$ and $\tilde{a}_{ij}\partial_{ij}\hat{u}$ can be represented by a neural network with $O(N_A + N_t)$ parameters, and $\tilde{c}\hat{u}_t$ can be represented by a network with $O(N_t + N_c)$ parameters, from Lemma 9. Hence $\hat{u}_{t+1}$ can be represented in $O(d^2(N_A + N_t) + N_f + N_c + N_t)$ parameters. Note that, throughout the entire proofs O hides independent constants. □

Combining the results of Lemma 6 and Lemma 7, we can get a recurrence for the number of parameters required to represent the neural network $\hat{u}_t$:

$$N_{t+1} \leq d^2 N_t + d^2 N_A + N_t + N_{\tilde{f}} + N_c$$

Unfolding this recurrence, we get $N_T \leq d^{2T} N_0 + \frac{d^2(d^T-1)}{d^2-1} N_A + T(N_f) + N_c)$.

The formal lemmas for the different operations on neural networks we can simulate using a new neural network are as follows:

Lemma 8 (Backpropagation, Rumelhart et al. [1986]). *Consider neural network $g : \mathbb{R}^m \to \mathbb{R}$ with depth l, N parameters and differentiable activation functions in the set $\{\sigma_i\}_{i=1}^A$. There exists a neural network of size $O(l+N)$ and activation functions in the set $\{\sigma_i, \sigma_i'\}_{i=1}^A$ that calculates the gradient $\frac{dg}{di}$ for all $i \in [m]$.*

Lemma 9 (Addition and Multiplication). *Given neural networks $g : \Omega \to \mathbb{R}$, $h : \Omega \to \mathbb{R}$, with N_g and N_h parameters respectively, the operations $g(x) + h(x)$ and $g(x) \cdot h(x)$ can be represented by neural networks of size $O(N_g + N_h)$, and square activation functions.*

Proof. For *Addition*, there exists a network h containing both networks f and g as subnetworks and an extra layer to compute the addition between their outputs. Hence, the total number of parameters in such a network will be $O(N_f + N_g)$.

For *Multiplication*, consider the operation $f(x) \cdot g(x) = \frac{1}{2}\big((f(x) + g(x))^2 - f(x)^2 - g(x)^2\big)$. Then following the same argument as for addition of two networks, we can construct a network h containing both networks and square activation function. □

While the representation result in Lemma 9 is shown using square activation, we refer to Yarotsky [2017] for approximation results with ReLU activation. The scaling with respect to the number of parameters in the network remains the same.

Finally, we have to deal with the fact that $\tilde{f}_{\text{span}}$ is not exactly a neural network, but only approximately so. The error due to this discrepancy can be characterized through the following lemma:

Lemma 10 (Error using f_{nn}). *Consider the update equation in Equation 2.13, where f_{nn} is a neural network with N_f. Then the neural network $\hat{u}_t$ approximates the function u_t such that $\|u_t - \hat{u}_t\|_2 \leq \epsilon_{\text{nn}}^{(t)}$ where $\epsilon_{\text{nn}}^{(t)}$ is*

$$O\left((\max\{1, t^2\eta e C\})^t \left(\epsilon_{\text{span}} + \epsilon_{\text{nn}} + 4\left(1 + \frac{\delta}{\gamma - \delta}\right)\lambda_k^t \|f\|_2\right)\right)$$

where $\delta = \max\left\{\frac{\epsilon_A}{m}, \frac{\epsilon_c}{\zeta}\right\}$, $\gamma = \frac{1}{\lambda_k} - \frac{1}{\lambda_{k+1}}$, and α is a multi-index.

The proof for the lemma is deferred to Section 10.2.2 of the Appendix. The main strategy to prove this lemma involves tracking the "residual" non-neural-network part of the iterates. Precisely, for every $t \in \mathbb{N}$, we will write $u_t = \hat{u}_t + r_t$, s.t. $\hat{u}_t$ is a neural network and bound $\|r_t\|_2$. $\{\hat{u}_t\}_{t=0}^{\infty}$ is defined such that

$$\begin{cases} \hat{u}_0 = u_0, \\ \hat{u}_{t+1} = \hat{u}_t - \eta\Big(\tilde{L}\hat{u}_t - f_{\text{nn}}\Big) \end{cases}$$

Correspondingly, as $r_t = u_t - \hat{u}_t$, we have:

$$\begin{cases} r_0 = 0, \\ r_{t+1} = (I - \eta\tilde{L})r_t - r \end{cases}$$

Unfolding the recurrence, we have $r_t = \sum_{i=0}^{t-1}(I - \eta\tilde{L})^{(i)}r$, which reduces the proof to bounding $\|(I - \eta\tilde{L})^{(i)}\|_2$. [3]

2.7 Applications to Learning Operators

A number of recent works attempt to simultaneously approximate the solutions for an entire family of PDEs by learning a *parametric map* that takes as inputs (some representation of) the coefficients of a PDE and returns its solution [Bhattacharya et al., 2020, Li et al., 2020b,a]. For example, given a set of observations that $\{a_j, u_j\}_{j=1}^N$, where each a_j denotes a coefficient of a PDE with corresponding solution u_j, they learn a neural network G such that for all j, $u_j = G(a_j)$. Our parametric results provide useful insights for why simultaneously solving an entire family of PDEs with a *single neural network* G is possible in the case of linear elliptic PDEs.

Consider the case where the coefficients a_j in the family of PDEs are given by neural networks with a fixed architecture, but where each instance of a PDE is characterized by a different setting of the weights in the models representing the coefficients. Lemma 7 shows that each iteration of our sequence (2.12) constructs a new network containing both the current solution and the coefficient networks as subnetworks. We can view our approximation as not merely approximating the solution to a single PDE but to every PDE in the family, by treating the coefficient networks as placeholder architectures whose weights are provided as inputs. Thus, our construction provides a parametric map between the coefficients of an elliptic PDE in this family and its solution.

2.8 Conclusion and Future Work

We derive parametric complexity bounds for neural network approximations for solving linear elliptic PDEs with Dirichlet boundary conditions, whenever the coefficients can be approximated by are neural networks with finite parameter counts. By simulating gradient descent in function spaces using neural networks, we construct a neural network that approximates the solution of a PDE. We show that the number of parameters in the neural network depends on the parameters required to represent the coeffcients and has a poly(d) dependence on the dimension of the input space, therefore avoiding the curse of dimensionality.

An immediate open question is related to the tightening our results: our current error bound is sensitive to the neural network approximation lying close to $\text{span}\{\varphi_1, \cdots, \varphi_k\}$ which could be alleviated by re-

[3] The reason we require that f_{nn} is close to f not only in the L_2 sense but also in terms of their higher order derivatives is since $\tilde{L}^{(t)}r$ involves $2t$-order derivatives of r to be bounded at each step.

laxing (by adding some kind of "regularity" assumptions) the dependence of our analysis on the first k eigenfunctions. Further, the dependencies in the exponent of d on R and κ in parametric bound may also be improvable. Finally, the idea of simulating an iterative algorithm by a neural network to derive a representation-theoretic result is broadly applicable, and may be a fertile ground for further work, both theoretically and empirically, as it suggest a particular kind of weight tying.

3 Benefits of Depth in Neural Approximations of PDEs

Abstract *A wave of experimental papers have recently demonstrated the promise of deep neural networks for approximating the solutions to high-dimensional Partial Differential Equations (PDEs). Recently, several theoretical works have addressed some attendant expressivity problems (e.g., conditions under which such solutions might escape the curse of dimensionality). However, a number of questions remain open, including whether approximating the solution to classical PDEs really requires that networks be* deep. *In this paper, we focus on benefits of depth, showing that even for approximating simple linear PDEs with constant boundary conditions, depth can provide significant advantages. Our analysis addresses the Helmholtz family* $\Delta u + k^2\pi^2 u = 0, k \in \mathbb{R}$ *with zero boundary condition, proving that approximating the solution with networks with less than* $\sqrt{\log k}$ *layers requires* $O(2^{\sqrt{\log(k)}})$*-sized networks. Our paper sets work on the theory of deep learning in dialogue with the nascent neural PDE literature, Our paper shows utilizes results from deep learning theory to show that neural network solutions to even simple PDEs may often lie amont the set of functions characterized by by depth separation.*

3.1 Introduction

By now, the broad success of neural network methods for producing approximate solutions to high-dimensional learning problems, e.g., those arising in in computer vision [Krizhevsky et al., 2012, Simonyan and Zisserman, 2014, He et al., 2016], and natural language processing [Bahdanau et al., 2014, Devlin et al., 2018]. More recently, a burgeoning line of research has successfully applied neural networks to approximate solutions to partial differential equations (PDEs) [Yu et al., 2017, Raissi et al., 2017]. PDEs are multivariate equations that define the relation of a function to its partial and are often used to model important scientific and social phenomena such as wave scattering [Bendali and Lemrabet, 1996], fluid dynamics [Anderson and Wendt, 1995] and the financial markets [Black and Scholes, 1973, Ehrhardt and Mickens, 2008]. Recent empirical work, including Yu et al. [2017], Raissi et al. [2017], Han et al. [2018], Grohs and Herrmann [2020], Moseley et al. [2020] have introduced scores of different architectures to approximate solutions of various PDEs. These papers demonstrate that for many families of PDEs such as the Hamilton-Jacobi-Bellman (HJB) and Black-Scholes equations (for example, Han et al. [2018] solve a 1000 dimensional HJB equation) neural networks are superior to classical grid-based methods.

However, unlike classical domains like images and text, where the architecture design is guided by strong intuitions about structural properties of the domain data, it is still unclear how neural network architec-

tures should be chosen when applied to PDEs—though some recent works Li et al. [2020a], Marwah et al. [2021] suggest that a deep architecture with blocks that have some form of weight-tying may be appropriate.

On the theoretical front, our understanding of the representational capabilities of neural networks in the context of PDE solvers is still nascent. Earlier works such as Sirignano and Spiliopoulos [2018] use universal approximation to show the existence of neural networks that approximate the solutions to parabolic PDEs. Works like Grohs and Herrmann [2020], Marwah et al. [2021] focused on exhibiting examples of PDE families for which neural networks can avoid the curse of dimensionality. Finally, Chen et al. [2021] characterized the Barron norms of solutions to PDEs, which are intimately tied to the size of shallow networks required to approximate them.

On the other hand, the limitations of neural network-based approaches to PDEs are not well understood. More broadly, concerning the representational power of neural networks, seminal works by Telgarsky [2016] and Eldan and Shamir [2016] proved that there exist functions that can be approximated by small deep networks, but not small shallow ones.

In this paper, we observe that these tools for understanding the benefit of depth in neural networks can be naturally applied to understand the need for deep architectures for PDEs—even for exceedingly simple types of PDEs and boundary conditions.

Precisely, we focus on the Helmholtz family of PDEs $\Delta u + k^2\pi^2 u = 0, k \in \mathbb{R}$—an exceedingly simple one-parameter family of *linear* PDEs, with zero Dirichlet boundary conditions. We show that for approximating the solutions to such PDEs with a shallow network requires that the size of each layer is large. Precisely, we show a network with less than $\sqrt{\log(k)}$ layers necessitates that the network is of size $O(2^{\sqrt{\log(k)}})$. By standard results on approximating trigonometric functions [Perekrestenko et al., 2018], this dependence on k is nearly tight—in the one-dimensional setting, a network with $O(\log(k))$ layers of size $O(1)$ can approximate the solution to the Helmholtz PDE.

We note that Helmholtz equations are independently of interest in many domains: they arise as a time-independent form of the wave equation in problems associated with electromagnetic wave scattering, seismology, and acoustics. Developing efficient numerical algorithms for Helmholtz equations has been an active area of research, including those in the high frequency regime ($k \gg 1$), such as the celebrated fast multipole method [Cheng et al., 2006] and fast directional multilevel method [Engquist and Ying, 2007]. As a result, there is also considerable interest in developing neural network based solution methods [Khoo and Ying, 2019, Wang et al., 2020].

3.2 Overview of Results

We will focus on the following one-parameter family of *linear* PDEs:

Definition 9 (Helmholtz PDE). *A Helmholtz PDE with Dirichlet boundary condition takes the following form:* [1]

$$\begin{cases} \Delta u + k^2\pi^2 u = 0, & \forall x \in D \\ u(x) = 0 & \forall x \in \partial D \end{cases} \tag{3.1}$$

where $u : \mathbb{R}^d \to \mathbb{R}$. Here, D is a bounded open set with boundary ∂D.

In this work, we focus on the case where the domain D is a d-dimensional hypercube, that is $\bar{D} = [0,1]^d$ ($\bar{D} := D \cup \partial D$). In this case a solution to the PDE in Definition 9 takes the following form:

$$\begin{aligned} u^\star(x) &= \prod_{i=1}^{d} \sin(k_i \pi x_i) \\ \text{subject to} \quad & \sum_{i}^{d} k_i^2 = k^2 \end{aligned} \tag{3.2}$$

Specifically, when $d = 1$, we have:

$$u_1^\star(x) = \sin(k\pi x) \tag{3.3}$$

We will show that:

Theorem (Informal). *A ReLU neural network with depth $O\left(\sqrt{\log(k)}\right)$ that approximates a solution to the Helmholtz equation in Equation 3.2 has size $\Omega(2^{\sqrt{\log(k)}})$. .*

Our proof proceeds by first establishing the lower bound for the one-dimensional case, and then generalizing it to the multi-dimensional case. In the one-dimensional case, we leverage the fact that the solution is highly oscillatory—namely, changes sign frequently. This allows us to estimate the error arising from regions where the approximating neural network f and $u_1^\star$ have opposite sign. In the case $d > 1$, we reduce the problem to a one-dimensional one by only tracking the error in one of the dimensions—coupled with the fact that the $u^\star$ factorizes over the individual dimensions.

We note that the dependence on k in the result is almost tight: by results from Perekrestenko et al. [2018], in the case of $d = 1$, a deep network with depth and size $O(\log(k))$ can approximate the solution of Equation 3.3.

3.3 Related Work

Over the past few years there has been a growing line of work that utilizes neural networks to parameterize the solution to a PDE. Works such as Weinan et al. [2017], Yu et al. [2017], Sirignano and Spiliopoulos [2018], Raissi et al. [2017] achieved impressive results on a variety of different applications and have

[1] Here, Δ is the Laplacian operator. For a twice differentiable function $f : \mathbb{R}^d \to \mathbb{R}$, $\Delta f = \sum_{i=1}^{d} \frac{\partial^2 f}{\partial x_i^2}$.

demonstrated the empirical efficacy of neural networks in solving high dimensional PDEs. This is a great and promising direction for solving PDEs since erstwhile dominant numerical approaches like the finite differences and finite element methods [LeVeque, 2007] depend primarily upon discretizing the input space, hence limiting their use for problems on low dimensional input space.

Several recent work look to theoretically analyse these neural network based approaches for solving PDEs. Mishra and Molinaro [2020] look at the generalization properties of physics informed neural networks. In Lu et al. [2021] show the generalization analysis for the Deep Ritz method for elliptic equations like the Poissoin equation and Lu and Lu [2021] extends their analysis to the Schrodinger eigenvalue problem.

In addition to analyzing the generalization capabilities of the neural networks, theoretical analysis into their representational capabilities has also gained a lot of attention. Khoo et al. [2021] show the existence of a network by discretizing the input space into a mesh and then using convolutional NNs, where the size of the layers is exponential in the input dimension. Sirignano and Spiliopoulos [2018] provide a universal approximation result, showing that for sufficiently regularized PDEs, there exists a multilayer network that approximates its solution. However, they do not comment on the complexity of the neural network, how it scales with the input dimension and also how to design such a network. In Jentzen et al. [2018], Grohs and Herrmann [2020], Hutzenthaler et al. [2020] show that provided a better-than-exponential dependence on the input dimension for some special parabolic PDEs, based on a stochastic representation using the Feynman-Kac Lemma, thus limiting the applicability of their approach to PDEs that have such a probabilistic interpretation. Moreover, their results avoid the curse of dimensionality only over domains that with unit volume (for example a hypercube with side length one). Furthermore, their construction does not provide any insight into architectural choices such as the depth of the neural network.

In Marwah et al. [2021] the authors show that for an elliptic PDE's coefficients are approximable by small neural networks with at most N, then for a sufficiently smooth solution, the neural network that approximates the solution avoids the curse of dimensionality and has a $O(poly(dN))$. Their analysis suggests that a certain weight-tied network architecture can be useful for approximating solutions to PDEs. In Chen et al. [2021] extends this analysis to the Barron space and shows that if the coefficients are in Barron spaces [Barron, 1993]. While these works show important representation theoretic results, they do not discuss the types of architectures that would enable the approximation of the solutions.

In Li et al. [2020a,b] the authors show (experimentally) that an entire family of different types of PDEs, such as elliptic, parabolic PDES, can be approximated by a single neural network. They approximate a map from a PDE's parameters to its solution and avoid the need for retraining for each set of coefficient. Furthermore, they show that construction of such a network follows an iterative architecture whereby each layer of the network is a function of the previous layer and a kernel operator. This paper does provide some key insights into architectures that could be useful for designing neural networks that approximate the solution/family of PDEs. However, they do not mention how many layers these networks would need or what the size of each layer should be.

Our work proves that for neural networks approximating solutions to a Helmholtz PDE (Definition 9) depth is key, and show that deep neural networks with $O(\log(k))$ depth and size $O(1)$. Furthermore, we show that shallow neural networks with less than $\sqrt{\log(k)}$ would need $\Omega(2^{\sqrt{\log(k)}})$ sized networks.

3.4 Main Result

Definition 10 (ReLU Network)**.** *An L-layer ReLU network is a function*

$$x \mapsto w_L \sigma(\dots \sigma(w_1^T x + b_1) \dots) + b_L.$$

where $\sigma(x) = \max\{0, x\}$*. The* size *of a neural network is the total number of nodes in the network.*

We now state our main result:

Theorem 2 (Depth Lower Bound)**.** *Consider the PDE in Definition 9 with solution* $u^\star$ *defined by Equation 3.2. Furthermore, let* $k_{\max} := \max\{k_1, k_2, \cdots, k_d\}$*. Then, for any ReLU network* $f : \mathbb{R}^d \to \mathbb{R}$ *with depth less than* $\sqrt{\log(\lfloor k_{\max} \rfloor)}$ *and size less than* $2^{\sqrt{\log(\lfloor k_{\max} \rfloor)}}$*, we have,*

$$\int_{[0,1]^d} |u^\star(x) - f(x)| dx \geq \frac{1}{2} \|u^*\|_{L_1}.$$

We remark on several aspects of the above statement.

Remark 1: The error on the left-hand-size is the l_1 error in approximating u^* by f. The right hand side has a natural scaling by the l_1 norm of the function u^* we are approximating.

Remark 2: While the proof of the above theorem relies on the highly oscillatory nature of the solution of the PDE, the coefficients of the PDE itself are exceedingly simple: in fact they are constant—as are the boundary conditions. This is particularly relevant in light of recent theoretical work providing upper bounds on the necessary size to approximate the solution of a PDE as a function of the size of the networks required to approximate the coefficients of the PDE [Marwah et al., 2021, Chen et al., 2021]. These result suggest that from the point of view of neural network approximation, PDEs whose coefficients are approximable by small networks are easier to represent by neural networks. Our lower bound show that even for PDEs that are easy in this sense, a very deep network is needed.

Remark 3: In this paper we consider a fully connected network with ReLU activation function. However, as mentioned in Hornik et al. [1990] and Telgarsky [2017] one can approximate a neural network with one choice of nonlinearity via a (comparably sized) neural network with another choice of nonlinearity, under very mild conditions on the nonlinearities. Crucially, this simulation only increases the size by a dimension-independent factor, and keeping the depth separation same. However, we note that the effect of using other architectures like convolutional neural networks are outside the scope of current work.

The proof for the above theorem is provided in Section 3.5. Moreover, the dependence on k is nearly optimal. Namely, in the case $d = 1$, a result by Perekrestenko et al. [2018] for approximating trigono-

metric functions can be directly applied to conclude that a network of depth and size $O(\log k)$ suffices to approximate u^*. Precisely:

Theorem 3 (Depth Upper Bound, Theorem 4.1 in Perekrestenko et al. [2018]). *Consider the function $u_1^\star$, which is the solution to the Helmholtz PDE in Definition 9 for $d = 1$. There exists a neural network $f : \mathbb{R} \to \mathbb{R}$ with ReLU activation function with depth $O((\log(1/\epsilon)^2 + \log(k))$ and width $O(1)$ such that*

$$\sup_{x\in[0,1]} |u^\star(x) - f(x)| \leq \epsilon$$

We note that the gap between the lower and upper bound ($O(\sqrt{\log k})$ vs $O(\log k)$) is expected, as such gaps in our understanding of depth separation for neural networks also exist outside of the context of PDEs [Telgarsky, 2016].

3.5 Proof of Theorem 2

The main intuition for the proof follows that from Telgarsky [2016]. A function with many oscillations will be poorly approximated by a network with fewer oscillations. We further use the fact that a deep neural network can represent a function with more oscillations (relative to the number of parameters) than its shallow counterpart, which will have fewer oscillations.

We begin the proof by stating two ingredients from Telgarsky [2016].

First, we lower bound the distance between two functions roughly by counting the number of times they change sign in one dimension. Precisely, consider that the function $u_1^\star : [0, 1] \to \mathbb{R}$ partitions the set D into intervals $\mathcal{I}_{u^\star}$ such that for each interval $\tilde{f}(x) = \mathbf{1}[f(x) \geq 0]$ is constant. Therefore, the number of times the function f changes sign is given by $|\mathcal{I}_f|$. We show that if a function f changes sign a much fewer number of times than $u_1^\star$, i.e., $|\mathcal{I}_f|L^2(\Omega)|\mathcal{I}_{u_1^\star}|$ then it will poorly approximate the function $u_1^\star$. Precisely:

Lemma 11 (Lemma 3.1 in Telgarsky [2016]). *Let $u_1^\star : \mathbb{R} \to \mathbb{R}$ and $f : \mathbb{R} \to \mathbb{R}$ be given functions, and denote*

$$\tilde{u}_1^\star := \mathbf{1}[u_1^\star \geq 0],$$
$$\tilde{f} := \mathbf{1}[f \geq 0].$$

Then we have,

$$\frac{1}{\lfloor k \rfloor + 1} \sum_{U\in\mathcal{I}_{u^\star}} \mathbf{1}\left[\forall x \in U; \tilde{u}_1^\star(x) \neq \tilde{f}(x)\right] \geq \frac{1}{2}\left(1 - 2\frac{|\mathcal{I}_f|}{\lfloor k \rfloor + 1}\right)$$

Next, we state a result that relates the number of layers and size of a neural network with the number of sign changes of the function.

Lemma 12 (Number of sign changes, Lemma 3.2 in Telgarsky [2016])**.** *Let g be a ReLU network with L layers with widths $(m_1, m_2, \ldots, m_L)$ that partitions the domain D into $\mathcal{I}_g$ intervals. If $m = \Pi_{i=1}^{L} m_i$, then*

$$|\mathcal{I}_g| \leq \left(\frac{2m}{L}\right)^L$$

.

Given Lemma 11 and Lemma 12 we now show the result for the lower bound on the error for the one-dimensional case.

Lemma 13 (Depth lower bound in one dimension)**.** *Consider the function $u_1^\star : \mathbb{R} \to \mathbb{R}$, the solution to the Helmholtz PDE (Definition 9) for $d = 1$. Let f be a ReLU network with L layers with $L \leq \sqrt{\log(k)}$ and size of each layer less than $2^{\sqrt{\log(k)}}$. Then we have:*

$$\int_{[0,1]} |u_1^\star(x) - f(x)| dx \geq \frac{1}{2\pi}.$$

Proof. Let $A_{u_1^\star}(U)$ denote the area under the curve defined by the function $u_1^\star$ calculated over the set U.

Note that for all $U \in \mathcal{I}_{u_1^\star}$, the area of the curve $u_1^\star$ ($A_{u_1^\star}(U)$) will be the same and can be calculated as the following,

$$\begin{aligned}
A_{u_1^\star}(U) &= \int_U \sin(k\pi x) dx \\
&= \int_{[0, \frac{1}{k}]} \sin(k\pi x) dx \\
&= \frac{1}{k\pi}\left(1 - \cos(k\pi \cdot \frac{1}{k})\right) \\
&= \frac{2}{k\pi}
\end{aligned}$$

Since f is a ReLU network with less than $\sqrt{\log(\lfloor k \rfloor) - 2}$ layers and less than $2^{\sqrt{\log(\lfloor k \rfloor - 2)}}$ nodes in each layer, using the result from Lemma 12 we have that the total number of sign changes of f is bounded as:

$$\begin{aligned}
|\mathcal{I}_f| &\leq \left(\frac{2 \cdot 2^{\sqrt{\log(\lfloor k \rfloor - 2)}}}{L}\right)^{\sqrt{\log(\lfloor k \rfloor - 2)}} \\
&\leq 2^{\log(\lfloor k \rfloor) - 2} = \frac{\lfloor k \rfloor}{4}
\end{aligned}$$

Here we have used the fact that $\frac{2}{\sqrt{\log(\lfloor k \rfloor) - 2}} \leq 1$.

Therefore, by Lemma 11 and Lemma 12 we can lower bound the error between the solution $u_1^\star$ and f as:

$$\begin{aligned}
&\int_\Omega |\sin(k\pi x) - f(x)|dx \\
&= \sum_{U\in\mathcal{I}_{u_1^\star}} \int_U |\sin(k\pi x) - f(x)|dx \\
&\geq \sum_{U\in\mathcal{I}_{u_1^\star}} \int_U |\sin(k\pi x)|\mathbf{1}[\forall x \in U : \tilde{u}_1^\star(x) \neq \tilde{f}(x)]dx \\
&\geq \sum_{U\in\mathcal{I}_{u_1^\star}} \int_U |\sin(k\pi x)|\mathbf{1}[\forall x \in U : \tilde{u}_1^\star(x) \neq \tilde{f}(x)]dx \\
&\geq \frac{2}{k\pi} \sum_{U\in\mathcal{I}_{u_1^\star}} \mathbf{1}[\forall x \in U : \tilde{u}_1^\star(x) \neq \tilde{f}(x)] \\
&\overset{(i)}{\geq} \frac{2}{k\pi}\frac{|\mathcal{I}_{u_1^\star}|}{2}\left(1 - \frac{2|\mathcal{I}_f|}{|\mathcal{I}_{u_1^\star}|}\right) \\
&\overset{(ii)}{\geq} \frac{\lfloor k\rfloor + 1}{k\pi}\left(1 - \frac{\lfloor k\rfloor}{2(\lfloor k\rfloor + 1)}\right) \\
&\geq \frac{\lfloor k\rfloor + 1}{2k\pi} \\
&\geq \frac{1}{2\pi}
\end{aligned}$$

Where we get (i) from Lemma 11 and we use the fact that $\frac{\lfloor k\rfloor}{\lfloor k\rfloor+1} \leq 1$ in (ii). □

We now extend the proof for Lemma 13 for the d dimensional case and complete the proof for Theorem 2.

Proof for Theorem 2. Let $\boldsymbol{x} := (x_1, x_2, \cdots, x_n)$ be a d-dimensional vector. Furthermore, will denote by $\boldsymbol{x}_{-i} := (x_1, \cdots, x_{i-1}, x_{i+1}, \cdots x_d)$.

We wish to lower bound the following error:

$$\begin{aligned}
&\int_{[0,1]^d} |u^\star(\boldsymbol{x}) - f(\boldsymbol{x})|d\boldsymbol{x} \\
&= \int_{[0,1]^d} |\prod_{j=1}^d \sin(k_j\pi x_j) - f(\boldsymbol{x})|d\boldsymbol{x} \\
&= \int_{[0,1]^{d-1}} \int_{x_i\in[0,1]} |\prod_{i=1}^d \sin(k_i\pi x_i) - f(\boldsymbol{x})|dx_i d\boldsymbol{x}_{-i} \qquad (3.4)
\end{aligned}$$

Consider a fixed value for the coordinates other than the i-th one, i.e. $\boldsymbol{x}_{-i} = (c_1, \cdots, c_{i-1}, c_{i+1}, \cdots, c_d)$ and define $\tilde{f}(x_i) := f(c; x_i)$ and $\prod_{\substack{j=1 \\ j\neq i}} \sin(k_j \pi c_j) =: C$. Hence, we wish to lower bound the following error,

$$\int_{x_i \in [0,1]} |C \sin(k_i \pi x_i) - \tilde{f}(x_i)| dx_i \tag{3.5}$$

From the proof of Lemma 13, we know that if the network $\tilde{f}$ has less than $\sqrt{\log(k_i) - 2}$ layers of size less than $2^{\sqrt{\log(k_i)-2}}$ then the error in Equation 3.5 can be lower bounded as the following:

$$\int_{x_i \in [0,1]} |C \sin(k_i \pi x_i) - \tilde{f}(x_i)| dx_i \geq \frac{|C|}{2\pi} \tag{3.6}$$

Substituting Equation 3.6 into Equation 3.4 we have,

$$\begin{aligned}
&\int_{[0,1]^{d-1}} \int_{x_i \in [0,1]} |\prod_{i=1}^{d} \sin(k_i \pi x_i) - f(\boldsymbol{x})| dx_i d\boldsymbol{x}_{-i} \\
&= \int_{[0,1]^{d-1}} \int_{x_i \in [0,1]} |C \sin(k_i \pi x_i) - \tilde{f}(x)| dx_i d\boldsymbol{x}_{-i} \\
&\geq \int_{[0,1]^{d-1}} \frac{|C|}{2\pi} d\boldsymbol{x}_{-i} \\
&= \frac{1}{2\pi} \int_{[0,1]^{d-1}} \left| \prod_{\substack{j=1 \\ j\neq i}}^{d} \sin(k_j \pi x_j) \right| d\boldsymbol{x}_{-i} \\
&\geq^{(1)} \frac{1}{2\pi} \int_{[0,1]^{d-1}} \left| \prod_{j=1}^{d} \sin(k_j \pi x_j) \right| d\boldsymbol{x}_{-i} \\
&= \frac{1}{2\pi} \|u\|_{L_1}
\end{aligned}$$

where (1) follows from $\sin(x) \leq 1$. □

3.6 Conclusion and Future Work

In this paper, we explore the role of depth in neural network architectures used to approximate PDEs. We show that in order to approximate the solution of linear Helmholtz PDE, a network with number of layers less than $O(\sqrt{\log(k)})$ would required layers of size $O(2^{\sqrt{\log(k)}})$—which is nearly tight in the one-dimensional case, as a network of depth and size $O(\log(k))$ suffices.

Exploring the limitations of neural networks when applied to PDE solvers is a wide open area — an immediate open question is to extend our lower bounds to more restricted classes of PDEs, e.g. elliptic

and parabolic PDEs. Studying more fine-grained architectural aspects (e.g. weight tieing, as suggested by results in Li et al. [2020a], Marwah et al. [2021]) seems also a promising direction for further work.

4 Neural Network Approximations of PDEs Beyond Linearity: A Representational Perspective

Abstract: *A burgeoning line of research leverages deep neural networks to approximate the solutions to high dimensional PDEs, opening lines of theoretical inquiry focused on explaining how it is that these models appear to evade the curse of dimensionality. However, most prior theoretical analyses have been limited to linear PDEs. In this work, we take a step towards studying the representational power of neural networks for approximating solutions to nonlinear PDEs. We focus on a class of PDEs known as* nonlinear elliptic variational PDEs*, whose solutions minimize an* Euler-Lagrange *energy functional* $\mathcal{E}(u) = \int_\Omega L(x, u(x), \nabla u(x)) - f(x)u(x)dx$*. We show that if composing a function with Barron norm* b *with partial derivatives of* L *produces a function of Barron norm at most* $B_L b^p$*, the solution to the PDE can be* ϵ*-approximated in the* L^2 *sense by a function with Barron norm* $O\left((dB_L)^{\max\{p\log(1/\epsilon), p^{\log(1/\epsilon)}\}}\right)$*. By a classical result due to Barron [1993], this correspondingly bounds the size of a 2-layer neural network needed to approximate the solution. Treating* p, ϵ, B_L *as constants, this quantity is polynomial in dimension, thus showing neural networks can evade the curse of dimensionality. Our proof technique involves neurally simulating (preconditioned) gradient in an appropriate Hilbert space, which converges exponentially fast to the solution of the PDE, and such that we can bound the increase of the Barron norm at each iterate. Our results subsume and substantially generalize analogous prior results for linear elliptic PDEs over a unit hypercube.*

4.1 Introduction

Scientific applications have become one of the new frontiers for the application of deep learning [Jumper et al., 2021, Tunyasuvunakool et al., 2021, Sønderby et al., 2020]. PDEs are a fundamental modeling techniques, and designing neural networks-aided solvers, particularly in high-dimensions, is of widespread usage in many scientific domains [Hsieh et al., 2019, Brandstetter et al., 2022]. One of the most common approaches for applying neural networks to solve PDEs is to parametrize the solution as a neural network and minimize a variational objective that represents the solution [Sirignano and Spiliopoulos, 2018, Yu et al., 2017]. The hope in doing so is to have a method which computationally avoids the "curse of dimensionality"—i.e., that scales less than exponentially with the ambient dimension.

To date, neither theoretical analysis nor empirical applications have yielded a precise characterization of the range of PDEs for which neural networks-aided methods outperform classical methods. Active research on the empirical side [Han et al., 2018, Weinan et al., 2017, Li et al., 2020a,b] has explored

several families of PDEs, e.g., Hamilton-Bellman-Jacobi and Black-Scholes, where neural networks have been demonstrated to outperform classical grid-based methods. On the theory side, a recent line of works [Marwah et al., 2021, Chen et al., 2021, 2022] has considered the following fundamental question:

For what families of PDEs, can the solution be represented by a small neural network?

The motivation for this question is computational: fitting the neural network (by minimizing some objective) is at least as expensive as the neural network required to represent it. Specifically, these works focus on understanding when the approximating neural network can be sub-exponential in size, thus avoiding the curse of dimensionality. However, to date, these results have only been applicable to *linear* PDEs.

In this paper, we take the first step beyond such work, considering a *nonlinear* family of PDEs and study *nonlinear variational PDEs*. These equations have the form $-\mathrm{div}_{\mathrm{x}}(\partial_{\nabla u} L(x, u, \nabla u)) + \partial_u L(x, u, \nabla u) = f$ and are a (very general) family of *nonlinear Euler-Lagrange* equations. Equivalently, the solution to the PDE is the minimizer of the energy functional $\mathcal{E}(u) = \int_\Omega (L(x, u(X), \nabla u(x)) - f(x)u(x))dx$. This paradigm is very general: it originated with Lagrangian formulations of classical mechanics, and for different L, a variety of variational problems can be modeled or learned [Schmidt and Lipson, 2009, Cranmer et al., 2020]. These PDEs have a variety of applications in scientific domains, e.g., (non-Newtonian) fluid dynamics [Koleva and Vulkov, 2018], meteorology [Weller et al., 2016], and nonlinear diffusion equations [Burgers, 2013].

Our main result is to show that *when the function L has "low complexity", so does the solution*. The notion of complexity we work with is the *Barron norm* of the function, similar to Chen et al. [2021], Lee et al. [2017]. This is a frequently used notion of complexity, as a function with small Barron norm can be represented by a small, two-layer neural network, due to a classical result [Barron, 1993]. Mathematically, our proof techniques are based on "neurally unfolding" an iterative preconditioned gradient descent in an appropriate function space: namely, we show that each of the iterates can be represented by a neural network with Barron norm not much worse than the Barron norm of the previous iterate—along with showing a bound on the number of required steps.

Importantly, our results go beyond the typical non-parametric bounds on the size of an approximator network that can be easily shown by classical regularity results of the solution to the nonlinear variational PDEs [De Giorgi, 1957, Nash, 1957, 1958] along with universal approximation results [Yarotsky, 2017].

4.2 Overview of Results

Let $\Omega := [0, 1]^d$ be a d-dimensional hypercube and let $\partial\Omega$ denote its boundary.

We first define the energy functional whose minimizers are represented by a nonlinear variational PDE—i.e., the Euler-Lagrange equation of the energy functional.

Definition 11 (Energy functional)**.** *For all $u : \Omega \to \mathbb{R}$ such that $u|_{\partial\Omega} = 0$, we consider an energy functional of the following form:*

$$\mathcal{E}(u) = \int_\Omega \Big(L(x, u(x), \nabla u(x)) - f(x)u(x) \Big) dx, \tag{4.1}$$

where $L : \Omega \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ and there exist constants $0 < \lambda \leq \Lambda$ such that for every $x \in \Omega$ the function $L(x, \cdot, \cdot) : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ is smooth and convex, i.e.,

$$\mathrm{diag}([0, \lambda \mathbf{1}_d]) \leq \nabla^2_{(y,z)} L(x, y, z) \leq \mathrm{diag}([\Lambda, \Lambda \mathbf{1}_d]) \tag{4.2}$$

for all $(y, z) \in \mathbb{R} \times \mathbb{R}^d$.

Further, we assume that the function $f : \Omega \to \mathbb{R}$ is such that $\|f\|_{L^2(\Omega)} < \infty$. Note that without loss of generality[1] we assume that $\lambda \leq 1/C_p$ (where C_p is the Poincare constant defined in Theorem 14).

The minimizer $u^\star$ of the energy functional $\mathcal{E}$ exists and is unique. The proof of existence and uniqueness is standard (following essentially along the same lines as Theorem 3.3 in Fernández-Real and Ros-Oton [2020]), and is stated in the following Lemma (with the full proof provided in Section 11.4.1 of the Appendix for completeness).

Lemma 14. *Let $L : \Omega \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ be the function as defined in Definition 11. Then the minimizer of the energy functional $\mathcal{E}$ exists and is unique.*

Writing down the condition for stationarity, we can derive a (nonlinear) elliptic PDE for the minimizer of the energy functional in Definition 11 .

Lemma 15. *Let $u^\star : \Omega \to \mathbb{R}$ be the unique minimizer for the energy functional in Definition 11. Then for all $\varphi \in H^1_0(\Omega)$, $u^\star$ satisfies the following condition:*

$$\begin{aligned} &D\mathcal{E}[u](\varphi) \\ &= \int_\Omega (\partial_{\nabla u} L(x, u, \nabla u) \nabla \varphi + \partial_u L(x, u, \nabla u)\varphi - f\varphi) dx \\ &= 0, \end{aligned} \tag{4.3}$$

where $d\mathcal{E}[u](\varphi)$ denotes the directional derivative of the energy functional calculated at u in the direction of φ. Thus, the minimizers of the energy functional satisfy the following PDE with Dirichlet boundary condition:

$$\begin{aligned} &D\mathcal{E}(u) \\ &:= -\mathrm{div}_{\mathrm{x}}(\partial_{\nabla u} L(x, u, \nabla u)) + \partial_u L(x, u, \nabla u) = f \end{aligned} \tag{4.4}$$

for all $x \in \Omega$ and $u(x) = 0, \forall x \in \partial\Omega$. Here $\mathrm{div}_{\mathrm{x}}$ denotes the divergence operator.

[1] Since λ is a lower bound on the strong convexity constant. If we choose a weaker lower bound, we can always ensure $\lambda \leq 1/C_p$.

The proof for the Lemma can be found in Appendix 11.4.2. Here $-\mathrm{div}_x(\partial_{\nabla u} L(\nabla \cdot))$ and $\partial_u L(x, \cdot, \nabla \cdot)$ are operators that acts on a function (in this case u).[2]

Our goal is to determine if the solution to the PDE in Equation 4.4 can be expressed by a neural network with a small number of parameters. In order do so, we rely on the concept of a *Barron norm*, which measures the complexity of a function in terms of its Fourier representation. We show that if composing with the function partial derivatives of the function L increases the Barron norm of u in a bounded fashion, then the solution to the PDE in Equation 4.4 will have a bounded Barron norm. The motivation for using this norm is a seminal paper [Barron, 1993], which established that any function with Barron norm C can be ϵ-approximated by a two-layer neural network in the L^2 sense by a 2-layer neural network with size $O(C^2/\epsilon)$, thus evading the curse of dimensionality if C is substantially smaller than exponential in d. Informally, we will show the following result:

Theorem 4 (Informal)**.** *Given the function L in Definition 11, such that composing a function with Barron norm b with $\partial_{\nabla u} L$ or $\partial_u L$ produces a function of Barron norm at most $B_L b^p$ for some constants $B_L, p > 0$. Then, $\forall \epsilon > 0$, the minimizer of the energy functional in Definition 11 can be ϵ-approximated in the L^2 sense by a function with Barron norm*

$$O\Big((dB_L)^{\max\{p\log(1/\epsilon), p^{\log(1/\epsilon)}\}}\Big).$$

As a consequence, when ϵ, p, B_L are thought of as constants, we can represent the solution to the Euler-Lagrange PDE Equation 4.4 by a polynomially-sized network, as opposed to an exponentially sized network, which is what we would get by standard universal approximation results and using regularity results for the solutions of the PDE.

We establish this by neurally simulating a preconditioned gradient descent (for a strongly-convex loss) in an appropriate Hilbert space, and show that the Barron norm of each iterate—which is a function—is finite, and at most polynomially bigger than the Barron norm of the previous iterate. We get the final bound by (i) bounding the growth of the Barron norm at every iteration; and (ii) bounding the number of iterations required to reach an ϵ-approximation to the solution. The result in formally stated in Section 4.5.

4.3 Related Work

Over the past few years there has been a growing line of work that utilizes neural networks to parameterize the solution to a PDE. Works such as Weinan et al. [2017], Yu et al. [2017], Sirignano and Spiliopoulos [2018], Raissi et al. [2017] achieved impressive results on a variety of different applications and have demonstrated the empirical efficacy of neural networks in solving high dimensional PDEs. This is a great and promising direction for solving high dimensional PDEs since erstwhile dominant numerical ap-

[2] For a vector valued function $F : \mathbb{R}^d \to \mathbb{R}^d$ we will denote the divergence operator either by $\mathrm{div}_x F$ or by $\nabla \cdot F$, where $\mathrm{div}_x F = \nabla \cdot F = \sum_{i=1}^d \frac{\partial_i F}{\partial x_i}$

proaches like the finite differences and finite element methods [LeVeque, 2007] depend primarily upon discretizing the input space, hence limiting their use for problems on low dimensional input space.

Several recent works look into the theoretical analysis into their representational capabilities has also gained a lot of attention. Khoo et al. [2021] show the existence of a network by discretizing the input space into a mesh and then using convolutional NNs, where the size of the layers is exponential in the input dimension. Sirignano and Spiliopoulos [2018] provide a universal approximation result, showing that for sufficiently regularized PDEs, there exists a multilayer network that approximates its solution. Jentzen et al. [2018], Grohs and Herrmann [2020], Hutzenthaler et al. [2020] show that provided a better-than-exponential dependence on the input dimension for some specific parabolic PDEs, based on a stochastic representation using the Feynman-Kac Lemma, thus limiting the applicability of their approach to PDEs that have such a probabilistic interpretation.

These representational results can be further be utilized towards analyzing the generalization properties of neural network approximations to PDE solutions. For example, Lu et al. [2021] show the generalization analysis for the Deep Ritz method for elliptic equations like the Poisson equation and Lu and Lu [2021] extends their analysis to the Schrodinger eigenvalue problem. Furthermore, Mishra and Molinaro [2020] look at the generalization properties of physics informed neural networks for a linear operators or for non-linear operators with well-defined linearization.

Closest to our work is a recent line of study that has focused on families of PDEs for which neural networks evade the curse of dimensionality—i.e. the solution can be approximated by a neural network with a subexponential size. In Marwah et al. [2021] the authors show that for elliptic PDEs whose coefficients are approximable by neural networks with at most N parameters, a neural network exists that ϵ-approximates the solution and has size $O(d^{\log(1/\epsilon)}N)$. Chen et al. [2021] extends this analysis to elliptic PDEs with coefficients with small Barron norm, and shows that if the coefficients have Barron norm bounded by B, an ϵ-approximate solution exists with Barron norm at most $O(d^{\log(1/\epsilon)}B)$. The work by Chen et al. [2022] derives related results for the Schrödinger equation on the whole space.

As mentioned, while most of previous works show key regularity results for neural network approximations of solution to PDEs, most of their analysis is limited to simple *linear* PDEs. The focus of this paper is towards extending these results to a family of PDEs referred to as nonlinear variational PDEs. This particular family of PDEs consists of many famous PDEs such as $p-$Laplacian (on a bounded domain) and is used to model phenomena like non-Newtonian fluid dynamics and nonlinear diffusion processes. The regularity results for these family of PDEs was posed as Hilbert's XIXth problem. We note that there are classical results like De Giorgi [1957] and Nash [1957, 1958] that provide regularity estimates on the solutions of a nonlinear variational PDE of the form in Equation 4.4. One can easily use these regularity estimates, along with standard universal approximation results Yarotsky [2017] to show that the solutions can be approximated arbitrarily well. However, the size of the resulting networks will be exponentially large (i.e. they will suffer from the curse of dimensionality)—so are of no use for our desired results.

4.4 Notation and Definition

In this section we introduce some key concepts and notation that will be used throughout the paper. For a vector $x \in \mathbb{R}^d$ we use $\|x\|_2$ to denote its ℓ_2 norm. $C^\infty(\Omega)$ is the set of function $f : \Omega \to \mathbb{R}$ that are infinitely differentiable. For a function $F(x, y, z)$ of multiple variables we use $\nabla_x F(x, y, z)$ and $\partial_x F(x, y, z)$ to denote the (partial) derivative w.r.t the variable x (we drop the subscript if the function takes in only a single variable). Similarly, Δ_x denotes the Laplacian operator where the derivatives are taken w.r.t $x \in \mathbb{R}^d$. With a slight abuse of notation, if a function $L : \Omega \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ takes functions u and ∇u as input, we will denote the partial derivatives w.r.t second and third set of coordinates as, $\partial_u L(x, u, \nabla u)$ and $\partial_{\nabla u} L(x, u, \nabla u)$, respectively.

We also define some important function spaces and associated key results below.

Definition 12. *For a vector valued function $g : \mathbb{R} \to \mathbb{R}^d$ we define the $L^p(\Omega)$ norm for $p \in [1, \infty)$ as*

$$\|g\|_{L^p(\Omega)} = \left(\int_\Omega \sum_i^d |g_i(x)|^p dx \right)^{1/p},$$

For $p = \infty$ we have

$$\|g\|_{L^\infty(\Omega)} = \max_i \|g_i\|_{L^\infty(\Omega)},$$

Definition 13. *For a domain Ω, the space of functions $H_0^1(\Omega)$ is defined as,*

$$H_0^1(\Omega) := \{g : \Omega \to \mathbb{R} : g \in L^2(\Omega), \\ \nabla g \in L^2(\Omega), g|_{\partial\Omega} = 0\}.$$

The corresponding norm for $H_0^1(\Omega)$ is defined as, $\|g\|_{H_0^1(\Omega)} = \|\nabla g\|_{L^2(\Omega)}$.

Finally, we will make use of the Poincaré inequality throughout several of our results.

Theorem 5 (Poincaré inequality, Poincaré [1890])**.** *For any domain $\Theta \subset \mathbb{R}^d$ which is open and bounded, there exists a constant $C_p > 0$ such that for all $u \in H_0^1(\Theta)$*

$$\|u\|_{L^2(\Theta)} \leq C_p \|\nabla u\|_{L^2(\Theta)}.$$

This constant can be very benignly behaved with dimension for many natural domains—even dimension independent. One such example are convex domains [Payne and Weinberger, 1960], for which $C_p \leq \pi^2 \text{diam}(\Omega)$. Furthermore, for $\Omega = [0, 1]^d$, the value of C_p can be explicitly calculated and is equal to $1/\pi^2 d$. This is a simple calculation, but we include it for completeness as the following lemma (proved in Section 11.4.3):

Lemma 16. *For the domain $\Omega := [0, 1]^d$, the Poincare constant is equal to $\frac{1}{\pi^2 d}$.*

4.4.1 Barron Norms

For a function $f : [0,1]^d \to \mathbb{R}$ the Fourier transform is defined as,

$$\hat{f}(\omega) = \int_{[0,1]^d} f(x) e^{-i2\pi x^T \omega} dx, \quad \omega \in \mathbb{N}^d, \tag{4.5}$$

where $\mathbb{N}^d$ is the set of vectors with natural numbers as coordinates. The inverse Fourier transform of a function is defined as,

$$f(x) = \sum_{\omega \in \mathbb{N}^d} e^{i2\pi x^T \omega} \hat{f}(\omega) \tag{4.6}$$

The Barron norm is an average of the norm of the frequency vector weighted by the Fourier magnitude $|\hat{f}(\omega)|$.

Definition 14 (Spectral Barron Norm, [Barron, 1993]). *Let Γ define a set of functions defined over $\Omega := [0,1]^d$ such that $\hat{f}(\omega)$ and $\omega \hat{f}(\omega)$ are absolutely summable, i.e.,*

$$\Gamma = \Bigg\{ f : \Omega \to \mathbb{R} : \sum_{\omega \in \mathbb{N}^d} |\hat{f}(\omega)| < \infty, \\ \& \sum_{\omega \in \mathbb{N}^d} \|\omega\|_2 |\hat{f}(\omega)| < \infty \Bigg\}$$

Then we define the spectral Barron norm $\|\cdot\|_{\mathcal{B}(\Omega)}$ as

$$\|f\|_{\mathcal{B}(\Omega)} = \sum_{\omega \in \mathbb{N}^d} (1 + \|\omega\|_2) |\hat{f}(\omega)|.$$

The Barron norm can be thought of as an L_1 relaxation of requiring sparsity in the Fourier basis—which is intuitively why it confers representational benefits in terms of the size of a neural network required. We refer to Barron [1993] for a more exhaustive list of the Barron norms of some common function classes.

The main theorem from Barron [1993] formalizes this intuition, by bounding the size of a 2-layer network approximating a function with small Barron norm:

Theorem 6 (Theorem 1, Barron [1993]). *Let $f \in \Gamma$ such that $\|f\|_{\mathcal{B}(\Omega)} \leq C$ and μ be a probability measure defined over Ω. There exists $a_i \in \mathbb{R}^d$, $b_i \in \mathbb{R}$ and $c_i \in \mathbb{R}$ such that $\sum_{i=1}^k |c_i| \leq 2C$, there exists a function $f_k(x) = \sum_{i=1}^k c_i \sigma\left(a_i^T x + b_i\right)$, such that we have,*

$$\int_\Omega (f(x) - f_k(x))^2 \mu(dx) \lesssim \frac{C^2}{k}.$$

Here σ denotes a sigmoidal activation function, i.e., $\lim_{x \to \infty} \sigma(x) = 1$ and $\lim_{x \to -\infty} \sigma(x) = 0$.

Note that while Theorem 6 is stated for sigmoidal activations like *sigmoid* and *tanh* (after appropriate rescaling), the results are also valid for ReLU activation functions, since $\mathrm{ReLU}(x) - \mathrm{ReLU}(x-1)$ is in fact sigmoidal. We will also need to work with functions that do not have Fourier coefficients beyond some size (i.e. are band limited), so we introduce the following definition:

Definition 15. *We will define the set Γ_W as the set of functions whose Fourier coefficients vanish outside a bounded ball, that is*

$$\begin{aligned}\Gamma_W =& \{f : \Omega \to \mathbb{R} : \textit{s.t. } f \in \Gamma, \\ & \& \ \forall w, \|w\|_\infty \geq W, \hat{f}(w) = 0\}.\end{aligned}$$

Finally, as we will work with vector valued functions, we will also define the Barron norm of a vector-valued function as the maximum of the Barron norms of its coordinates:

Definition 16. *For a vector valued function $g : \Omega \to \mathbb{R}^d$, we define $\|g\|_{\mathcal{B}(\Omega)} = \max_i \|g_i\|_{\mathcal{B}(\Omega)}$.*

4.5 Main Result

Before stating the main result we introduce the key assumption.

Assumption 1. *The function L in Definition 11 can be approximated by a function $\mathbf{L} : \Omega \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ such that there exists a constant $\epsilon_L \in [0, \lambda)$ for all $x \in \Omega$ and $u \in H_0^1(\Omega)$ define $q := (x, u(x), \nabla u(x)) \in \Omega \times \mathbb{R} \times \mathbb{R}^d$*

$$\begin{aligned}\sup_q \|\partial_u L(q) - \partial_u \mathbf{L}(q)\|_2 \leq \epsilon_L \|u(x)\|_2, \\ \textit{and, } \sup_q \|\partial_{\nabla u} L(q) - \partial_{\nabla u} \mathbf{L}(q)\|_2 \leq \epsilon_L \|u(x)\|_2,\end{aligned}$$

Furthermore, we assume that $\mathbf{L}$ is such that for all $g \in H_0^1(\Omega)$, we have $\mathbf{L}(x, g, \nabla g) \in H_0^1(\Omega)$, $\mathbf{L}(x, g, \nabla g) \in \Gamma$ and for all $x \in \Omega$

$$\begin{aligned}\|\partial_u \mathbf{L}(x, g, \nabla g)\|_{\mathcal{B}(\Omega)} \leq B_{\mathbf{L}} \|g\|_{\mathcal{B}(\Omega)}^{p_{\mathbf{L}}}, \\ \textit{and, } \|\partial_{\nabla u} \mathbf{L}(x, g, \nabla g)\|_{\mathcal{B}(\Omega)} \leq B_{\mathbf{L}} \|g\|_{\mathcal{B}(\Omega)}^{p_{\mathbf{L}}}.\end{aligned} \tag{4.7}$$

for some constants $B_{\mathbf{L}} \geq 0$, and $p_{\mathbf{L}} \geq 0$. Finally, if $g \in \Gamma_W$ then $\partial_u \mathbf{L}(x, g, \nabla g) \in \Gamma_{k_{\mathbf{L}} W}$ and $\partial_{\nabla u} \mathbf{L}(x, g, \nabla g) \in \Gamma_{k_{\mathbf{L}} W}$ for a $k_{\mathbf{L}} > 0$.

We refer to *Remark 4* for an example of how the conditions in the assumption manifest for a linear elliptic PDE.

This assumption is fairly natural: it states that the function L is such that its partial derivatives w.r.t u and ∇u can be approximated (up to ϵ_L) by a function $\tilde{L}$ with partial derivatives that have the property that when applied to a function g with small Barron norm, the new Barron norm is not much bigger

than that of g. The constant p specifies the order of this growth. The functions for which our results are most interesting are when the dependence of $B_{\mathbf{L}}$ on d is at most polynomial—so that the final size of the approximating network does not exhibit curse of dimensionality. For instance, we can take L to be a multivariate polynomial of degree up to P: we show in Lemma 23 the constant $B_{\mathbf{L}}$ is $O(d^P)$ (intuitively, this dependence comes from the total number of monomials of this degree), whereas p and k are both $O(P)$.

With all the assumptions stated, we now state our main theorem,

Theorem 7 (Main Result)**.** *Consider the nonlinear variational PDE in Equation 4.4 which satisfies Assumption 1 and let $u^\star \in H_0^1(\Omega)$ denote the unique solution to the PDE. If $u_0 \in H_0^1(\Omega)$ is a function such that $u_0 \in \Gamma_{W_0}$, then for all sufficiently small $\epsilon > 0$, and*

$$T := \left\lceil \log\left(\frac{2}{\epsilon}\frac{\mathcal{E}(u_0) - \mathcal{E}(u^\star)}{\lambda}\right) / \log\left(\frac{1}{1 - \frac{\lambda^6}{(1+C_p)^{10}\Lambda^5}}\right) \right\rceil,$$

there exists a function $u_T \in H_0^1(\Omega)$ such that $u_T \in \Gamma_{(2\pi k_{\mathbf{L}})^T W_0}$ with Barron norm $\|u_T\|_{\mathcal{B}(\Omega)}$ bounded by

$$\begin{aligned}&\Big((1 + \eta 2\pi k_{\mathbf{L}} W_0 (2\pi k_{\mathbf{L}} d + 1) B_{\tilde{L}})\big(1 + \eta \|f\|_{\mathcal{B}(\Omega)}\big)\Big)^{pt + \frac{p^t - 1}{p-1}} \\ &\quad \cdot \Big(\max\{1, \|u_0\|_{\mathcal{B}(\Omega)}^{p^t}\}\Big).\end{aligned} \tag{4.8}$$

Furthermore u_T satisfies $\|u_T - u^\star\|_{H_0^1(\Omega)} \leq \epsilon + \tilde{\epsilon}$ where,

$$\tilde{\epsilon} \leq \frac{\epsilon_L R}{\epsilon_L + \Lambda}\Big(\big(1 + \eta(1 + C_p)^2(\epsilon_L + \Lambda)\big)^T - 1\Big),$$

where $R := \|u^\star\|_{H_0^1(\Omega)} + \frac{1}{\lambda}\mathcal{E}(u_0)$ and $\eta = \frac{\lambda^4}{4(1+C_p)^7\Lambda^4}$.

Remark 1: The function u_0 can be seen as an initial estimate of the solution, that can be refined to an estimate u_T, which is progressively better at the expense of a larger Barron norm. A trivial choice could be $u_0 = 0$, which has Barron norm 1, and which by Lemma 17 would result in $\mathcal{E}(u_0) \leq \Lambda \|u^*\|_{H_0^1(\Omega)}^2$.

Remark 2: The final approximation error has two terms, and note that T goes to infinity as ϵ tends to zero and is a consequence of the way u_T is constructed — by simulating a functional (preconditioned) gradient descent which converges to the solution to the PDE. $\tilde{\epsilon}$ stems from the approximation that we make between $\tilde{L}$ and L, which grows as T increases — it is a consequence of the fact that the gradient descent updates with $\tilde{L}$ and L progressively drift apart as $T \to \infty$.

Remark 3: As in the informal theorem, if we think of $p, \Lambda, \lambda, C_p, k, \|u_0\|_{\mathcal{B}(\Omega)}$ as constants, the theorem implies that $u^\star$ can be ϵ-approximated in the L^2 sense by a function with Barron norm $O\Big((dB_L)^{\max\{p\log(1/\epsilon), p^{\log(1/\epsilon)}\}}\Big)$.

Therefore, combining results from Theorem 8 and Theorem 6 the total number of parameters required to $\epsilon-$approximate the solution $u^\star$ by a $2-$layer neural network is

$$O\left(\frac{1}{\epsilon^2}(dB_L)^{2\max\{p\log(1/\epsilon),p^{\log(1/\epsilon)}\}}\right).$$

Remark 4: The theorem recovers (and vastly generalizes) prior results which bound the Barron norm of linear elliptic PDEs like Chen et al. [2021] over the hypercube. In these results, the elliptic PDE takes the form that for all $u \in H_0^1(\Omega)$, $-\mathrm{div}_x(A\nabla u) + cu = f$ and the functions $A : \mathbb{R}^d \to \mathbb{R}^{d\times d}$ and $c : \mathbb{R}^d \to \mathbb{R}$ are such that $\forall x \in \Omega, A(x)$ is positive definite and $c(x)$ is non-negative and bounded. Further, the functions A and c are assumed to have bounded Barron norm. To recover this setting from our result, consider choosing

$$L(x, u(x), \nabla u(x)) := \frac{1}{2}(\nabla u(x))^T A(x)(\nabla u(x)) + \frac{1}{2}c(x)u(x)^2.$$

For this L, we have $\partial^2_{\nabla u} L(x, u(x), \nabla u(x)) = A(x)$ and $\partial^2_u L(x, u(x), \nabla u(x)) = c(x)$. The conditions in Equation 4.2 in Definition 1 require that $\lambda \le A(x) \le \Lambda$ and $0 \le c(x) \le \Lambda$, which match the conditions on the coefficients A and c in Chen et al. [2021].

Further, by a simple application of Lemma 21, one can show, $\|\partial_{\nabla u} L(x, u, \nabla u)\|_{\mathcal{B}(\Omega)} \le d^2\|A\|_{\mathcal{B}(\Omega)}\|u\|_{\mathcal{B}(\Omega)}$, and $\|\partial_u L(x, u, \nabla u)\|_{\mathcal{B}(\Omega)} \le \|A\|_{\mathcal{B}(\Omega)}\|u\|_{\mathcal{B}(\Omega)}$ and therefore satisfy Equation 4.7 in Assumption 1 with $B_{\tilde{L}} = \max\{d^2\|A\|_{\mathcal{B}(\Omega)}, \|c\|_{\mathcal{B}(\Omega)}\}$ and $p = 1$. Plugging these quantities in Theorem 8, we recover the exact same bound from Chen et al. [2021].

4.6 Proof of Main Result

The proof will proceed by "neurally unfolding" a preconditioned gradient descent on the objective $\mathcal{E}$ in the Hilbert space $H_0^1(\Omega)$. This is inspired by previous works by Marwah et al. [2021], Chen et al. [2021] where the authors show that for a linear elliptic PDE, an objective which is quadratic can be designed. In our case, we show that $\mathcal{E}$ is "strongly convex" in some suitable sense — thus again, bounding the amount of steps needed.

More precisely, the result will proceed in two parts:

1. First, we will show that the sequence of functions $\{u_t\}_{t=0}^\infty$, where $u_{t+1} \leftarrow u_t - \eta(I - \Delta_x)^{-1} d\mathcal{E}(u_t)$ can be interpreted as performing preconditioned gradient descent, with the (constant) preconditioner $(I - \Delta_x)^{-1}$. We show that in some appropriate sense (Lemma 17), $\mathcal{E}$ is strongly convex in $H_0^1(\Omega)$ — thus the updates converge at a rate of $O(\log(1/\epsilon))$.

2. We then show that the Barron norm of each iterate u_{t+1} can be bounded in terms of the Barron norm of the prior iterate u_t. We show this in Lemma 20, where we show that given Assumption 1, $\|u_{t+1}\|_{\mathcal{B}(\Omega)}$ can be bounded as $O(d\|u_t\|^p_{\mathcal{B}(\Omega)})$. By unrolling this recursion we show that the Barron norm of the ϵ-approximation of $u^\star$ is of the order $O(d^{p^T}\|u_0\|^p_{\mathcal{B}(\Omega)})$ where T are the total

steps required for ϵ-approximation and $\|u_0\|_{\mathcal{B}(\Omega)}$ is the Barron norm of the first function in the iterative updates.

We now proceed to delineate the main technical ingredients for both of these parts.

4.6.1 Convergence Rate of Sequence

The proof to show the convergence to the solution $u^\star$ is based on adapting the standard proof (in finite dimension) for convergence of gradient descent when minimizing a strongly convex function f. Recall, the basic idea is to Taylor expand $f(x+\delta) \approx f(x) + \nabla f(x)^T\delta + O(\|\delta\|^2)$. Taking $\delta = \eta\nabla f(x)$, we lower bound the progress term $\eta\|\nabla f(x)\|^2$ using the convexity of f, and upper bound the second-order term $\eta^2\|\nabla f(x)\|^2$ using the smoothness of f.

We follow analogous steps, and prove that we can lower bound the progress term by using some appropriate sense of convexity of $\mathcal{E}$, and upper bound using some appropriate sense of smoothness of $\mathcal{E}$, when considered as a function over $H_0^1(\Omega)$. Precisely, we show:

Lemma 17 (Strong convexity of $\mathcal{E}$ in H_0^1)**.** *If $\mathcal{E}$, L are as in Definition 11, we have*

1. $\forall u, v \in H_0^1(\Omega) : \langle D\mathcal{E}(u), v\rangle_{L^2(\Omega)} = \int_\Omega(-\mathrm{div}_x(\partial_{\nabla u}L(x,u,\nabla u)) + \partial_u(x,u,\nabla u))v dx = \int_\Omega \partial_{\nabla u}L(x,u,\nabla u)\cdot\nabla v + \partial_u L(x,u,\nabla u)v\, dx.$
2. $\forall u, v \in H_0^1(\Omega) : \lambda\|u-v\|^2_{H_0^1(\Omega)} \le \langle D\mathcal{E}(u) - D\mathcal{E}(v), u-v\rangle_{L^2(\Omega)} \le (1+C_p^2)\Lambda\|u-v\|^2_{H_0^1(\Omega)}.$
3. $\forall u, v \in H_0^1(\Omega) : \frac{\lambda}{2}\|\nabla v\|^2_{L^2(\Omega)} + \langle D\mathcal{E}(u) - f, v\rangle_{L^2(\Omega)} \le \mathcal{E}(u+v) - \mathcal{E}(u) \le \langle D\mathcal{E}(u) - f, v\rangle_{L^2(\Omega)} + \frac{(1+C_p)^2\Lambda}{2}\|\nabla v\|^2_{L^2(\Omega)}.$
4. $\forall u \in H_0^1(\Omega) : \frac{\lambda}{2}\|u-u^\star\|^2_{H_0^1(\Omega)} \le \mathcal{E}(u) - \mathcal{E}(u^\star) \le \frac{(1+C_p)^2\Lambda}{2}\|u-u^\star\|^2_{H_0^1(\Omega)}.$

Part 1 is a helpful way to rewrite an inner product of a "direction" v with $D\mathcal{E}(u)$—it is essentially a consequence of integration by parts and the Dirichlet boundary condition. Part 2 and 3 are common proxies of convexity and smoothness: they are ways of formalizing the notion that $\mathcal{E}$ is strongly convex has "Lipschitz gradients", when viewed as a function over $H_0^1(\Omega)$. Finally, Part 4 is a consequence of strong convexity, capturing the fact that if the value of $\mathcal{E}(u)$ is suboptimal, u must be (quantitatively) far from u^*. The proof of the Lemma can be found in Appendix 11.1.1.

When analyzing gradient descent in (finite dimensions) to minimize a loss function $\mathcal{E}$, the standard condition for progress is that the inner product of the gradient with the direction towards the optimum is lower bounded as $\langle D\mathcal{E}(u), u^* - u\rangle_{L^2(\Omega)} \ge \alpha\|u-u^*\|^2_{L^2(\Omega)}$ (we have $L^2(\Omega)$ inner product vs $H_0^1(\Omega)$ norm). From Parts 2 and 3 of Lemma 17 one can readily see that the above condition is only satisfied "with the wrong norm": i.e. we only have $\langle D\mathcal{E}(u), u^* - u\rangle_{L^2(\Omega)} \ge \alpha\|u-u^*\|^2_{H_0^1(\Omega)}$. Moreover, since in general, $\|\nabla g\|_{L^2(\Omega)}$ can be arbitrarily bigger than $\|g\|_{L^2(\Omega)}$, there is no way to upper bound the $H_0^1(\Omega)$ norm by the $L^2(\Omega)$ norm.

We can fix this mismatch by instead doing preconditioned gradient, using the fixed preconditioner $(I - \Delta_x)^{-1}$. Towards that, the main lemma about the preconditioner we will need is the following one:

Lemma 18 (Norms with preconditioning). *For all $u \in H_0^1(\Omega)$ we have*

1. $\|(I - \Delta_x)^{-1}\nabla_x \cdot \nabla_x u\|_{L^2(\Omega)} = \|(I - \Delta_x)^{-1}\Delta_x u\|_{L^2(\Omega)} \leq \|u\|_{L^2(\Omega)}$.
2. $\|(I - \Delta_x)^{-1}u\|_{L^2(\Omega)} \leq \|u\|_{L^2(\Omega)}$
3. $\langle (I - \Delta_x)^{-1}u, u\rangle_{L^2(\Omega)} \geq \frac{1}{1+C_p}\langle (-\Delta_x)^{-1}u, u\rangle_{L^2(\Omega)}$.

The first part of the lemma is a relatively simple consequence of the fact that Δ_x and ∇_x "commute", thus can be re-ordered, and the second part that the operator $(I - \Delta_x)^{-1}$ only decreases the $H_0^1(\Omega)$ norm. The latter lemma can be understood intuitively as $(I - \Delta_x)^{-1}$ and Δ_x^{-1} act as similar operators on eigenfunctions of Δ_x with large eigenvalues (the extra I does not do much) – and are only different for eigenfunctions for small eigenvalues. However, since the smallest eigenvalue is lower bounded by $1/C_p$, their gap can be bounded.

Combining Lemma 17 and Lemma 18, we can show that preconditioned gradient descent exponentially converges to the solution to the nonlinear variational PDE in 4.4.

Lemma 19 (Convergence of Preconditioned Gradient Descent). *Let $u^\star$ denote the unique solution to the PDE in Definition 4.4 For all $t \in \mathbb{N}$, we define the sequence of functions*

$$u_{t+1} \leftarrow u_t - \eta(I - \Delta_x)^{-1}(D\mathcal{E}(u_t) - f). \tag{4.9}$$

where $\eta = \frac{\lambda^4}{4(1+C_p)^7\Lambda^4}$. If $u_0 \in H_0^1(\Omega)$, then after t iterations we have,

$$\mathcal{E}(u_{t+1}) - \mathcal{E}(u^\star) \leq \left(1 - \frac{\lambda^6}{(1 + C_p)^{10}\Lambda^5}\right)(\mathcal{E}(u_0) - \mathcal{E}(u^\star)).$$

The complete proof for convergence can be found in Section 11.1.3 of the Appendix.

Therefore, using the result from Lemma 17 part 4, i.e., $\|u_t - u^\star\|^2_{H_0^1(\Omega)} \leq \frac{2}{\lambda}(\mathcal{E}(u_t) - \mathcal{E}(u^\star))$, we have

$$\begin{aligned}&\|u_t - u^\star\|^2_{H_0^1(\Omega)}\\ &\quad\leq \frac{2}{\lambda}\left(1 - \frac{\lambda^6}{(1 + C_p)^{10}\Lambda^5}\right)^t(\mathcal{E}(u_0) - \mathcal{E}(u^\star)).\end{aligned}$$

and $\|u_T - u^\star\|^2_{H_0^1(\Omega)} \leq \epsilon$ after T steps, where,

$$T \geq \log\left(\frac{\mathcal{E}(u_0) - \mathcal{E}(u^\star)}{\lambda\epsilon/2}\right) / \log\left(\frac{1}{1 - \frac{\lambda^6}{(1+C_p)^{10}\Lambda^5}}\right). \tag{4.10}$$

4.6.2 Bounding the Barron Norm

Having obtained a sequence of functions that converge to the solution $u^\star$, we bound the Barron norms of the iterates. We draw inspiration from Marwah et al. [2021], Lu et al. [2021] and show that the Barron norm of each iterate in the sequence increases the Barron norm of the previous iterate in a bounded fashion. Note that in general, the Fourier spectrum of a composition of functions cannot easily be expressed in terms of the Fourier spectrum of the functions being composed. However, from Assumption 1 we know that the function L can be approximated by $\mathbf{L}$ such that $\partial_{\nabla u}\mathbf{L}(x, u, \nabla u)$ and $\partial_u L(x, u, \nabla u)$ increases the Barron norm of u in a bounded fashion. Thus, if we instead of tracking the iterates in Equation 11.16 we track

$$\tilde{u}_{t+1} = \tilde{u}_t - \eta(I - \Delta)^{-1} D\tilde{\mathcal{E}}(\tilde{u}_t). \tag{4.11}$$

we can derive the following result (the proof is deferred to Section 11.3.1 of the Appendix):

Lemma 20. *For the updates in Equation 4.11, if $\tilde{u}_t \in \Gamma_{W_t}$ then for all $\eta \in (0, \eta]$ we have $\tilde{u}_{t+1} \in \Gamma_{k_{\mathbf{L}} W_t}$ and the Barron norm $\|\tilde{u}_{t+1}\|_{|\mathcal{B}(\Omega)}$ can be bounded as follows,*

$$(1 + \eta(2\pi k_{\mathbf{L}} d + 1) B_{\mathbf{L}} (2\pi W_t)^{p_{\mathbf{L}}}) \|\tilde{u}\|^{p_{\mathbf{L}}}_{\mathcal{B}(\Omega)} + \eta \|f\|_{\mathcal{B}(\Omega)}.$$

The proof consists of using the result in Equation 4.7 about the Barron norm of composition of a function with $\mathbf{L}$, as well as counting the increase in the Barron norm of a function by any basic algebraic operation, as established in Lemma 21. Precisely we show:

Lemma 21 (Barron norm algebra)**.** *If $g, g_1, g_2 \in \Gamma$, then the following set of results hold,*

- *Addition:* $\|g_1 + g_2\|_{\mathcal{B}(\Omega)} \leq \|g_1\|_{\mathcal{B}(\Omega)} + \|g_2\|_{\mathcal{B}(\Omega)}$.
- *Multiplication:* $\|g_1 \cdot g_2\|_{\mathcal{B}(\Omega)} \leq \|g_1\|_{\mathcal{B}(\Omega)} \|g_2\|_{\mathcal{B}(\Omega)}$
- *Derivative: if $h \in \Gamma_W$ for $i \in [d]$ we have $\|\partial_i g\|_{\mathcal{B}(\Omega)} \leq 2\pi W \|g\|_{\mathcal{B}(\Omega)}$.*
- *Preconditioning: if $g \in \Gamma$, then $\|(I - \Delta)^{-1} g\|_{\mathcal{B}(\Omega)} \leq \|g\|_{\mathcal{B}(\Omega)}$.*

The proof for the above lemma can be found in Appendix 11.3.4. It bears similarity to an analogous result in Chen et al. [2021], with the difference being that our bounds are defined in the *spectral* Barron space which is different from the definition of the Barron norm used in Chen et al. [2021]. Other than preconditioning, the other properties follow by a straightforward calculation. For preconditioning, the main observation is that $(I - \Delta)^{-1}$ acts as a diagonal operator in the Fourier basis—thus the Fourier coefficients of $(I - \Delta)^{-1} h$ can be easily expressed in terms of those of h.

Expanding on the recurrence in Lemma 21 we can bound the Barron norm of the function u_T after T iterations as:

Lemma 22. *Given the updates in Equation 4.11 and function $u_0 \in \Gamma_{W_0}$ with Barron norm $\|u_0\|_{\mathcal{B}(\Omega)}$, then after T iterations we have $\tilde{u}_T \in \Gamma_{(2\pi k_{\mathbf{L}})^T W_0}$ and $\|u_0\|_{\mathcal{B}(\Omega)}$ is bounded by,*

$$\begin{aligned}&\big((1+\eta 2\pi k_{\mathbf{L}} W_0(2\pi k_{\mathbf{L}} d+1)B_{\tilde{L}})\big(1+\eta\|f\|_{\mathcal{B}(\Omega)}\big)\big)^{pt+\frac{p^t-1}{p-1}}\\ &\quad\cdot\Big(\max\{1,\|u_0\|^{p^t}_{\mathcal{B}(\Omega)}\}\Big)\end{aligned} \tag{4.12}$$

Finally, we exhibit a natural class of functions that satisfy the main Barron growth property in Equations 4.7. Precisely, we show (multivariate) polynomials of bounded degree have an effective bound on p and B_L:

Lemma 23. *Let $f(x)=\sum_{\alpha,|\alpha|\le P}\Big(A_\alpha\prod_{i=1}^d x_i^{\alpha_i}\Big)$ where α is a multi-index and $x\in\mathbb{R}^d$. If $g:\mathbb{R}^d\to\mathbb{R}^d$ is such that $g\in\Gamma_W$, then we have $f\circ g\in\Gamma_{PW}$ and the Barron norm can be bounded as $\|f\circ g\|_{\mathcal{B}(\Omega)}\le d^{P/2}\Big(\sum_{\alpha,|\alpha|\le P}|A_\alpha|^2\Big)^{1/2}\|g\|^P_{\mathcal{B}(\Omega)}$*

Hence if $\mathbf{L}$ is a polynomial of degree P then using the fact that for a functions $g:\Omega\to\mathbb{R}$ such that $g\in\Gamma_W$, from Lemma 21 $\max\{\|g\|_{\mathcal{B}(\Omega)},\|\nabla g\|_{\mathcal{B}(\Omega)}\}\le 2\pi W\|g\|_{\mathcal{B}(\Omega)}$, we will have

$$\begin{aligned}&\|\mathbf{L}(x,g,\nabla g)\|_{\mathcal{B}(\Omega)}\\ &\le d^{P/2}\left(\sum_{\alpha,|\alpha|\le P}|A_\alpha|^2\right)^{1/2}(2\pi W)^P\|g\|^P_{\mathcal{B}(\Omega)}.\end{aligned}$$

Using the *derivative* result from Lemma 21, the constants in Assumption 1 will take the following values $B_{\mathbf{L}}=d^{P/2}(2\pi W)^{P+1}\Big(\sum_{\alpha,|\alpha|\le P}|A_\alpha|^2\Big)^{1/2}$, and $r=2\pi WP$.

Finally, since we are using an approximation of the function L we will incur an error at each step of the iteration. The following Lemma shows that the error between the iterates u_t and the approximate iterates $\tilde{u}_t$ increases with t. The error is calculated by recursively tracking the error between u_t and $\tilde{u}_t$ for each t in terms of the error at $t-1$. Note that this error can be controlled by using smaller values of η.

Lemma 24. *Let $\mathbf{L}:\mathbb{R}^d\to\mathbb{R}$ be the function satisfying the properties in Assumption 1 and we have*

$$\begin{aligned}\mathcal{E}(u)&=\int_\Omega L(x,u(x),\nabla u(x))-f(x)u(x)\,dx\\ \text{and}\quad\tilde{\mathcal{E}}(u)&=\int_\Omega \mathbf{L}(x,u(x),\nabla u(x))-f(x)u(x)dx.\end{aligned}$$

For $\eta\in(0,\frac{\lambda^4}{4(1+C_p)^7\Lambda^4}]$ consider the sequences,

$$\begin{aligned}u_{t+1}&=u_t-\eta(I-\Delta)^{-1}D\mathcal{E}(u_t),\\ \text{and,}\quad\tilde{u}_{t+1}&=\tilde{u}_t-\eta(I-\Delta)^{-1}D\tilde{\mathcal{E}}(u_t)\end{aligned}$$

then for all $t \in \mathbb{N}$ *and denoting* $R := \|u^\star\|_{H_0^1(\Omega)} + \frac{1}{\lambda}\mathcal{E}(u_0)$ *we have,*

$$\begin{aligned}&\|u_t - \tilde{u}_t\|_{H_0^1(\Omega)} \\ &\quad\leq \frac{\epsilon_L R}{\epsilon_L + \Lambda}\Big(\big(1 + \eta(1 + C_p)^2(\epsilon_L + \Lambda)\big)\Big)^t - 1\Big)\end{aligned}$$

4.7 Conclusion and Future Work

In this work, we take a representational complexity perspective on neural networks, as they are used to approximate solutions of nonlinear elliptic variational PDEs of the form $-\mathrm{div}_{\mathrm{x}}(\partial_{\nabla u} L(x, u, \nabla u)) + \partial_u L(x, u, \nabla u) = f$. We prove that if L is such that composing partial derivatives of L with function of bounded Barron norm increases the Barron norm in a bounded fashion, then we can bound the Barron norm of the solution $u^\star$ to the PDE—potentially evading the curse of dimensionality depending on the rate of this increase. Our results subsume and vastly generalize prior work on the linear case [Marwah et al., 2021, Chen et al., 2021] when the domain is a hypercube. Our proof consists of neurally simulating preconditioned gradient descent on the energy function defining the PDE, which we prove is strongly convex in an appropriate sense.

There are many potential avenues for future work. Our techniques (and prior techniques) strongly rely on the existence of a variational principle characterizing the solution of the PDE. In classical PDE literature, these classes of PDEs are also considered better behaved: e.g. proving regularity bounds is much easier for such PDEs [Fernández-Real and Ros-Oton, 2020]. There are many non-linear PDEs that come without a variational formulation for which regularity estimates are derived using non-constructive methods like comparison principles. It is a wide open question to construct representational bounds for any interesting family of PDEs of this kind. It is also a very interesting question to explore other notions of complexity—e.g. number of parameters in a (potentially deep) network like in Marwah et al. [2021], Rademacher complexity, among others.

Part III

Empirical Validation and Large Models

5 Deep Equilibrium Based Neural Operators for Steady-State PDEs

Abstract: *Data-driven machine learning approaches are being increasingly used to solve partial differential equations (PDEs). They have shown particularly striking successes when training an operator, which takes as input a PDE in some family, and outputs its solution. However, the architectural design space, especially given structural knowledge of the PDE family of interest, is still poorly understood. We seek to remedy this gap by studying the benefits of weight-tied neural network architectures for steady-state PDEs. To achieve this, we first demonstrate that the solution of most steady-state PDEs can be expressed as a fixed point of a non-linear operator. Motivated by this observation, we propose FNO-DEQ, a deep equilibrium variant of the FNO architecture that directly solves for the solution of a steady-state PDE as the infinite-depth fixed point of an implicit operator layer using a black-box root solver and differentiates analytically through this fixed point resulting in $\mathcal{O}(1)$ training memory. Our experiments indicate that FNO-DEQ-based architectures outperform FNO-based baselines with $4\times$ the number of parameters in predicting the solution to steady-state PDEs such as Darcy Flow and steady-state incompressible Navier-Stokes. Finally, we show FNO-DEQ is more robust when trained with datasets with more noisy observations than the FNO-based baselines, demonstrating the benefits of using appropriate inductive biases in architectural design for different neural network based PDE solvers. Further, we show a universal approximation result that demonstrates that FNO-DEQ can approximate the solution to any steady-state PDE that can be written as a fixed point equation.*

5.1 Introduction

Partial differential equations (PDEs) are used to model a wide range of processes in science and engineering. They define a relationship of (unknown) function and its partial derivatives. Most PDEs do not admit a closed form solution, and are solved using a variety of classical numerical methods such as finite element [LeVeque, 2007], finite volume [Moukalled et al., 2016], and spectral methods [Kopriva, 2009, Boyd, 2001]. These methods are often very computationally expensive, both as the ambient dimension grows, and as the desired accuracy increases.

This has motivated a rapidly growing area of research in data-driven approaches to PDE solving. One promising approach involves learning *neural solution operators* [Chen and Chen, 1995, Lu et al., 2019, Bhattacharya et al., 2021, Li et al., 2020b], which take in the coefficients of a PDE in some family and output its solution—and are trained by examples of coefficient-solution pairs.

While several architectures for this task have been proposed, the design space—in particular taking into account structural properties of the PDEs the operator is trained on—is still largely unexplored. Most

present architectures are based on "neuralizing" a classical numerical method. For instance, Li et al. [2020a] take inspiration from spectral methods, and introduce FNO: a trained composition of (parametrized) kernels in Fourier space. Brandstetter et al. [2022] instead consider finite-difference methods and generalize them into (learnable) graph neural networks using message-passing.

Our work focuses on families of PDEs that describe the steady-state of a system (that is, there is no time variable). Namely, we consider equations of the form:

$$L(a(x), u(x)) = f(x), \qquad \forall x \in \Omega, \tag{5.1}$$

where $u : \Omega \to \mathbb{R}^{d_u}$, $a : \Omega \to \mathbb{R}^{d_a}$ and $f : \Omega \to \mathbb{R}^{d_f}$ are functions defined over the domain Ω, and L is a (possibly non-linear) operator. This family includes many natural PDE families like Poisson equations, electrostatic equations, and steady-state Navier-Stokes.

We take inspiration from classical numerical approaches of fast-converging Newton-like iterative schemes [LeVeque, 2007, Faragó and Karátson, 2002] to solve steady-state PDEs, as well as recent theoretical works for elliptic (linear and non-linear PDEs) [Marwah et al., 2021, Chen et al., 2021, Marwah et al., 2022] to hypothesize that very deep, but heavily weight-tied architectures would provide a useful architectural design choice for steady-state PDEs.

In this paper, we show that for steady state equations it is often more beneficial to weight-tie an existing neural operator, as opposed to making the model deeper—thus increasing its size. To this end, we introduce **FNO-DEQ**, a new architecture for solving steady-state PDEs. FNO-DEQ is a deep equilibrium model (DEQ) that utilizes weight-tied FNO layers along with implicit differentiation and root-solvers to approximate the solution of a steady-state PDE. DEQs are a perfect match to the desiderata laid out above: they can be viewed alternately as directly parameterizing the fixed points of some iterative process; or by explicitly expanding some iterative fixed point solver like Newton's or Broyden's method as an infinitely deep, weight-tied model.

Such an architecture has a distinct computational advantage: implicit layer models effectively backpropagate through the infinite-depth network while using only constant memory (equivalent to a single layer's activations). Empirically, we show that for steady-state PDEs, weight-tied and DEQ based models perform better than baselines with $4\times$ the number of parameters, and are robust to training data noise. In summary, we make the following contributions:

- We show the benefits of weight-tying as an effective architectural choice for neural operators when applied to steady-state PDEs.
- We introduce FNO-DEQ, a FNO based deep equilibrium model (DEQ) that uses implicit layers and root solving to approximate the solution of a steady-state PDE. We further attest to the empirical performance of FNO-DEQ by showing that it performs as well as FNO and its variants with $4\times$ number of parameters.

- We show that FNO-DEQ and weight tied architectures are more robust to both input and observation noise, thus showing that weight-tying is a useful inductive bias for architectural design for steady-state PDEs.
- By leveraging the universal approximation results of FNO [Kovachki et al., 2021a] we show that FNO-DEQ based architectures can universally approximate the solution operator for a wide variety of steady-state PDE families.
- Finally, we create a dataset of pairs of steady-state incompressible Navier-Stokes equations with different forcing functions and viscosities, along with their solutions, which we will make public as a community benchmark for steady-state PDE solvers.

5.2 Related Work

Neural network based approaches for solving PDEs can broadly be divided into two categories. First are hybrid solvers [Bar-Sinai et al., 2019, Kochkov et al., 2021, Hsieh et al., 2019] which use neural networks in conjunction with existing numerical solvers. The main motivation is to not only improve upon the existing solvers, but to also replace the more computationally inefficient parts of the solver with a learned counter part. Second set of approaches are full machine learning based approaches that aim to leverage the approximation capabilities of neural networks [Hornik et al., 1989] to directly learn the dynamics of the physical system from observations.

Hybrid solvers like Hsieh et al. [2019] use a neural network to learn a correction term to correct over an existing hand designed solver for a Poisson equation, and also provide convergence guarantees of their method to the solution of the PDE. However, the experiments in their paper are limited to linear elliptic PDEs. Further, solvers like Bar-Sinai et al. [2019] use neural networks to derive the discretizations for a given PDE, thus enabling the use of a low-resolution grid in the numerical solver. Furthermore, Kochkov et al. [2021] use neural networks to interpolate differential operators between grid points of a low-resolution grid with high accuracy. This work specifically focuses on solving Navier-Stokes equations, their method is more accurate than numerical techniques like Direct Numerical Simulation (DNS) with a low-resolution grid, and is also $80\times$ more faster. Brandstetter et al. [2022] introduced a message passing based hybrid scheme to train a hybrid solver and also propose a loss term which helps improve the stability of hybrid solvers for time dependent PDEs. However, most of these methods are equation specific, and are not easily transferable to other PDEs from the same family.

The neural network based approach that has recently garnered the most interest by the community is that of the operator learning framework [Chen and Chen, 1995, Kovachki et al., 2021b, Lu et al., 2019, Li et al., 2020a, Bhattacharya et al., 2021], which uses a neural network to approximate and infinite dimensional operator between two Banach spaces, thus learning an entire family of PDEs at once. Lu et al. [2019] introduces DeepONet, which uses two deep neural networks, referred to as the branch net and trunk net, which are trained concurrently to learn from data. Another line of operator learning framework is that of neural operators Kovachki et al. [2021b]. The most successful methodology for neural operators being the Fourier neural operators (FNO) [Li et al., 2020a]. FNO uses convolution based integral kernels which are evaluated in the Fourier space. Future works like Tran et al. [2021] introduce ar-

chitectural improvements that enables one to train deeper FNO networks, thus increasing their size and improving their the performance on a variety of (time-dependent) PDEs. Moreover, the success of Transformers in domains like language and vision has also inspired transformer based neural operators in works like Li et al. [2022], Hao et al. [2023a] and Liu et al. [2022]. Theoretical results pertaining to the neural operators mostly include universal approximation results Kovachki et al. [2021a], Lanthaler et al. [2022] which show that architectures like FNO and DeepONet can indeed approximate the infinite dimension operators.

In this work, we focus on steady-state equations and show the benefits of weight-tying in improving the performance of FNO for steady-state equations. We show that instead of making a network deeper and hence increasing the size of a network, weight-tied FNO architectures can outperform FNO and its variants $4\times$ its size. We further introduce FNO-DEQ, a deep equilibrium based architecture to simulate an infinitely deep weight-tied network (by solving for a fixed point) with $\mathcal{O}(1)$ training memory. Our work takes inspiration from recent theoretical works like Marwah et al. [2021], Chen et al. [2021], Marwah et al. [2022] which derive parametric rates for some-steady state equations, and in fact prove that neural networks can approximate solutions to some families of PDEs with just $\text{poly}(d)$ parameters, thus evading the curse of dimensionality.

5.3 Preliminaries

We now introduce some key concepts and notation.

Definition 17 ($L^2(\Omega;\mathbb{R}^d)$)**.** *For a domain Ω we denote by $L^2(\Omega;\mathbb{R}^d)$ the space of square integrable functions $g:\Omega\to\mathbb{R}^d$ such that $\|g\|_{L^2(\Omega)}<\infty$, where $\|g\|_{L^2(\Omega)}=\left(\int_\Omega \|g(x)\|_{\ell_2}^2 dx\right)^{1/2}$.*

5.3.1 Neural Operators

Neural operators [Lu et al., 2019, Li et al., 2020a, Bhattacharya et al., 2021, Patel et al., 2021, Kovachki et al., 2023] are a deep learning approach to learning solution operators which map a PDE to its solution. Fourier Neural Operator (FNO) [Li et al., 2020a] is a particularly successful recent architecture parametrized as a sequence of kernel integral operator layers followed by non-linear activation functions. Each kernel integral operator layer is a convolution-based kernel function that is instantiated through a linear transformation in Fourier domain, making it less sensitive to the level of spatial discretization. Specifically, an L-layered FNO $G_\theta:\mathbb{R}^{d_u}\to\mathbb{R}^{d_u}$ with learnable parameters θ, is defined as

$$G_\theta := \mathcal{Q}\circ\mathcal{L}_L\circ\mathcal{L}_{L-1}\circ\cdots\circ\mathcal{L}_1\circ\mathcal{P} \tag{5.2}$$

where $\mathcal{P}:L^2(\Omega;\mathbb{R}^{d_u})\to L^2(\mathbb{R}^{d_v};\mathbb{R}^{d_v})$ and $\mathcal{Q}:L^2(\mathbb{R}^{d_v};\mathbb{R}^{d_v})\to L^2(\mathbb{R}^{d_v};\mathbb{R}^{d_u})$ are projection operators, and $\mathcal{L}_l:L^2(\mathbb{R}^{d_v};\mathbb{R}^{d_v})\to L^2(\mathbb{R}^{d_v};\mathbb{R}^{d_v})$ for $l\in[L]$ is the l^{th} FNO layer defined as,

$$\mathcal{L}_l(v_l)=\sigma(W_l v_l+b_l+\mathcal{K}_l(v_l))). \tag{5.3}$$

Here σ is a non-linear activation function, W_l, b_l are the l^{th} layer weight matrix and bias terms. Finally $\mathcal{K}_l$ is the l^{th} integral kernel operator which is calculated using the Fourier transform as introduced in Li et al. [2020a] defined as follows,

$$\mathcal{K}_l(v_l) = \mathcal{F}^{-1}(R_l \cdot (\mathcal{F}v_l))(x) \qquad \forall x \in \Omega, \tag{5.4}$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ are the Fourier transform and the inverse Fourier transform, with R_l representing the learnable weight-matrix in the Fourier domain. Therefore, ultimately, the trainable parameters θ is a collection of all the weight matrices and biases, i.e, $\theta := \{W_l, b_l, R_l, \cdots, W_1, b_1, R_1\}$.

5.3.2 Equilibrium Models

Equilibrium models [Liao et al., 2018, Bai et al., 2019, Revay et al., 2020, Winston and Kolter, 2020] compute internal representations by solving for a fixed point in their forward pass. Specifically, consider a deep feedforward network with L layers :

$$z^{[i+1]} = f_\theta^{[i]}\left(z^{[i]}; x\right) \quad \text{for } i = 0, ..., L-1 \tag{5.5}$$

where $x \in \mathbb{R}^{n_x}$ is the input injection, $z^{[i]} \in \mathbb{R}^{n_z}$ is the hidden state of i^{th} layer with $z^{[0]} = \mathbf{0}$, and $f_\theta^{[i]} : \mathbb{R}^{n_x \times n_z} \mapsto \mathbb{R}^{n_z}$ is the feature transformation of i^{th} layer, parametrized by θ. Suppose the above model is weight-tied, *i.e.,* $f_\theta^{[i]} = f_\theta, \forall i$, and $\lim_{i\to\infty} f_\theta\left(z^{[i]}; x\right)$ exists and its value is $z^\star$. Further, assume that for this $z^\star$, it holds that $f_\theta(z^\star; x) = z^\star$. Then, equilibrium models can be interpreted as the infinite-depth limit of the above network such that $f_\theta^\infty(z^\star; x) = z^\star$

Under certain conditions[1], and for certain classes of f_θ[2], the output $z^\star$ of the above weight-tied network is a fixed point. A simple way to solve for this fixed point is to use fixed point iterations, *i.e.,* repeatedly apply the update $z^{[t+1]} = f_\theta(z^{[t]}; x)$ some fixed number of times, and backpropagate through the network to compute gradients. However, this can be computationally expensive. Deep equilibrium (DEQ) models [Bai et al., 2019] explicitly solve for $z^\star$ through iterative root finding methods like Broyden's method [Broyden, 1965], Newton's method, Anderson acceleration [Anderson, 1965]. DEQs use implicit function theorem to directly differentiate through the fixed point $z^\star$ at equilibrium, thus requiring constant memory to backpropagate through an infinite-depth network:

$$\frac{\partial z^\star}{\partial \theta} = \left(I - \frac{\partial f_\theta(z^\star; x)}{\partial z^\star}\right)^{-1} \frac{\partial f_\theta(z^\star; x)}{\partial \theta} \tag{5.6}$$

Computing the inverse of Jacobian can quickly become intractable as we deal with high-dimensional feature maps. One can replace the inverse-Jacobian term with an identity matrix *i.e.,* Jacobian-free [Fung et al., 2022] or an approximate inverse-Jacobian [Geng et al., 2021] without affecting the final performance. There are alternate formulations of DEQs [Winston and Kolter, 2020] that guarantee existence

[1] The fixed point can be reached if the dynamical system is globally contractive. This is usually not true in practice for most choices of f_θ, and divergence is possible.

[2] Bai et al. [2019] state that f_θ needs to be stable and constrained. In general, by Banach's fixed point theorem, global convergence is guaranteed if f_θ is contractive over its input domain.

of a unique equilibrium point. However, designing f_θ for these formulations can be challenging, and in this work we use the formulation by Bai et al. [2019].

5.4 Problem setting

We first formally define the system of steady-state PDEs that we will solve for:

Definition 18 (Steady-State PDE)**.** *Given a bounded open set $\Omega \subset \mathbb{R}^d$, a steady-state PDE can be written in the following general form:*

$$L(a(x), u(x)) = f(x), \qquad \forall x \in \Omega \tag{5.7}$$

Here L is a continuous operator, the function $u \in L^2\left(\Omega; \mathbb{R}^{d_u}\right)$ is the unknown function that we wish to solve for and $a \in L^2\left(\Omega; \mathbb{R}^{d_a}\right)$ collects all the coefficients describing the PDE, and $f \in L^2\left(\Omega; \mathbb{R}^{d_f}\right)$ is a function independent of u. We will, for concreteness, assume periodic boundary conditions, i.e. $\forall z \in \mathbb{N}^d, \forall x \in \Omega$ we have $u(x+z) = u(x)$. (Equivalently, $\Omega := \mathbb{T}^d = [0, 2\pi]^d$ can be identified with the torus.) [3] *Finally, we will denote $u^\star : \Omega \to \mathbb{R}$ as the solution to the PDE.*

Steady-state models a system at stationarity, *i.e.,* when some quantity of interest like temperature or velocity no longer changes over time. Classical numerical solvers for these PDEs include iterative methods like Newton updates or conjugate gradient descent, typically with carefully chosen preconditioning to ensure benign conditioning and fast convergence. Furthermore, recent theoretical works [Marwah et al., 2021, Chen et al., 2021, Marwah et al., 2022] show that for many families of PDEs (e.g., steady-state elliptic PDEs that admit a variational formulation), iterative algorithms can be efficiently "neuralized", that is, the iterative algorithm can be represented by a compact neural network, so long as the coefficients of the PDE are also representable by a compact neural network. Moreover, the architectures constructed in these works are heavily weight-tied.

We will operationalize these developments through the additional observation that all these iterative schemes can be viewed as algorithms to find a fixed point of a suitably chosen operator. Namely, we can design an operator $\mathcal{G} : L^2(\Omega; \mathbb{R}^{d_u}) \times L^2(\Omega; \mathbb{R}^{d_f}) \to L^2(\Omega; \mathbb{R}^{d_u})$ [4] such that $u^\star = \mathcal{G}(u^\star, f)$ and a lot of common (preconditioned) first and second-order methods are natural ways to recover the fixed points $u^\star$.

Before describing our architectures, we introduce two components that we will repeatedly use.

[3] This is for convenience of exposition, our methods can readily be extended to other boundary conditions like Dirichet, Neumann etc.

[4] We note that the choice of defining the operator with the forcing function f is made for purely expository purposes the operator $\mathcal{G}$ can be defined as $\mathcal{G} : L^2(\Omega; \mathbb{R}^{d_u}) \times L^2(\Omega; \mathbb{R}^{d_a}) \to L^2(\Omega; \mathbb{R}^{d_u})$ as well.

Definition 19 (Projection and embedding layers)**.** *A projection and embedding layer, respectively* $\mathcal{P} : L^2(\Omega;\mathbb{R}^{d_u}) \times L^2(\Omega;\mathbb{R}^{d_f}) \to L^2(\mathbb{R}^{d_v};\mathbb{R}^{d_v}) \times L^2(\mathbb{R}^{d_v};\mathbb{R}^{d_v})$ *and* $\mathcal{Q} : L^2(\mathbb{R}^{d_v};\mathbb{R}^{d_v}) \to L^2(\mathbb{R}^{d_v};\mathbb{R}^{d_u})$*, are defined as*

$$\mathcal{P}(v,f) = \Big(\sigma\Big(W_P^{(1)}v + b_P^{(1)}\Big), \sigma\Big(W_P^{(2)}f + b_P^{(2)}\Big)\Big),$$
$$\mathcal{Q}(v) = \sigma(W_Q v + b_Q)$$

where $W_P^{(1)} \in \mathbb{R}^{d_u \times d_v}, W_P^{(2)} \in \mathbb{R}^{d_f \times d_v}, W_Q \in \mathbb{R}^{d_v \times d_u}$ *and* $b_P^{(1)}, b_P^{(2)} \in \mathbb{R}^{d_v}, b_Q \in \mathbb{R}^{d_u}$*.*

Definition 20 (Input-injected FNO layer)**.** *An input-injected FNO layer* $\mathcal{L} : L^2(\mathbb{R}^{d_v};\mathbb{R}^{d_v}) \times L^2(\mathbb{R}^{d_v};\mathbb{R}^{d_v}) \to L^2(\mathbb{R}^{d_v};\mathbb{R}^{d_v})$ *is defined as*

$$\mathcal{L}(v,g) := g + \sigma\big(Wv + b + \mathcal{F}^{-1}(R^{(k)} \cdot (\mathcal{F}v)\big). \tag{5.8}$$

where $W \in \mathbb{R}^{d_v \times d_v}$*,* $b \in \mathbb{R}^{d_v}$ *and* $R^{(k)} \in \mathbb{R}^{d_v \times d_v}$ *for all* $k \in [K]$ *are learnable parameters.*

Note the difference between the FNO layer specified above, and the standard FNO layer Equation 5.3 is the extra input g to the layer, which in our architecture will correspond to a projection of (some or all) of the PDE coefficients. We also note that this change to the FNO layer also enables us to learn deeper FNO architectures, as shown in Section 9.8. With this in mind, we can discuss the architectures we propose.

Weight-tied architecture I: Weight-tied FNO The first architecture we consider is a weight-tied version of FNO (introduced in Section 5.3), in which we repeatedly apply (M times) the same transformation in each layer. More precisely, we have:

Definition 21 (FNO Weight-Tied)**.** *We define a* M *times weight-tied neural operator* G_θ^M *as,*

$$G_\theta^M = \mathcal{Q} \circ \underbrace{\mathcal{B}^L \circ \mathcal{B}^L \circ \cdots \circ \mathcal{B}^L}_{M \text{ times}} \circ \mathcal{P}$$

such that: $\mathcal{P}, \mathcal{Q}$ *are projection and embedding layers as in Definition 19 an* $\mathcal{B}^L : L^2(\mathbb{R}^{d_v};\mathbb{R}^{d_v}) \times L^2(\mathbb{R}^{d_v};\mathbb{R}^{d_v}) \to L^2(\mathbb{R}^{d_v};\mathbb{R}^{d_v})$ *is an L-layer FNO block, i.e,* $\mathcal{B}^L = \mathcal{L}_L \circ \mathcal{L}_{L-1} \circ \mathcal{L}_{L-2} \circ \mathcal{L}_1$ *where for all* $l \in [L]$*,* $\mathcal{L}_l(\cdot, \mathcal{P}(f))$ [5] *is an input-injected FNO block as in Definition 20.*

Weight-tied architecture II: FNO-DEQ In cases where we believe a weight-tied G_θ^M converges to some fixed point as $M \to \infty$, unrolling G_θ^M for a large M requires a lot of hardware memory for training: training the model requires one to store intermediate hidden units for each weight-tied layer for backpropagation, resulting in a $\mathcal{O}(M)$ increase in the amount of memory required.

To this end, we use Deep Equilibrium models (DEQs) which enables us to implicitly train $G_\theta := \lim_{M\to\infty} G_\theta^M$ by directly solving for the fixed point by leveraging black-box root finding algorithms like quasi-Newton

[5] We are abusing the notation somewhat and denoting by $\mathcal{P}(f)$ the second coordinate of $\mathcal{P}$, which only depends on f.

methods, [Broyden, 1965, Anderson, 1965]. Therefore we can think of this approach as an infinite-depth (or infinitely unrolled) weight-tied network. We refer to this architecture as **FNO-DEQ**.

Definition 22 (FNO-DEQ). *Given $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{B}^L$ in Definition 21, FNO-DEQ is trained to parametrize the fixed point equation $\mathcal{B}^L(v^\star, \mathcal{P}(f)) = v^\star$ and outputs $u^\star = \mathcal{Q}(v^\star)$.*

Usually, it is non-trivial to differentiate through these black-box root solvers. DEQs enable us to implicitly differentiate through the equilibrium fixed point efficiently without any need to backpropagate through these root solvers, therefore resulting in $\mathcal{O}(1)$ training memory.

5.5 Experiments

Network architectures. We consider the following network architectures in our experiments.

FNO: We closely follow the architecture proposed by Li et al. [2020a] and construct this network by stacking four FNO layers and four convolutional layers, separated by GELU activation [Hendrycks and Gimpel, 2016]. Note that in our current set up, we recover the original FNO architecture if the input to the l^{th} layer is the output of $(l-1)^{\text{th}}$ layer *i.e.,* $v_l = \mathcal{B}_{l-1}(v_{l-1})$.

Improved FNO (FNO++): The original FNO architecture suffers from vanishing gradients, which prohibits it from being made deeper [Tran et al., 2021]. We overcome this limitation by introducing residual connections within each block of FNO, with each FNO block $\mathcal{B}_l$ comprising of three FNO layers $\mathcal{L}$ *i.e.,* $\mathcal{B}_l = \mathcal{L}^l_{L_1} \circ \mathcal{L}^l_{L_2} \circ \mathcal{L}^l_{L_3}$ and three convolutional layers, where $\mathcal{L}$ is defined in Equation 5.8.

Weight-tied network (FNO-WT): This is the weight-tied architecture introduced in Definition 21, where we initialize $v_0(x) = 0$ for all $x \in \Omega$.

FNO-DEQ: As introduced in Definition 22, FNO-DEQ is a weight-tied network where we explicitly solve for the fixed point in the forward pass with a root finding algorithm. We use Anderson acceleration [Anderson, 1965] as the root solver. For the backward pass, we use approximate implicit gradients [Geng et al., 2021] which are light-weight and more stable in practice, compared to implicit gradients computed by inverting Jacobian.

Note that both weight-tied networks and FNO-DEQs leverage weight-tying but the two models differ in the ultimate goal of the forward pass: DEQs explicitly solve for the fixed point during the forward pass, while weight-tied networks trained with backpropagation may or may-not reach a fixed point [Anil et al., 2022]. Furthermore, DEQs require $\mathcal{O}(1)$ memory, as they differentiate through the fixed point implicitly, whereas weight-tied networks need to explicitly create the entire computation graph for backpropagation, which can become very large as the network depth increases. Additionally, FNO++ serves as a non weight-tied counterpart to a weight-tied input-injected network. Like weight-tied networks, FNO++ does not aim to solve for a fixed point in the forward pass.

Experimental setup. We test the aforementioned network architectures on two families of steady-state PDEs: Darcy Flow equation and steady-state Navier-Stokes equation for incompressible fluids. For experiments with Darcy Flow, we use the dataset provided by Li et al. [2020a], and generate our own dataset

for steady-state Navier-Stokes. For more details on the datasets and the data generation processes we refer to Sections 12.2.1 and 12.2.2 of the Appendix. For each family of PDE, we train networks under 3 different training setups: clean data, noisy inputs and noisy observations. For experiments with noisy data, both input and observations, we add noise sampled from a sequence of standard Gaussians with increasing values of variance $\{\mathcal{N}(0, (\sigma_k^2))\}_{k=0}^{M-1}$, where M is the total number of Gaussians we sample from. We set $\sigma_0^2 = 0$ and $\sigma_{\text{max}}^2 = \sigma_{M-1}^2 \leq 1/r$, where r is the resolution of the grid. Thus, the training data includes equal number of PDEs with different levels of Gaussian noise added to their input or observations. We add noise to training data, and always test on clean data. We follow prior work [Li et al., 2020b] and report relative L_2 norm between ground truth $u^\star$ and prediction on test data. The total depth of all networks besides FNO is given by $6B + 4$, where B is the number of FNO blocks. Each FNO block has 3 FNO layers and convolutional layers. In addition, we include the depth due to $\mathcal{P}$, $\mathcal{Q}$, and an additional final FNO layer and a convolutional layer. We further elaborate upon network architectures and loss functions in in 12.1.

5.5.1 Darcy Flow

For our first set of experiments we consider stationary Darcy Flow equations, a form of linear elliptic PDE with in two dimensions. The PDE is defined as follows,

$$\begin{aligned} -\nabla \cdot (a(x)\nabla u(x)) &= f(x), \qquad && x \in (0,1)^2 \\ u(x) &= 0 && x \in \partial(0,1)^2. \end{aligned} \tag{5.9}$$

Note that the diffusion coefficient $a \in L^\infty(\Omega)(\Omega; \mathbb{R}_+)$, i.e., the coefficients are always positive, and $f \in L^2(\Omega; \mathbb{R}^{d_f})$ is the forcing term. These PDEs are used to model the steady-state pressure of fluids flowing through a porous media. They can also be used to model the stationary state of the diffusive process with $u(x)$ modeling the temperature distribution through the space with a defining the thermal conductivity of the medium. The task is to learn an operator $G_\theta : L^2(\Omega; \mathbb{R}^{d_u}) \times L^2(\Omega; \mathbb{R}^{d_a}) \to L^2(\Omega; \mathbb{R}^{d_u})$ such that $u^\star = G_\theta(u^\star, a)$.

We report the results of our experiments on Darcy Flow in Table 12.5. The original FNO architecture does not improve its performance with increased number of FNO blocks $\mathcal{B}$. FNO++ with residual connections scales better but saturates at around 4 FNO blocks. In contrast, FNO-WT and FNO-DEQ with just a *single* FNO block outperform deeper non-weight-tied architectures on clean data and on data with noisy inputs. When observations are noisy, FNO-WT and FNO-DEQ outperform FNO++ with a similar number of parameters, and perform comparably to FNO++ with $4\times$ parameters.

We also report results on shallow FNO-DEQ, FNO-WT and FNO++ architectures. An FNO block in these shallow networks has a single FNO layer instead of three layers. In our experiments, shallow weight-tied networks outperform non-weight-tied architectures including FNO++ with $7\times$ parameters on clean data and on data with noisy inputs, and perform comparably to deep FNO++ on noisy observations. In case of noisy observations, we encounter training instability issues in FNO-DEQ. We believe that this shallow network lacks sufficient representation power and cannot accurately solve for the fixed point during the forward pass. These errors in fixed point estimation accumulate over time, leading to incorrect values of implicit gradients, which in turn result in training instability issues.

Architecture	Parameters	#Blocks	Test error ↓		
			$\sigma^2_{\max} = 0$	$(\sigma^2_{\max})^i = 0.001$	$(\sigma^2_{\max})^t = 0.001$
FNO	2.37M	1	0.0080 ± 5e-4	0.0079 ± 2e-4	0.0125 ± 4e-5
FNO	4.15M	2	0.0105 ± 6e-4	0.0106 ± 4e-4	0.0136 ± 2e-5
FNO	7.71M	4	0.2550 ± 2e-8	0.2557 ± 8e-9	0.2617 ± 2e-9
FNO++	2.37M	1	0.0075 ± 2e-4	0.0075 ± 2e-4	0.0145 ± 7e-4
FNO++	4.15M	2	0.0065 ± 2e-4	0.0065 ± 9e-5	0.0117 ± 5e-5
FNO++	7.71M	4	0.0064 ± 2e-4	0.0064 ± 2e-4	**0.0109 ± 5e-4**
S-FNO++	1.78M	0.66	0.0093 ± 5e-4	0.0094 ± 7e-4	0.0402 ± 6e-3
FNO-WT	2.37M	1	**0.0055 ± 1e-4**	**0.0056 ± 5e-5**	0.0112 ± 4e-4
FNO-DEQ	2.37M	1	**0.0055 ± 1e-4**	**0.0056 ± 7e-5**	0.0112 ± 4e-4
S-FNO-WT	1.19M	0.33	0.0057 ± 3e-5	0.0057 ± 5e-5	0.0112 ± 1e-4
S-FNO-DEQ	1.19M	0.33	0.0056 ± 4e-5	0.0056 ± 5e-5	0.0136 ± 0.011

Table 5.1: Results on Darcy flow: clean data (Col 4),noisy inputs (Col 5) and noisy observations (Col 6) with max variance of added noise being $(\sigma^2_{\max})^i$ and $(\sigma^2_{\max})^t$, respectively. Reported test error has been averaged on three different runs with seeds 0, 1, and 2. Here, S-FNO++, S-FNO-WT and S-FNO-DEQ are shallow versions of FNO++, FNO-WT and FNO-DEQ respectively.

5.5.2 Steady-state Navier-Stokes Equations for Incompressible Flow

We consider the steady-state Navier-Stokes equation for an incompressible viscous fluid in the vorticity form defined on a torus, i.e., with periodic boundary condition,

$$\begin{aligned} u \cdot \nabla \omega &= \nu \Delta \omega + f, \qquad && x \in \Omega \\ \nabla \cdot u &= 0 && x \in \Omega \end{aligned} \tag{5.10}$$

where $\Omega := (0, 2\pi)^2$, and $u : \Omega \to \mathbb{R}^2$ is the velocity and $\omega : \Omega \to \mathbb{R}$ where $\omega = \nabla \times u$, $\nu \in \mathbb{R}_+$ is the viscosity and $f : \Omega \to \mathbb{R}$ is the external force term. We learn an operator $G_\theta : L^2(\Omega; \mathbb{R}^{d_u}) \times L^2(\Omega; \mathbb{R}^{d_f}) \to L^2(\Omega; \mathbb{R}^{d_u})$, such that $u^\star = G_\theta(u^\star, f)$. We train all the models on data with viscosity values $\nu = 0.01$ and $\nu = 0.001$, and create a dataset for steady-state incompressible Navier-Stokes, which we will make public as a community benchmark for steady-state PDE solvers.

Results for Navier-Stokes equation have been reported in Table 5.2 and Table 5.3. For both values of viscosity, FNO-DEQ outperforms other architectures for all three cases: clean data, noisy inputs and noisy observations. FNO-DEQ is more robust to noisy inputs compared to non-weight-tied architectures. For noisy inputs, FNO-DEQ matches the test-error of noiseless case in case of viscosity 0.01 and almost matches the test-error of noiseless case in case of viscosity 0.001. We provide additional results for noise level 0.004 in Appendix 12.5. FNO-DEQ and FNO-WT consistently outperform non-weight-tied architectures for higher levels of noise as well.

In general, DEQ-based architectures are slower to train (upto $\sim$2.5$\times$ compared to feedforward networks of similar size) as we solve for the fixed point in the forward pass. However, their inductive bias provides

performance gains that cannot be achieved by simply stacking non-weight-tied FNO layers. In general, we observe diminishing returns in FNO++ beyond 4 blocks. Additionally, training the original FNO network on more than 4 FNO blocks is challenging, with the network often diverging during training, and therefore we do not include these results in the paper.

Architecture	Parameters	#Blocks	Test error ↓		
			$\sigma^2_{\max} = 0$	$(\sigma^2_{\max})^i = 0.001$	$(\sigma^2_{\max})^t = 0.001$
FNO	2.37M	1	0.184 ± 0.002	0.218 ± 0.003	0.184 ± 0.001
FNO	4.15M	2	0.162 ± 0.024	0.176 ± 0.004	0.152 ± 0.005
FNO	7.71M	4	0.157 ± 0.012	0.187 ± 0.004	0.166 ± 0.013
FNO++	2.37M	1	0.199 ± 0.001	0.230 ± 0.001	0.197 ± 0.001
FNO++	4.15M	2	0.154 ± 0.005	0.173 ± 0.003	0.154 ± 0.006
FNO++	7.71M	4	0.151 ± 0.003	0.165 ± 0.004	0.149 ± 0.003
FNO-WT	2.37M	1	$\mathbf{0.123 \pm 0.004}$	$\mathbf{0.129 \pm 0.004}$	0.124 ± 0.005
FNO-DEQ	2.37M	1	$\mathbf{0.123 \pm 0.005}$	$\mathbf{0.129 \pm 0.004}$	$\mathbf{0.123 \pm 0.006}$

Table 5.2: Results on incompressible steady-state Navier-Stokes (viscosity=0.001): clean data (Col 4), noisy inputs (Col 5) and noisy observations (Col 6) with max variance of added noise being $(\sigma^2_{\max})^i$ and $(\sigma^2_{\max})^t$, respectively. Reported test error has been averaged on three different runs with seeds 0, 1, and 2.

Architecture	Parameters	#Blocks	Test error ↓		
			$\sigma^2_{\max} = 0$	$(\sigma^2_{\max})^i = 0.001$	$(\sigma^2_{\max})^t = 0.001$
FNO	2.37M	1	0.181 ± 0.005	0.186 ± 0.003	0.178 ± 0.006
FNO	4.15M	2	0.138 ± 0.007	0.150 ± 0.006	0.137 ± 0.012
FNO	7.71M	4	0.152 ± 0.006	0.163 ± 0.002	0.151 ± 0.008
FNO++	2.37M	1	0.188 ± 0.002	0.207 ± 0.004	0.187 ± 0.003
FNO++	4.15M	2	0.139 ± 0.004	0.153 ± 0.002	0.140 ± 0.005
FNO++	7.71M	4	0.130 ± 0.005	0.151 ± 0.004	0.128 ± 0.009
FNO-WT	2.37M	1	0.089 ± 0.004	$\mathbf{0.089 \pm 0.003}$	0.089 ± 0.004
FNO-DEQ	2.37M	1	$\mathbf{0.085 \pm 0.005}$	0.090 ± 0.003	$\mathbf{0.087 \pm 0.007}$

Table 5.3: Results on incompressible steady-state Navier-Stokes (viscosity=0.01): clean data (Col 4), noisy inputs (Col 5) and noisy observations (Col 6) with max variance of added noise being $(\sigma^2_{\max})^i$ and $(\sigma^2_{\max})^t$, respectively. Reported test error has been averaged on three different runs with seeds 0, 1, and 2.

5.6 Universal Approximation and Fast Convergence of FNO-DEQ

Though the primary contribution of our paper is empirical, we show (by fairly standard techniques) that FNO-DEQ is a universal approximator, under mild conditions on the steady-state PDEs. Moreover, we also show that in some cases, we can hope the fixed-point solver can converge rapidly.

As noted in Definition 18, we have $\Omega := \mathbb{T}^d$. We note that all continuous function $f \in L^2(\Omega;\mathbb{R})$ and $\int_\Omega |f(x)|dx < \infty$ can be written as, $f(x) = \sum_{\omega\in\mathbb{N}^d} e^{ix^T\omega}\hat{f}_w$. where $\{\hat{f}_\omega\}_{\omega\in\mathbb{N}^d}$ are the Fourier coefficients of the function f. We define as $L^2_N(\Omega)$ as the space of functions such that for all $f_N \in L^2_N(\Omega)$ with Fourier coefficients that vanish outside a bounded ball. Finally, we define an orthogonal projection operator $\Pi_N : L^2(\Omega) \to L^2_N(\Omega)$, such that for all $f \in L^2(\Omega)$ we have,

$$f_n = \Pi_N(f) = \Pi_N\left(\sum_{\omega\in\mathbb{N}^d} f_\omega e^{ix^T\omega}\right) = \sum_{\|\omega\|_\infty \le N} \hat{f}_\omega e^{ix^T\omega}. \tag{5.11}$$

That is, the projection operator Π_N takes an infinite dimensional function and projects it to a finite dimensional space. We prove the following universal approximation result:

Theorem 8. *Let $u^\star \in L^2(\Omega;\mathbb{R}^{d_u})$ define the solution to a steady-state PDE in Definition 18, Then there exists an operator $\mathcal{G} : L^2(\Omega;\mathbb{R}^{d_u}) \times L^2(\Omega;\mathbb{R}^{d_f}) \to L^2(\Omega;\mathbb{R}^{d_u})$ such that, $u^\star = \mathcal{G}(u^\star, f)$. Furthermore, for every $\epsilon > 0$ there exists an $N \in \mathbb{N}$ such that for compact sets $K_u \subset L^2(\Omega;\mathbb{R}^{d_u})$ and $K_f \subset L^2(\Omega;\mathbb{R}^{d_f})$ there exists a neural network $G_\theta : L^2_N(\Omega;\mathbb{R}^{d_u}) \times L^2_N(\Omega;\mathbb{R}^{d_f}) \to L^2_N(\Omega;\mathbb{R}^{d_u})$ with parameters θ, such that,*

$$\sup_{u\in K_u, f\in K_f} \|u^\star - G_\theta(\Pi_N u^\star, \Pi_N f)\|_{L^2(\Omega)} \le \epsilon.$$

The proof for the above theorem is relatively straightforward and provided in Appendix 12.3. The proof uses the fact that $u^\star$ is a fixed-point of the operator $G(u, f) = u - (L(u) - f)$, allowing us to use the the results in Kovachki et al. [2021a] that show a continuous operator can be approximated by a network as defined in Equation 5.2. Note that the choice of G is by no means unique: one can "universally approximate" any operator $G(u, f) = u - A(L(u) - f)$, for a continuous operator A. Such a G can be thought of as a form of "preconditioned" gradient descent, for a preconditioner A. For example, a Newton update has the form $G(u, f) = u - L'(u)^{-1}(L(u) - f)$, where $L' : L^2(\Omega;\mathbb{R}^{d_u}) \to L^2(\Omega;\mathbb{R}^{d_u})$ is the Frechet derivative of the operator L.

The reason this is relevant is that the DEQ can choose to universally approximate a fixed-point equation for which the fixed-point solver it is trained with also converges rapidly. As an example, the following classical result shows that under Lax-Milgram-like conditions (a kind of strong convexity condition), Newton's method converges doubly exponentially fast:

Lemma 25 (Faragó and Karátson [2002], Chapter 5)**.** *Consider the PDE defined Definition 18, such that $d_u = d_v = d_f = 1$. such that $L'(u)$ defines the Frechet derivative of the operator L. If for all $u, v \in L^2(\Omega;\mathbb{R})$ we have $\|L'(u)v\|_{L^2(\Omega)} \ge \lambda\|v\|_{L^2(\Omega)}$ and $\|L'(u) - L'(v)\|_{L^2(\Omega)} \le \Lambda\|u - v\|_{L^2(\Omega)}$ for $0 < \lambda \le \Lambda < \infty$, then for the Newton update, $u_{t+1} \leftarrow u_t - L'(u_t)^{-1}(L(u_t) - f)$, with $u_0 \in L^2(\Omega;\mathbb{R})$, there exists an $\epsilon > 0$, such that $\|u_T - u^\star\|_{L^2(\Omega)} \le \epsilon$ if $T \ge \log\left(\log\left(\frac{1}{\epsilon}\right)/\log\left(\frac{2\lambda^2}{\Lambda\|L(u_0)-f\|_{L^2(\Omega)}}\right)\right)$.*

For completeness, we include the proof of the above lemma in the Appendix (Section 12.4). We note that the conditions of the above lemma are satisfied for elliptic PDEs like Darcy Flow, as well as many variational non-linear elliptic PDEs (e.g., those considered in Marwah et al. [2022]). Hence, we can expect FNO-DEQs to quickly converge to the fixed point, since they employ quasi-Newton methods like Broyden and Anderson methods [Broyden, 1965, Anderson, 1965].

6 On the Benefits of Memory for Modeling Time-Dependent PDEs

Abstract: *Data-driven techniques have emerged as a promising alternative to traditional numerical methods. For time-dependent PDEs, many approaches are Markovian—the evolution of the trained system only depends on the current state, and not the past states. In this work, we investigate the benefits of using memory for modeling time-dependent PDEs: that is, when past states are explicitly used to predict the future. Motivated by the Mori-Zwanzig theory of model reduction, we theoretically exhibit examples of simple (even linear) PDEs, in which a solution that uses memory is arbitrarily better than a Markovian solution. Additionally, we introduce Memory Neural Operator (MemNO), a neural operator architecture that combines recent state space models (specifically, S4) and Fourier Neural Operators (FNOs) to effectively model memory. We empirically demonstrate that when the PDEs are supplied in low resolution or contain observation noise at train and test time, MemNO significantly outperforms the baselines without memory—with up to* $6\times$ *reduction in test error. Furthermore, we show that this benefit is particularly pronounced when the PDE solutions have significant high-frequency Fourier modes (e.g., low-viscosity fluid dynamics) and we construct a challenging benchmark dataset consisting of such PDEs.*

6.1 Introduction

Time-dependent partial differential equations (PDEs) are central to modeling various scientific and physical phenomena, necessitating the design of accurate and computationally efficient solvers. Recently, data-driven approaches based on neural networks [Li et al., 2021b, Lu et al., 2019] have emerged as an attractive alternative to classical numerical solvers, such as finite element and finite difference methods [LeVeque, 2007]. Classical approaches are computationally expensive in high dimension and struggle with PDEs which are sensitive to initial conditions. Learned approaches can often negotiate these difficulties better, at least for the PDE family they are trained on.

One example of a data-driven approach is learning a *neural solution operator*, which for a time-dependent PDE learns to predict future states based on previous ones [Li et al., 2021a, 2022]. Recent works [Tran et al., 2023, Lippe et al., 2023b] suggest that optimal performance across various PDE families can be achieved by conditioning the models only on the immediate past state—i.e., treating the system as Markovian. In contrast, other works propose architectures that explicitly use memory of past states [Li et al., 2021a, 2022, Hao et al., 2024a]. None of these works elucidate whether and when modeling memory is helpful.

In this work, we demonstrate that when the solution of the PDE is only *partially observed* (e.g. observed at low resolution), explicitly modeling memory can be beneficial. Partial observability is natural in many practical settings. This could be due to limited resolution of the measurement devices collecting the data, inherent observational errors in the system, or prohibitive computational difficulty in generating high-quality synthetic data. This can lead to significant information loss, particularly in systems like turbulent flows [Pope, 2001] or shock formation in fluid dynamics [Christodoulou, 2007], where PDEs change abruptly in space and time. In such situations, classical results from dynamical systems (Mori-Zwanzig theory), suggest that the system becomes strongly non-Markovian.

More precisely, Mori-Zwanzig theory [Mori, 1965, Zwanzig, 1961, Ma et al., 2018] is an ansatz to understand the evolution of a subspace of a system (e.g., the span of the k largest Fourier components). Under certain conditions, this evolution can be divided into a *Markovian term* (the evolution of the chosen subspace under the PDE), a *memory term* (which is a weighted sum of the values of all previous iterates in the chosen subspace), and an *"unobservable" term*, which depends on the values of the initial conditions orthogonal to the selected subspace.

The main focus of this paper is studying *when* explicitly modeling this memory term is useful. We give an example of a very simple (in fact, linear) PDE where we show theoretically that the solution which takes into account the memory term can be arbitrarily better than the Markovian solution. We also provide a way to *operationalize* the Mori-Zwanzig formalism by introducing **Memory Neural Operator (MemNO)**, a neural operator architecture that combines the Fourier Neural Operator (FNO) [Li et al., 2021a, Tran et al., 2023] to model the spatial dynamics of the PDE, and the S4 state space model [Gu et al., 2022b, 2023] to maintain a compressed representation of the past states. We show that MemNO outperforms its Markovian (memoryless) counterpart in PDEs observed on low resolution grids or with observation noise —achieving up to $6\times$ less test error. Our contributions are as follows:

- We identify a setting in which explicitly modeling memory is helpful: namely, when there is a combination of *lossy observations* of the solution of the PDE (e.g., due to limited resolution or observation noise) and significant contributions from *high-frequency Fourier modes* in the solution.

- Even in simple (in fact, linear) PDEs, we *theoretically* show the memory term can result in a solution that is (arbitrarily) closer to the correct solution, compared to the Markovian approximation —in particular when the operator describing the PDE "mixes" the observed and unobserved subspace.

- Across several families of one-dimensional and two-dimensional PDEs, we *empirically* demonstrate that when the input is supplied on a low-resolution grid, or contains observation noise, neural operators with memory outperform Markovian operators by a significant margin. More precisely, to *operationalize* memory, we introduce MemNO, a neural operator architecture combining Fourier Neural Operators (FNOs) and S4, which achieves the best performance across several Markovian and memory baselines.

- We observe that many current benchmarks for PDE solvers predominantly include PDEs in which there is little contribution from high-frequency Fourier modes. Consequently, we *construct more challenging datasets* where the solutions have significant high-frequency modes, which we believe will be of broader significance to the community beyond testing the effects of memory— especially

given recent meta-studies suggesting many current PDE benchmarks are too easy [McGreivy and Hakim, 2024].

6.2 Related Work

Data-driven neural solution operators [Chen and Chen, 1995, Bhattacharya et al., 2021, Lu et al., 2019, Kovachki et al., 2023] have emerged as the dominant approach for approximating PDEs, given their ability to model multiple families of PDEs at once, and their computational efficiency at inference time. Many architectures have been proposed to improve their performance across different families of PDEs: Li et al. [2021a] introduced the Fourier Neural Operator (FNO), a resolution invariant architecture that uses a convolution-based integral kernel evaluated in the Fourier space; Tran et al. [2023] later introduced the Factorized FNO (FFNO) architecture, which builds upon and improves the FNO architecture by adding separable spectral layers and residual connections; Cao [2021] proposed a Transformer method with linear attention over the spatial sequence; other recent works have used U-Net-based architectures [Gupta and Brandstetter, 2023, Rahman et al., 2023].

Focusing on memory, Tran et al. [2023] performed ablations that suggest the Markov assumption is optimal and outperforms models that use the history of past timesteps as input. Lippe et al. [2023b] performed a similar study for long rollouts of the PDE solution and concluded the optimal performance is indeed achieved under the Markovian assumption. We show that these findings can be replicated only when the resolution of the observation grid is high. On the other hand, we show that MemNO effectively models memory to achieve much superior performance than Markovian operators in low resolution, while not dropping performance in the high resolution case.

Our work is motivated by the Mori-Zwanzig formalism [Zwanzig, 1961, Mori, 1965] which shows that a partial observation of the current state of the system can be compensated using memory of past states. Ma et al. [2018] also study the effects of memory, but when modeling the dynamics of a single PDE by a neural architecture. The authors draw parallels to the Mori-Zwanzig equations and LSTM [Hochreiter and Schmidhuber, 1997] to model the dynamics of the k largest Fourier components of a time-dependent 1-D Kuramoto-Sivashinsky and 2-D shear flow equations, for a single PDE. However, in our work, we study the benefits of memory in neural operator settings, i.e., we have a single model that learns the dynamics of an entire family of PDE at once, and also show conditions under which not maintaining memory can result in arbitrarily large errors.

6.3 Preliminaries

In this section, we introduce several definitions, as well as background on the Mori-Zwanzig formalism as applied to our setting.

6.3.1 Partial Differential Equations (PDEs)

Definition 23 (Space of square integrable functions)**.** *For integers d, V and an open set $\Omega \subset \mathbb{R}^d$, we define $L^2\big(\Omega; \mathbb{R}^V\big)$ as the space of square integrable functions $u : \Omega \to \mathbb{R}^V$ such that $\|u\|_{L^2} \leq \infty$, where $\|u\|_{L^2} = \big(\int_\Omega \|u(x)\|_2^2 dx\big)^{\frac{1}{2}}$.*

Definition 24 (Restriction)**.** *Given a function $u : \Omega \to \mathbb{R}^V$ and a subset $A \subset \Omega$, we denote $u|_A$ as the restriction of u to the domain A, i.e. $u|_A : A \to \mathbb{R}^V$.*

The general form of the PDEs we consider in this paper will be the following:

Definition 25 (Time-Dependent PDE)**.** *For an open set $\Omega \subset \mathbb{R}^d$ and an interval $[0, T] \subset \mathbb{R}$, a Time-Dependent PDE is the following expression:*

$$\frac{\partial u}{\partial t}(t, x) = \mathcal{L}[u](t, x), \qquad \forall t \in [0, T], x \in \Omega, \tag{6.1}$$

$$u(0, x) = u_0(x), \qquad \forall x \in \Omega, \tag{6.2}$$

$$\mathcal{B}[u|_{\partial\Omega}](t) = 0, \qquad \forall t \in [0, T] \tag{6.3}$$

where $\mathcal{L} : L^2\big(\Omega; \mathbb{R}^V\big) \to L^2\big(\Omega; \mathbb{R}^V\big)$ is a differential operator in x which is independent of time, $u_0(x) \in L^2\big(\Omega; \mathbb{R}^V\big)$ and $\mathcal{B}$ is an operator defined on the boundary of $\partial\Omega$, commonly referred to as the boundary condition.

Finally, we will frequently talk about a grid of a given resolution:

Definition 26 (Equispaced grid with resolution f)**.** *Let $\Omega = [0, L]^d$. An equispaced grid with resolution f in Ω is the following set $\mathcal{S} \subset \mathbb{R}^d$:*

$$\mathcal{S} = \left\{ \left(i_1 \frac{L}{f}, \cdots, i_k \frac{L}{f} \right) \middle| 0 \leq i_k \leq f - 1 \text{ for } 1 \leq k \leq d \right\}.$$

We will also denote by $|\mathcal{S}|$ the number of points in $\mathcal{S}$.

In our theory and experiments, we will work with periodic boundary conditions. For completeness, we give a precise definition of periodic boundary conditions for the PDE defined in Definition 25:

Definition 27 (Periodic Boundary Conditions)**.** *For a PDE given by Definition 25 with $\Omega = [0, L]^d$, we define the periodic boundary conditions as the condition:*

$$u(x_1, \cdots, x_{k-1}, 0, x_{k+1}, \cdots x_d) = u(x_1, \cdots, x_{k-1}, L, x_{k+1}, \cdots x_d)$$

for all $(x_1, \cdots, x_{k-1}, x_{k+1}, \cdots, x_L) \in [0, L]^{d-1}$ and all $k = 1, \cdots, d$.

6.3.2 Mori-Zwanzig Formalism

The Mori-Zwanzig formalism [Zwanzig, 2001] considers the setting in which an equation is known for a full system, but only a part of it is observed. It leverages the knowledge of past states of a system to compensate for the loss of information that arises from the partial observation of the current state. In our work, partial observation can refer to observing the solution at a discretized grid in space or only observing the Fourier modes up to a critical frequency. In the context of time-dependent PDEs, the Mori-Zwanzig principle is formalized as the *Nakajima–Zwanzig equation* [Nakajima, 1958].

We will give an overview of the Nakajima-Zwanzig equation and set up the notation for the rest of the paper. Assume we have a PDE as in Definition 25. Let $\mathcal{P} : L^2\big(\Omega; \mathbb{R}^V\big) \to L^2\big(\Omega; \mathbb{R}^V\big)$ be a linear projection operator. We define $\mathcal{Q} = I - \mathcal{P}$, where I is the identity operator. In our setting, for the PDE solution at timestep t $u_t \in L^2\big(\Omega; \mathbb{R}^V\big)$, $\mathcal{P}[u_t]$ is the part of the solution that we observe and $\mathcal{Q}[u_t]$ is the unobserved part. Thus, the initial information we receive for the system is $\mathcal{P}[u_0]$. Applying $\mathcal{P}$ and $\mathcal{Q}$ to Equation 6.1 and using $u = \mathcal{P}[u] + \mathcal{Q}[u]$, we get:

$$\frac{\partial}{\partial t}\mathcal{P}[u](t,x) = \mathcal{P}\mathcal{L}[u](t,x) = \mathcal{P}\mathcal{L}\mathcal{P}[u](t,x) + \mathcal{P}\mathcal{L}\mathcal{Q}[u](t,x) \tag{6.4}$$

$$\frac{\partial}{\partial t}\mathcal{Q}[u](t,x) = \mathcal{Q}\mathcal{L}[u](t,x) = \mathcal{Q}\mathcal{L}\mathcal{P}[u](t,x) + \mathcal{Q}\mathcal{L}\mathcal{Q}[u](t,x) \tag{6.5}$$

Solving for 6.5 yields $\mathcal{Q}[u](t,x) = \int_0^t \exp \mathcal{Q}\mathcal{L}(t-s)\mathcal{Q}\mathcal{L}\mathcal{P}[u](s,x)ds + e^{\mathcal{Q}\mathcal{L}t}\mathcal{Q}[u_0](t,x)$.

Plugging into 6.4, we obtain a *Generalized Langevin Equation* [Mori, 1965] for $\mathcal{P}[u]$:

$$\frac{\partial}{\partial t}\mathcal{P}[u](t,x) = \mathcal{P}\mathcal{L}\mathcal{P}[u](t,x) + \mathcal{P}\mathcal{L}\int_0^t \exp(\mathcal{Q}\mathcal{L}(t-s))\mathcal{Q}\mathcal{L}\mathcal{P}[u](s,x)ds + \mathcal{P}\mathcal{L}e^{\mathcal{Q}\mathcal{L}t}\mathcal{Q}[u_0](t,x) \tag{6.6}$$

We will refer to the first summand on the right hand side of Equation 6.6 as the **Markovian** term because it only depends on $\mathcal{P}[u](t,x)$, the second summand as the **memory** term because it depends on $\mathcal{P}[u](s,x)$ for $0 \leq s \leq t$, and the third summand as the **unobserved residual** as it depends on $\mathcal{Q}[u_0]$ which is never observed.

Since Equation 6.6 is exact, it is equivalent to solving the full system. The term that is typically most difficult to compute is the memory term, and many methods to approximate it have been proposed.

In the *physics literature*, some techniques include a perturbation expansion of the exponential $\exp(\mathcal{Q}\mathcal{L}(t-s))$ [Breuer and Petruccione, 2002], or approximations using operators defined in $\mathcal{P}\big[L^2\big(\Omega; \mathbb{R}^V\big)\big]$ [Shi and Geva, 2003, Zhang et al., 2006, Montoya-Castillo and Reichman, 2016, Kelly et al., 2016]. In the *classical numerical PDE solver* literature, the memory term has been approximated by leveraging the structure of the orthogonal dynamics of the $\mathcal{P}$ semi-group [Gouasmi et al., 2017], and the Mori-Zwanzig formalism has been applied to a variety of fluid dynamics PDEs [Parish and Duraisamy, 2017]. In the *machine learning literature*, Ma et al. [2018] develop the equations for the case when the operator $\mathcal{P}$ keeps

only the k largest Fourier modes, and designed a hybrid approach where the memory term was approximated with an LSTM [Hochreiter and Schmidhuber, 1997]. In this work, we consider a single neural operator that learns the temporal (e.g. memory) and spatial dynamics of a PDE at once.

6.4 Theoretical motivation for memory: a simple example

In this section, we provide a simple, but natural example of a (linear) PDE, along with (in the nomenclature of Section 6.3.2) a natural projection operator given by a *Fourier truncation measurement operator*, such that the memory term in the generalized Langevin equation (GLE) can have an arbitrarily large impact on the quality of the calculated solution. We will work with periodic functions over $[0, 2\pi]$ which have a convenient basis:

Definition 28 (Basis for 2π-periodic functions)**.** *A function $f : \mathbb{R} \to \mathbb{R}$ is 2π-periodic if $f(x + 2\pi) = f(x)$. We can identify 2π-periodic functions with functions over the torus $T := \{e^{i\theta} : \theta \in \mathbb{R}\} \subseteq \mathbb{C}$ by the map $\tilde{f}(e^{ix}) = f(x)$. Note that $\{e^{ixn}\}_{n\in\mathbb{Z}}$ is a basis for the set of 2π-periodic functions.*

We will define the following measurement operator:

Definition 29 (Fourier truncation measurement)**.** *The operator $\mathcal{P}_k : L^2(T;\mathbb{R}) \to L^2(T;\mathbb{R})$ acts on $f \in L^2(T;\mathbb{R})$, $f(x) = \sum_{n=-\infty}^{\infty} a_n e^{inx}$ as $\mathcal{P}_k(f) = \sum_{n=-k}^{k} a_n e^{inx}$.*

For notational convenience, we will also define the functions $\{\mathbf{e}_n\}_{n\in\mathbb{Z}}$, where $\mathbf{e}_n(x) := e^{-inx} + e^{inx}$. Now, we consider the following operator to define a linear time-dependent PDE:

Proposition 2. *Let $\mathcal{L} : L^2(T;\mathbb{R}) \to L^2(T;\mathbb{R})$ be defined as $\mathcal{L}u(x) = -\Delta u(x) + B \cdot (e^{-ix} + e^{ix})u(x)$ for $B > 0$. Then, we have:*

$$\forall 1 \leq n \in \mathbb{N},\ \ \mathcal{L}(\mathbf{e}_n) = n^2\mathbf{e}_n + B(\mathbf{e}_{n-1} + \mathbf{e}_{n+1}) \quad \& \quad \mathcal{L}(\mathbf{e}_0) = 2B\mathbf{e}_1$$

The crucial property of this operator is that it acts by "mixing" the n-th Fourier basis with the $(n-1)$-th and $(n+1)$-th: thus information is propagated to both the higher and lower-order part of the spectrum. Given the above proposition, we can easily write down the evolution of a PDE with operator $\mathcal{L}$ in the basis $\{\mathbf{e}_n\}_{n\in\mathbb{Z}}$:

Proposition 3. *Let $\mathcal{L}$ be defined as in Proposition 2. Consider the PDE*

$$\begin{aligned} \frac{\partial}{\partial t}u(t,x) &= \mathcal{L}u(t,x) \\ u(0,x) &= \sum_{n\in\mathbb{N}_0} a_n(0)\mathbf{e}_n \end{aligned}$$

Let $u(t,x) = \sum_{n\in\mathbb{N}_0} a_n^{(t)}\mathbf{e}_n$. *Then, the coefficients* $a_n^{(t)}$ *satisfy:*

$$\forall 1 \leq n \in \mathbb{N}, \quad \frac{\partial}{\partial t}a_n^{(t)} = n^2 a_n^{(t)} + B\left(a_{n-1}^{(t)} + a_{n+1}^{(t)}\right) \tag{6.7}$$

$$\frac{\partial}{\partial t}a_0^{(t)} = 2Ba_1^{(t)} \tag{6.8}$$

With this setup in mind, we will show that as B grows, the memory term in Equation 6.6 can have an arbitrarily large effect on the calculated solution:

Theorem 9 (Effect of memory)**.** *Consider the operator* $\mathcal{L}$ *defined in Proposition 2, the Fourier truncation operator* $\mathcal{P}_1$, *and let* $\mathcal{Q} = I - \mathcal{P}_1$. *Let* $u(0,x)$ *have the form in Proposition 3 for* $B > 0$ *sufficiently large, and let* $a_n^{(0)} > 0, \forall n > 0$. *Consider the memoryless and memory-augmented PDEs:*

$$\frac{\partial u_1}{\partial t} = \mathcal{P}_1\mathcal{L}u_1 \tag{6.9}$$

$$\frac{\partial u_2}{\partial t} = \mathcal{P}_1\mathcal{L}u_2 + \mathcal{P}_1\mathcal{L}\int_0^t \exp \mathcal{Q}\mathcal{L}(t-s)\mathcal{Q}\mathcal{L}u_2(s)ds \tag{6.10}$$

with $u_1(0,x) = u_2(0,x) = \mathcal{P}_1 u(0,x)$. *Then,* u_1 *and* u_2 *satisfy:*

$$\forall t > 0, \|u_1(t) - u_2(t)\|_{L_2} \gtrsim Bt\|u_1(t)\|_{L_2} \tag{6.11}$$

$$\forall t > 0, \|u_1(t) - u_2(t)\|_{L_2} \gtrsim Bt\exp(\sqrt{2}Bt) \tag{6.12}$$

Remark 3. *Note that the two conclusions of the theorem mean that both the absolute difference, and the relative difference between the PDE including the memory term Equation 6.10 and not including the memory term Equation 6.9 can be arbitrarily large as* $B, t \to \infty$.

Remark 4. *The choice of* $\mathcal{L}$ *is made for ease of calculation of the Markovian and memory term. Conceptually, we expect the solution to Equation 6.10 will differ a lot from the solution to Equation 6.9 if the action of the operator* $\mathcal{L}$ *tends to "mix" components in the span of* $\mathcal{P}$ *and the span of* $\mathcal{Q}$.

Remark 5. *If we solve the equation* $\frac{\partial}{\partial t}u(t,x) = \mathcal{L}u(t,x)$ exactly, *we can calculate that* $\|u(t)\|_{L_2}$ *will be on the order of* $\exp(2Bt)$. *This can be seen by writing the evolution of the coefficients of* $u(t)$ *in the basis* $\{\mathbf{e}_n\}$, *which looks like:* $\frac{\partial}{\partial t}\begin{pmatrix} a_0 \\ a_1 \\ \dots \end{pmatrix} = \mathcal{O}\begin{pmatrix} a_0 \\ a_1 \\ \dots \end{pmatrix}$ *where* $\mathcal{O}$ *is roughly a tridiagonal Toeplitz operator* $\mathcal{O} = \begin{pmatrix} & \vdots & \vdots & \vdots & \vdots & \\ \dots & B & n^2 & B & 0 & \dots \\ \dots & 0 & B & (n+1)^2 & B & \dots \\ & \vdots & \vdots & \vdots & \vdots & \end{pmatrix}$. *The largest eigenvalue of this operator can be shown to be on the order of at least* $2B$ *(Equation 4 in Noschese et al. [2013]). The Markovian term results in a solution of order* $\exp(\sqrt{2}Bt)$ *(Equation 13.3 and Equation 13.4), which is multiplicatively smaller by a factor of*

$\exp((2-\sqrt{2})Bt)$. *The result in this Theorem shows that the memory-based PDE Equation 6.10 results in a multiplicative "first order" correction which can be seen by Taylor expanding* $\exp(\sqrt{2}Bt) \approx 1+\sqrt{2}Bt+\frac{1}{2}(\sqrt{2}B)^2t^2+\ldots$.

6.5 Experimental Setup

6.5.1 Dataset generation

PDEs with high-frequency Fourier modes: From the expression for the memory term in Equation 6.6 and the presence of high-frequency terms in the solution of the PDE of Theorem 9, we should intuitively expect that memory will be most useful when the PDE solutions contain significant contributions from high-frequency Fourier modes[1]. Nevertheless, current benchmarks like PDEBench [Takamoto et al., 2023] rarely contain PDEs whose solutions have substantial high-frequency components, as we quantitatively show in Appendix **??**. A solution which predominantly contains low-frequency Fourier modes can be accurately approximated by its Fourier truncation (Definition 29), so it can be represented by a finite-dimensional space, which implies that the unobserved part of the solution ($\mathcal{Q}[u]$ in the notation of Section 6.3.2) should be small.

Therefore, we construct a new benchmark dataset which is specifically designed to contain PDEs in which the high-frequency Fourier modes have substantial contribution. Specifically, we generate a benchmark from solutions to the Kuramoto-Sivashinsky equation with low viscosity (Section 6.6.1). In the case of Navier-Stokes (Section 6.6.2) and Burgers' equation (Section **??**), we directly take datasets from previous works. Details on data generation procedure are provided in Appendix **??**.

Datasets with different resolutions: To construct our datasets, we first take discretized trajectories of a PDE on a *high resolution* discretized spatial grid $\mathcal{S}^{HR} \subset \mathbb{R}^d$, i.e. $u(t) \in \mathbb{R}^{|\mathcal{S}^{HR}|}$. We then produce datasets that consist of *lower resolution* versions of the above trajectories, i.e. on a grid $\mathcal{S}^{LR}$ of lower resolution f, and show the performance of models that were trained and tested at such resolution. For 1-dimensional datasets, the discretized trajectory on $\mathcal{S}^{LR}$ is obtained by cubic interpolation of the trajectory in the highest resolution grid. In 2D, the discretized trajectory is obtained by downsampling.

6.5.2 Training and evaluation procedure

Task: Let $u \in \mathcal{C}\big([0,T]; L^2\big(\Omega; \mathbb{R}^V\big)\big)$ be the solution of the PDE given by Definition 25. Let $\mathcal{S}$ be an equispaced grid in Ω with resolution f, and let $\mathcal{T}$ be another equispaced grid in $[0,T]$ with N_t+1 points. Given $u_0(x)|_{\mathcal{S}}$, our goal is to predict $u(t,x)|_{\mathcal{S}}$ for $t \in \mathcal{T}$ using a neural operator.

[1] Note, this is meant to be an intuitive rule-of-thumb rather than a formal statement. In general, the "observation" operator and the PDE will interact in complicated ways, but the combination of low-resolution grids and examining high-frequency components in the Fourier basis seems to be very predictive in practice.

Training objective: As is standard, we proceed by empirical risk minimization on a dataset of trajectories. More specifically, given a loss function $\ell : (\mathbb{R}^{|\mathcal{S}|}, \mathbb{R}^{|\mathcal{S}|}) \to \mathbb{R}$, a dataset of training trajectories $\left(u(t,x)^{(i)}\right)_{i=0}^{N}$, and parametrized maps $\mathcal{G}_t^\Theta : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ for $t \in \mathcal{T}$, we optimize:

$$\Theta^* = \text{argmin}_\Theta \frac{1}{N}\sum_{i=0}^{N-1}\frac{1}{N_t}\sum_{t=1}^{N_t}\ell\Big(u(t,x)^{(i)}|_{\mathcal{S}}, \mathcal{G}_t^\Theta\Big[u_0^{(i)}(x)|_{\mathcal{S}}\Big]\Big)$$

Training and evaluation metric: Our training loss and evaluation metric is *normalized Root Mean Squared Error (nRMSE)*:

$$\text{nRMSE}(u(t,x)|_{\mathcal{S}}, \hat{u}(t)) := \frac{\|u(t,x)|_{\mathcal{S}} - \hat{u}(t)\|_2}{\|u(t,x)|_{\mathcal{S}}\|_2},$$

where $\|\cdot\|_2$ is the Euclidean norm in $\mathbb{R}^{|\mathcal{S}|}$.

Further details on training hyperparameters are given in Appendix 13.1.

6.5.3 Architecture Framework: Memory Neural Operator

In this section we describe Memory Neural Operator (MemNO), a deep learning framework to incorporate memory into neural operators. Let NO_t^Θ be a neural operator with L layers, and denote $\text{NO}_t^\Theta[u_0]$ the prediction of the solution of the PDE at time t. We will assume that this Neural Operator follows the Markovian assumption, i.e. we can write:

$$\text{NO}_{t_{i+1}}^\Theta[u_0] = r_{\text{out}} \circ \ell_L \circ \ell_{L-1} \circ ... \circ \ell_0 \circ r_{\text{in}}[\text{NO}_{t_i}^\Theta[u_0]], \tag{6.13}$$

where $r_{\text{in}} : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|\times h_0}$ and $r_{\text{out}} : \mathbb{R}^{|\mathcal{S}|\times h_{L+1}} \to \mathbb{R}^{|\mathcal{S}|}$ are projector operators; $\ell_j : \mathbb{R}^{|\mathcal{S}|\times h_j} \to \mathbb{R}^{|\mathcal{S}|\times h_{j+1}}$ are parametrized layers; and h_j is the dimension of the j-th hidden layer. Essentially, the solution for each new timestep is obtained by applying the *same* L layers to the immediately previous predicted timestep.

Our goal is to define a network $\mathcal{G}_t^\Theta$ that builds upon NO_t^Θ and can incorporate memory. For this, we take inspiration from the Mori-Zwanzig formalism summarized in Section 6.3.2. Comparing Equation 6.13 with Equation 6.6, we identify $\ell_L \circ \ell_{L-1} \circ ... \circ \ell_0$ with the Markov term which models the spatial dynamics. To introduce the memory term, we interleave an additional residual sequential layer $\mathcal{M}$ that acts on hidden representations of the solution at previous timesteps. Concretely, the MemNO architecture can be written as:

$$\mathcal{G}_{t_{i+1}}^\Theta[u_0] = r_{\text{out}} \circ \ell_L \circ ... \circ \ell_{k+1} \circ \mathcal{M} \circ \ell_k \circ ... \circ \ell_0 \circ r_{\text{in}}\Big[\mathcal{G}_{t_i}^\Theta[u_0], \mathcal{G}_{t_{i-1}}^\Theta[u_0], ..., u_0\Big],$$

where $-1 \leq k \leq L$ is a chosen hyperparameter.[2] For notation, we will refer to $v^{(j)}(t') \in \mathbb{R}^{|S| \times h_j}$ as the hidden representation at the j-th layer for a timestep $t' \leq t_i$, and $v^{(j)}(t', x) \in \mathbb{R}^{h_j}$ as the value of such hidden representation at a spatial point $x \in \mathcal{S}$. Then, the spatial ℓ_j layers are understood to be applied timestep-wise, i.e. $\ell_j\left[v^{(j)}(t_i), ..., v^{(j)}(t_0)\right] := \left[\ell_j[v^{(j)}(t_i)], ..., \ell_j[v^{(j)}(t_0)]\right]$, and analogously for r_{in} and r_{out}. Thus, the ℓ_j layers still follow the Markovian assumption. The memory is introduced through $\mathcal{M}$, which is a sequential model that uses the history of the previous timesteps to predict the next one. For computational efficiency, we consider a sequential model $\mathcal{M} : \mathbb{R}^{i \times h_k} \longrightarrow \mathbb{R}^{h_k}$ that is applied to each element of the spatial dimension $|\mathcal{S}|$ independently, i.e. for each $x \in \mathcal{S}$, $\left(\mathcal{M}[v^{(k)}(t_i), ..., v^{(k)}(t_0)]\right)(x) := \mathcal{M}[v^{(k)}(t_i, x), ..., v^{(k)}(t_0, x)]$. We provide PyTorch pseudocode for this architecture in Appendix **??**.

Note that our MemNO framework can be combined with *any* existing neural operator layer ℓ, and with any (causal) sequential model $\mathcal{M}$. Thus it provides a modular architecture design framework which we hope can serve as a useful tool for practitioners.

6.5.4 Instantiating the Memory Neural Operator framework: S4FFNO

For our experiments, we introduce S4 Factorized Fourier Neural Operator (S4FFNO), which instantiates the MemNO framework by combining the Factorized Fourier Neural Operator (FFNO) [Tran et al., 2023] as the Markovian neural operator and S4 [Gu et al., 2022b] as the sequential layer. We choose S4 models over recurrent architectures like LSTM [Hochreiter and Schmidhuber, 1997] due to superior performance in modeling long range dependencies [Gu et al., 2022b, Tay et al., 2020], ease of training, and favorable memory and computational scaling with both state dimension and sequence length. An ablation comparing S4 to LSTM and Transformers is provided in Appendix 13.2.

6.6 Memory helps in low-resolution and input noise: a case study

In this section we present a case study for several PDEs of practical interest, showing that neural operators with memory confer accuracy benefits when the data is supplied in low resolution or with observation noise. We will use three Markovian baselines: The Galerkin Transformer **(GKT)** [Cao, 2021], the U-Net Neural Operator **(U-Net)** [Gupta and Brandstetter, 2023], and the Factorized Fourier Neural Operator **(FFNO)** [Tran et al., 2023]. For a memory-augmented baseline, we consider the Multi Input Factorized Fourier Neural Operator **(Multi input FFNO)**, which takes as input the last 4 timesteps of the solution of the PDE to predict the next one, as proposed in the original FNO paper [Li et al., 2021a], yet using the architectural design of FFNO. The architectural details for all the models are elaborated upon in Appendix **??**.

[2] $k = L$ refers to inserting M after all the S layers, and $k = -1$ refers to inserting M as the first layer. In Appendix 13.2.2, we show our experiments are not very sensitive to the choice of k.

Architecture	Uses memory	Resolution	nRMSE ↓			
			KS			Burgers'
			$\nu = 0.075$	$\nu = 0.1$	$\nu = 0.125$	$\nu = 0.001$
GKT	No	32	0.588	0.601	0.314	0.356
U-Net	No	32	0.542	0.511	0.249	0.188
FFNO	No	32	0.500	0.446	0.187	0.207
Multi Input FFNO	Yes	32	0.364	0.308	0.092	0.099
S4FFNO (Ours)	Yes	32	**0.139**	**0.108**	**0.031**	**0.053**
GKT	No	64	0.401	0.120	0.016	0.349
U-Net	No	64	0.147	0.062	0.022	0.171
FFNO	No	64	0.107	0.033	**0.004**	0.146
Multi Input FFNO	Yes	64	0.108	0.046	0.005	0.054
S4FFNO (Ours)	Yes	64	**0.036**	**0.011**	**0.004**	**0.037**
GKT	No	128	0.028	0.013	0.007	0.307
U-Net	No	128	0.033	0.027	0.014	0.112
FFNO	No	128	**0.006**	**0.004**	**0.002**	0.099
Multi Input FFNO	Yes	128	0.057	0.052	0.023	**0.028**
S4FFNO (Ours)	Yes	128	0.008	0.005	0.003	0.030

Table 6.1: nRMSE values at different resolutions for Burgers' and KS with different viscosities. S4FFNO achieves up to 6x less error than its memoryless counterpart (FFNO) in KS at resolution 32. The final time of KS is 2.5 seconds and it contains 25 timesteps. The final times of Burgers' is 1.4 seconds and it contains 20 timesteps. For the prediction at time t, S4FFNO has access to the (compressed) memory of all previous timesteps, whereas Multi Input FFNO takes as input the previous four timesteps. More details on training are given in Appendix 13.1, and on the Burgers' experiment in Appendix **??**.

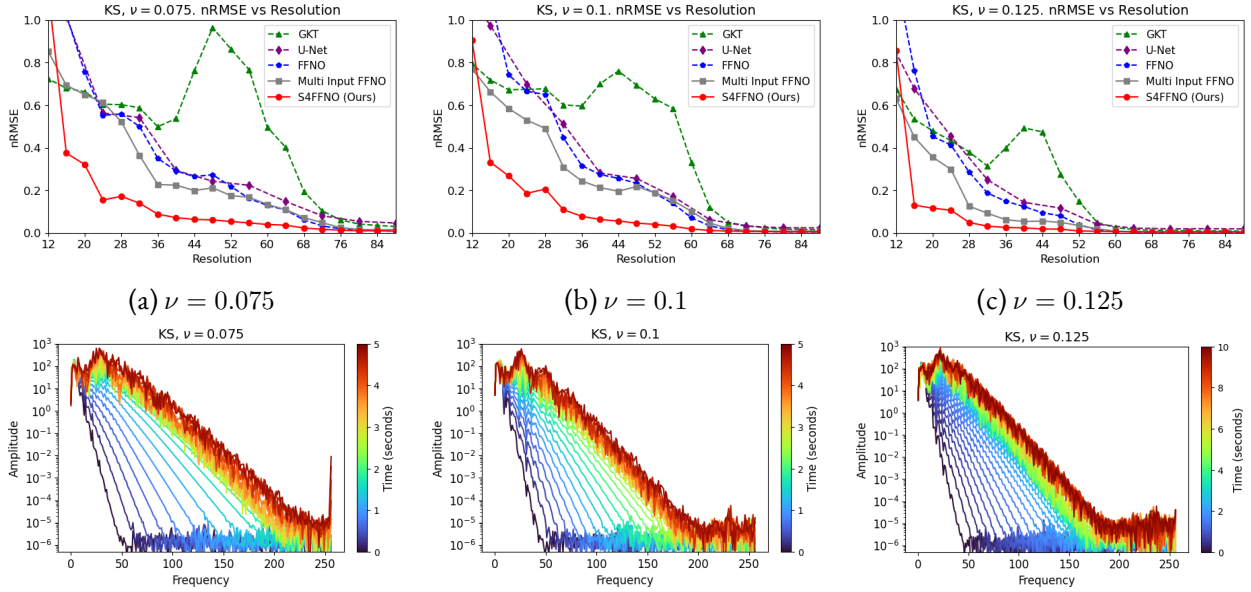
6.6.1 Kuramoto–Sivashinsky equation (1D): study in low-resolution

The Kuramoto-Sivashinsky equation (KS) is a nonlinear PDE that is used as a modeling tool in fluid dynamics, chemical reaction dynamics, and ion interactions. Due to its chaotic behavior it can model instabilities in various physical systems. For viscosity $\nu \in \mathbb{R}_+$, it is written as $u_t + uu_x + u_{xx} + \nu u_{xxxx} = 0$. We generated datasets for KS at different viscosities and resolutions, and show the results in Table 6.1. At resolutions 32 and 64, the memory models (S4FFNO and Multi Input FFNO) outperform the Markovian baselines. In particular, S4FFNO can achieve up to $6\times$ less error than the best performing Markovian Neural Operator (FFNO) and additionally $3\times$ less error than Multi Input FFNO. We note that S4FFNO only has around 1% more parameters than FFNO (see Table **??**).

By contrast, at resolution 128, FFNO has similar performance compared to S4FFNO, and it outperforms Multi Input FFNO. This is in agreement with other works which propose following the Markovian assumption in neural operators [Tran et al., 2023, Lippe et al., 2023b], where it is argued that incorporating previous timesteps as input is not necessary and can lead to difficulties in learning, as it seems to happen with Multi Input FFNO. By contrast, S4FFNO effectively models memory when it is useful (resolutions 32 and 64) without compromising performance at higher resolutions.

In Figure 6.1 we show the performance of all models across a continuous range of resolutions. It can be seen that there is a "cutoff" resolution at which memory models (i.e. S4FFNO) start outperforming Markovian (i.e. FFNO) by a large margin. Very importantly, this cutoff resolution depends on the viscosity, being around 76 when $\nu = 0.075$, 68 when $\nu = 0.1$, and 52 when $\nu = 0.125$. In the KS equation, a lower viscosity leads to the appearance of higher frequencies in the Fourier spectrum (see second row of Figure 6.1), which are not well captured at low resolutions. Thus, we identify the *resolution relative to the Fourier frequency spectrum of the solution* as a key factor for the improved performance of MemNO over memoryless neural operators. We provide a similar study on 1D Burgers equation in Appendix **??**.

We note that even if the initial condition does not contain high frequencies, in the KS equation high frequencies will appear as the system evolves—indeed, this dataset was generated with initial conditions whose maximum Fourier mode was 8.



(a) $\nu = 0.075$ (b) $\nu = 0.1$ (c) $\nu = 0.125$

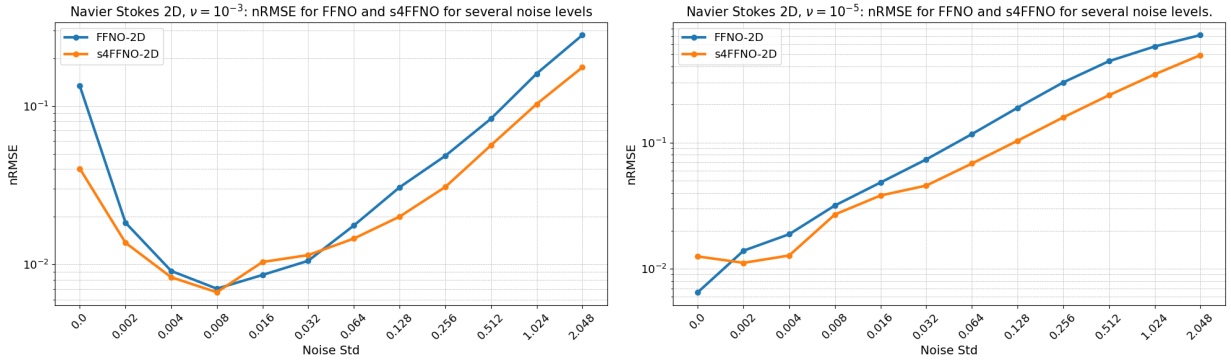
Figure 6.1: (First row) nRMSE for several models in the KS dataset at different resolutions, where each column is a different viscosity. The final time is $T = 2.5s$ and there are $N_t = 25$ timesteps. (Second row) A visualization of the whole frequency spectrum at each of the 25 timesteps for a single trajectory in the dataset. The spectrum is obtained with the ground truth solution at resolution 512.

6.6.2 Navier-Stokes equation (2D): study in observation noise

The Navier-Stokes equation describes the motion of a viscous fluid. Like in Li et al. [2021a], we consider the incompressible form in the 2D unit torus, which is given by:

$$\begin{aligned} \frac{\partial w(x,t)}{\partial t} + u(x,t)\cdot\nabla w(x,t) &= \nu\Delta w(x,t) + f(x), && x\in(0,1)^2, t\in(0,T] \\ \nabla\cdot u(x,t) &= 0, && x\in(0,1)^2, t\in[0,T] \\ w(x,0) &= w_0(x), && x\in(0,1)^2 \end{aligned}$$

Where $w = \nabla \times u$ is the vorticity, $w_0 \in L^2((0,1)^2; \mathbb{R})$ is the initial vorticity, $\nu \in \mathbb{R}_+$ is the viscosity coefficient, and $f \in L^2((0,1)^2; \mathbb{R})$ is the forcing function. In general, the lower the viscosity, the more rapid the changes in the solution and the harder it is to solve it numerically or with a neural operator. We investigate the effect of memory when adding i.i.d. Gaussian noise to the inputs of our neural networks. The noise is sampled i.i.d. from a Gaussian distribution $\mathcal{N}(0, \sigma)$, and then added to training and test inputs. During training, for each trajectory a different noise (with the same σ) is sampled at each iteration of the optimization algorithm. The targets in training and testing represent our ground truth, and do not contain added noise. In Figure 6.2a, we show the results for $\nu = 10^{-3}$ when adding noise levels from



(a) $\nu = 10^{-3}, T = 16s, N_t = 32$ (b) $\nu = 10^{-5}, T = 3.2s, N_t = 32$

Figure 6.2: nRMSE of FFNO-2D and S4FFNO-2D trained on Navier-Stokes 2D with different noise standard deviations σ added to training and test inputs. Two configurations of viscosity ν and final time T are shown.

$\sigma = 0.0$ (no noise) to $\sigma = 2.048$. S4FFNO-2D outperforms FFNO-2D across most noise levels, and the difference between the two is especially significant for noise levels beyond 0.128, where FFNO-2D is around 50% higher than S4FFNO-2D (note the logarithmic scale). For this viscosity, adding small levels of noise actually helps training, which was also observed in other settings in Tran et al. [2023]. Figure 6.2b shows the same experiment performed with $\nu = 10^{-5}$. Again, S4FFNO-2D outperforms FFNO-2D across most noise levels. FFNO-2D losses are similarly around 50% higher for noise levels above 0.032. In this viscosity, adding these levels of noise does not help performance.

6.6.3 Relationship with fraction of unobserved modes

In this section, we provide a simple experiment to quantify the effect of the fraction of unobserved modes on the performance of memory based models. Precisely, suppose $u \in L^2(\Omega; \mathbb{R}^V)$ is the solution of a 1-dimensional PDE at a certain timestep, and a_n for $n \in \mathbb{Z}$ is its Fourier Transform. If we observe it at a resolution f, we can only estimate its top $\lfloor \frac{f}{2} \rfloor$ modes[3]. Thus, we define ω_f as the ratio of unobserved modes at resolution f:

$$\omega_f := \frac{\sum_{|n| > \lfloor \frac{f}{2} \rfloor} |a_n|^2}{\sum_{n \in \mathbb{Z}} |a_n|^2} \tag{6.14}$$

[3] This is a consequence of the Nyquist–Shannon sampling theorem.

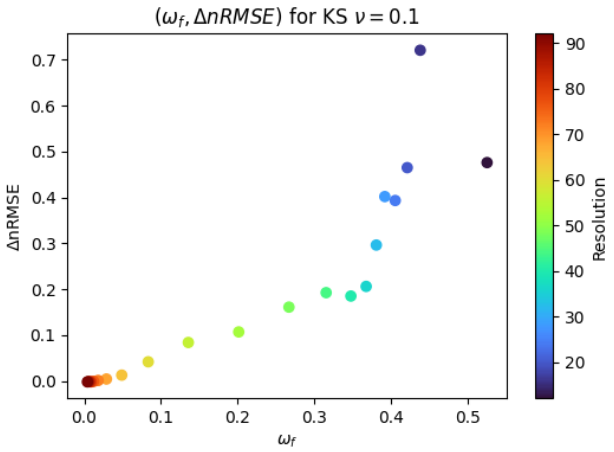


Figure 6.3: Values of ω_f and the difference in nRMSE between FFNO and S4FFNO for different resolutions in the KS experiment of Section 6.6.1 with $\nu = 0.1$. ω_f is averaged across all trajectories in the dataset and across all timesteps.

ω_f is an approximate indicator of the amount of information that is lost when the solution of the PDE is observed at resolution f. In practice, ω_f can be computed by approximating the a_n with the discrete Fourier modes of the solution in the highest resolution available. We show that there is a positive correlation between ω_f and the difference in nRMSE between FFNO and S4FFNO for the KS experiment in Figure 6.3, and also the for Burgers' experiments of Appendix **??** in Figure **??**. This demonstrates the benefits of memory as a way to compensate for missing information in the observations.

6.7 Conclusion and Future Work

We study the benefits of maintaining memory while modeling time dependent PDE systems. When we only observe part of the initial conditions (for example, PDEs observed on low-resolution or with input noise), the system is no longer Markovian, and the dynamics depend on a *memory term*. Taking inspiration from the Mori-Zwanzig formalism, we introduce MemNO, an architecture that combines Fourier Neural Operators (FNO) to model the spatial dynamics of the PDE, and the S4 sequence model to incorporate memory of past states. Through our experiments on different 1D and 2D PDEs, we show that the MemNO architecture outperforms the memoryless baselines, particularly when the solution to the PDE has large components on high-frequency Fourier modes.

We present several avenues for future work. First, our experiments on observation noise are limited to the setting where the input noise is i.i.d. Further, extending the experiments and observing the effects of memory in more real-world settings (for example, with non-i.i.d. noise or in the presence of aliasing) seems fertile ground for future work, and also necessary to ensure that the application of this method does not have unintended negative consequences when broadly applied in society. Lastly, while we limit our study of the effects of memory to FNO-based architectures, performing similar studies for different architectures like Transformer based neural operators [Hao et al., 2023a] and diffusion based operators [Lippe et al., 2023b] is an interesting direction for future work.

7 UPS: Efficiently Building Foundation Models for PDE Solving via Cross-Modal Adaptation

Abstract: *We present Unified PDE Solvers (UPS), a data- and compute-efficient approach to developing unified neural operators for diverse families of spatiotemporal PDEs from various domains, dimensions, and resolutions. UPS embeds different PDEs into a shared representation space and processes them using a FNO-transformer architecture. Rather than training the network from scratch, which is data-demanding and computationally expensive, we warm-start the transformer from pretrained LLMs and perform explicit alignment to reduce the modality gap while improving data and compute efficiency. The cross-modal UPS achieves state-of-the-art results on a wide range of 1D and 2D PDE families from PDEBench, outperforming existing unified models using 4 times less data and 26 times less compute. Meanwhile, it is capable of few-shot transfer to unseen PDE families and coefficients.*

7.1 Introduction

Partial Differential Equations (PDEs) play a pivotal role in modeling and understanding real-world phenomena, such as fluid dynamics and heat transfer. Although there exists a rich body of classical PDE solvers [Boyd, 2001, LeVeque, 2007, Moukalled et al., 2016] that are effective and mathematically proven, these solvers often incur substantial computational costs when used in practice, as they need to be re-run every time a coefficient or boundary condition changes. This motivates the development of *neural operators* [Li et al., 2020a, Chen and Chen, 1995, Lu et al., 2019], which use neural networks to approximate a solution map for a PDE family and can generalize to different initial/boundary conditions or coefficients. While existing neural operators [Lippe et al., 2023a, Hao et al., 2023a, Marwah et al., 2023] have demonstrated strong performance on various practical benchmarks [Takamoto et al., 2022, Gupta and Brandstetter, 2022], most of them are designed to work with a *single PDE family*. Training a separate model for each PDE family remains costly.

Several recent works, such as Subramanian et al. [2023], MPP [McCabe et al., 2023], and DPOT[1] [Hao et al., 2024b], have taken initial steps towards developing foundation models for PDE solving, learning unified operators that transfer across PDE families. These models are *pretrained from scratch* using extensive amounts of data and compute. For example, MPP is trained with over 80,000 PDE trajectories on 8 NVIDIA H100 GPUs for 200,000 steps. Despite the development costs, the resulting models are

[1] This work was done at the same time as ours.

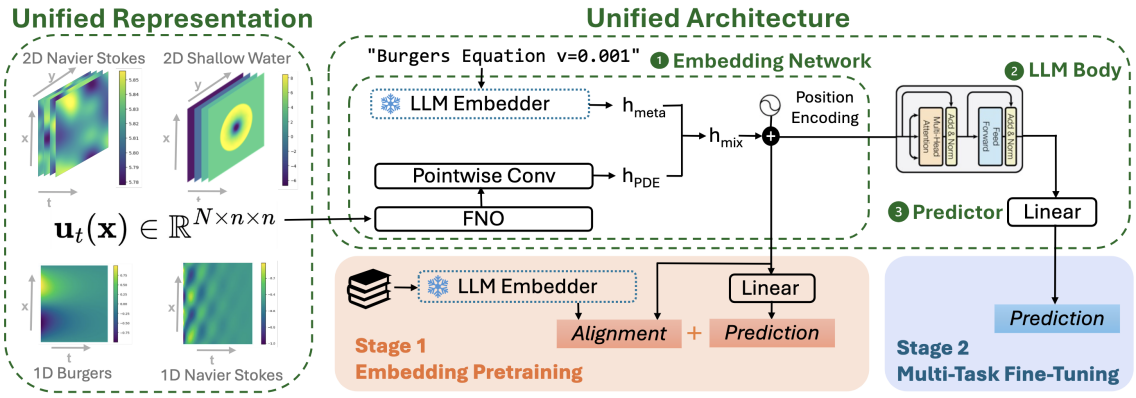


Figure 7.1: To adapt pretrained LLMs for PDE solving, UPS first transforms PDE of different dimensions, channels, and resolutions into a *unified representation* (left panel). Then, the data is processed with a *unified architecture* that integrates FNO layers, PDE metadata, and LLMs (right panel). The architecture is trained in two stages. In stage 1, we pretrain the embedding network using a joint loss that simultaneously optimizes (i) the distribution similarity between PDE features and text embeddings to align the modalities, and (ii) the prediction performance of extracted PDE features. In stage 2, we fine-tune the entire model on a dataset that combines multiple families of spatiotemporal PDEs with varying domain dimensions, initial/boundary conditions, and coefficients. UPS achieves competitive results with significantly better sample-efficiency than existing methods.

limited in generalization ability—all existing unified models focus on pretraining with 2D PDEs. Finally, as these models are developed using only PDE trajectories, they do not leverage meta information that could help distinguish between various PDE families, such as the name of the family and the coefficients.

We present Unified PDE Solvers (UPS), which learns unified neural operators for complex time-dependent PDEs with improved efficiency and generalization ability. Unlike existing efforts that train models from scratch, we propose a novel method to adapt pretrained Large Language Models (LLMs) to PDE solving. This is inspired by the line of work that repurposes LLMs for scientific domains like mathematics [Lewkowycz et al., 2022], computational biology [Shen et al., 2023, Vinod et al., 2023, Joachimiak et al., 2023], and chemistry [Bran et al., 2023, Shen et al., 2024]. These works not only show how LLMs utilize both text and non-text information to solve scientific problems and transfer effectively to unseen tasks, but also provide strong evidence that the general-purpose pretraining and inductive biases of LLMs could substantially reduce the sample complexity needed for adaptation.

Concretely, UPS adapts pretrained LLMs to time-evolution operators that map the current state of a PDE to its future state for general spatiotemporal PDEs (see Section 7.3 Equation 7.1 for definition) using two key designs (see also Figure 7.1 and Section 7.3):

1. We propose a **unified data representation scheme** to align PDEs with varying dimensions and physical quantities into the same feature space. Given the space and time discretization $\boldsymbol{u} = \{\bar{g}_t(\mathbf{x})\}_{t=0}^{T}$, where $\mathbf{x} \in \mathbb{R}^d$ is the spatial variable and $\bar{g}_t(\mathbf{x})$ is the state variable, UPS homogenizes $\bar{g}_t(\mathbf{x})$ from diverse PDEs into a shared "superspace" $\mathbb{R}^{N\times n^{d_{\max}}}$, where $d_{\max}$ is the maximum dimension of $\mathbf{x}$ among all PDEs considered, N is the superset of the physical quantities, and n is the resolution. This framework of embedding lower-dimensional PDEs in a higher dimension enables

UPS to model cross-dimensional PDEs simultaneously and distinguishes us from all existing unified operators, which do not consider low dimensional PDEs in 1D.

2. We employ a **unified network architecture** to predict $\bar{g}_{t+1}(\mathbf{x})$ based on $\bar{g}_t(\mathbf{x})$. To leverage pretrained LLMs, we design a three-way architecture that consists of (i) a FNO [Li et al., 2020a] based embedding network to convert PDE data into resolution-invariant, sequential features; (ii) an LLM body to process the PDE features and the text embeddings of the PDE metadata; and (iii) a prediction network to generate the final output. Inspired by previous cross-modality adaptation work [Shen et al., 2023], we employ a two-stage align-then-refine process for model training. However, we improve the align stage by using a joint loss that adds feature extraction on top of alignment to pretrain the embedding network. We improve the refine stage by fine-tuning on a collection of PDE tasks rather than a single task. Our enhanced workflow outperforms naive transfer and previous cross-modal approaches, reducing both the data and compute needed for training.

By design, UPS can handle diverse PDE families, data dimensions, channels, and resolutions. More crucially, by warm-starting with pretrained LLM weights and applying explicit alignment, UPS strikes a balance between effectiveness and efficiency—it achieves state-of-the-art performance across 9 datasets from PDEBench [Takamoto et al., 2022] (7 in-distribution, 2 out-of-distribution), using about 20,000 training trajectories , a single A6000, 60,000 train steps, and under 100 GPU hours. This means that we achieve better results than existing unified models using 4 times less data and 26 times less compute.

Beyond prediction accuracy, we confirm that UPS preserves key properties of neural operators, such as grid- and resolution-invariance. We also show that UPS is compatible with a variety of LLM backbones, including RoBERTa [Liu et al., 2019], T5 [Raffel et al., 2020b], and CLIP [Radford et al., 2021], and demonstrates better performance when scaled to larger backbones. We believe that the model-agnostic design of UPS offers a systematic approach to harnessing the advancements in LLMs for PDE solving, and it takes a further step towards building generalized foundation models for more complex physical systems efficiently. Code is available at https://github.com/sjunhongshen/UnifiedPDESolvers.

7.2 Related Work

Recent years has seen a variety of neural-network-based methods for approximating PDE solutions. Hybrid solvers [Hsieh et al., 2019, Bar-Sinai et al., 2019, Kochkov et al., 2021] apply classical solvers like finite element/volumn methods [LeVeque, 2007, Moukalled et al., 2016] to a low-resolution grid and use neural networks to predict the correction terms. Others directly approximate the PDE solutions with neural networks [Sirignano and DGM, 2017, Raissi et al., 2019, Khoo et al., 2021, Shen et al., 2022], using variational losses [Yu et al., 2018] or physical constraints defined by the PDE [Raissi et al., 2019, Bruna et al., 2024]. Being mostly equation-specific, these methods can solve one PDE at a time. The learned models do not apply to other PDEs in the same family, let alone other families.

A more general approach involves learning neural operators [Lu et al., 2019, Li et al., 2020a,b] which approximate an infinite-dimensional operator between two functional spaces. For time-dependent PDEs, a neural operator maps the current state of a PDE to the next state, with quantities like initial conditions provided as input. Neural operators can be implemented using any architecture. For example, Fourier

neural operator (FNO) [Li et al., 2020a] uses convolution-based integral kernels evaluated in the Fourier space. Other works also use transformer models [Cao, 2021, Li et al., 2022, Hao et al., 2023a] or U-Net [Lippe et al., 2023a]. Learning neural operators enables solving an entire family of PDE and they can easily adapt to new parameterizations of a PDE without fine-tuning. However, the learned operators cannot extend to different PDE families.

To facilitate operator transfer across PDE families, two recent works develop large pretrained models for multiple physical systems: Subramanian et al. [2023] train FNOs on steady-state linear PDEs with periodic boundary conditions; McCabe et al. [2023] design a new transformer architecture based on the axial attention [Ho et al., 2020] and train it using various 2D non-linear, time-dependent PDEs. While these methods show that a unified operator can outperform single-family operators, they are limited in two aspects. First, existing unified methods consider mainly 2D PDEs for pretraining and evaluation. In contrast, UPS leverages a unified representation scheme to tackle both 1D and 2D PDEs. This method can be also extended to any d-dimensional systems in theory. Second, existing methods pretrain large models from scratch and necessitate extensive GPU resources and pretraining data, which can be prohibitive to collect for high-dimensional complex PDEs. However, by adapting from pretrained LLMs and closing the modality gap between text and PDE efficiently, UPS achieves competitive results using 4x less data and 26x less compute.

Beyond the aforementioned works, DPOT [Hao et al., 2024b] was developed concurrently with our work and presents an auto-regressive denoising strategy for pretraining. While DPOT has shown better transferability to unseen PDE tasks than MPP, it shares the same limitations of focusing on 2D problems for pretraining and requiring large amount of data and compute (8 A800 GPUs for 500,000 steps).

A final work that is related to ours is ORCA [Shen et al., 2023], which proposes a general workflow for adapting pretrained language/vision transformers to non-text/vision inputs. While ORCA uses PDEBench in its evaluation, it is not tailored to PDE solving and requires adapting a separate model for every dataset. The resulting models are not grid- or resolution-invariant, which are key properties of neural operators and achieved by UPS. Moreover, by learning from multiple PDEs and sharing knowledge across families, UPS obtains significantly better empirical results than ORCA.

7.3 Methodology

Our goal is to train unified neural operators for spatiotemporal PDEs with varying domain dimensions, coefficients, initial and boundary conditions. These PDEs could model a range of quantities that evolve over time, from scalars (e.g., pressure, density) to vectors (e.g., velocity). To achieve this, we propose UPS, which consists of a unified way to represent the PDE and a LLM-based network to model them.

7.3.1 Unified Data Representation

We model PDEs that follow the general form:

$$\frac{\partial \bar{g}(t,\mathbf{x})}{\partial t} = L\left(\bar{g}(t,\mathbf{x}), \frac{\partial \bar{g}(t,\mathbf{x})}{\partial \mathbf{x}}, \frac{\partial \bar{g}(t,\mathbf{x})}{\partial \mathbf{x}^2}, \cdots\right) \qquad (7.1)$$
$$u(0,\mathbf{x}) = u_0(\mathbf{x}) \quad B(\bar{g}(t,\boldsymbol{y})) = h(y)$$

where $\mathbf{x} \in \Omega \subset \mathbb{R}^d$ is the spatial variable, $\bar{g} : [0,T] \times \Omega \to \mathbb{R}^{d_u}$ is a time-varying function defined over the domain Ω for finite time T. Here, L is a (possibly non-linear) operator which acts on $\bar{g}$ and multiple partial derivatives of $\bar{g}$ w.r.t the spatial variable $\mathbf{x}$. $\bar{g}_0(\mathbf{x}) : \Omega \to \mathbb{R}^{d_u}$ denotes PDE's initial condition, and the operator B defines the boundary condition where $\boldsymbol{y} \in \partial\Omega$ is a point on domain's boundary, and $h : \partial\Omega \to \mathbb{R}^{d_u}$ defines the given function over the boundary[2] PDE families in this form include Navier-Stokes equations, Reaction-Diffusion equations, Burgers equations, and many others that describe phenomena like fluid dynamics and heat flow over time. They also constitute most PDE benchmarks in the field of machine learning [Takamoto et al., 2022, Tu et al., 2022, Roberts et al., 2021].

Consider a set of S spatiotemporal PDEs $\{\bar{g}^s\}_{s=1}^S$. Here, each $\bar{g}^s = \{\bar{g}_t^s(\mathbf{x})\}_{t=1}^{T_s}$ is a solution to a PDE of the form defined in Equation 7.1 such that for all $t \in [T_s]$, we have $\bar{g}_t^s(\mathbf{x}) \in \mathbb{R}^{d_u^s}$ and $\mathbf{x} \in \Omega^s \subset \mathbb{R}^{d^s}$, where d^s is the dimension of the PDE s. For each $\bar{g}_t^s$, we assume that we have an n-point discretization of the functions $\{\bar{g}_t^s\}_{t=1}^{T_s}$ at points $W_n^s = \{\mathbf{x}_1^s, \mathbf{x}_2^s, \cdots, \mathbf{x}_{n^{d^s}}^s\}$, where each $\mathbf{x}_i^s \in \mathbb{R}^{d^s}$. That is, for each PDE $s \in S$ and $t \in T_s$, we have the realization of the function $\bar{g}_t^s$ on a grid with each dimension divided into n parts, thus giving rise to n^{d^s} points in the set. We assume that n is constant across PDE families. We note that this value n is a hyperparameter, and ideally should be the minimum number of points that work well for *all the PDEs* considered, for example, low-viscosity Navier-Stokes may require more discretization points compared to the high viscosity counterparts. Denote the set of N physical quantities considered for each PDE as $V = \{v_1, v_2, \cdots v_N\}$. Our goal is to learn an operator $\mathcal{G}_\theta$ which, for a given PDE s, predicts the state of the PDE at time $t+1$ based on its state at time $t \in [T_s]$, i.e., $\hat{\bar{g}}_{t+1}^s(\mathbf{x}) = \mathcal{G}_\theta(\bar{g}_t^s(\mathbf{x}))$. We thus need a unified representation for the inputs so a model can handle different quantities at once.

Unifying Dimension Let d^s denote the dimension of the PDE s and $d = \max_{s\in S} d^s$. We want to represent all datasets in $\mathbb{R}^d$. Thus, for PDEs with $d^s < d$, the final $d - d^s$ coordinates of $\mathbf{x}_i^s \in W_n^s$ are set to zero. In this work, we mainly consider PDEs defined over one- and two-dimensional domains, i.e., $d^s \in \{1,2\}\ \forall s \in S$. Hence, for PDEs with $d^s = 1$, the point $\mathbf{x} \in \Omega^s$ is represented with the 2D-coordinate $(x, 0)$. Note that our methodology to unify the total number of dimensions in the PDE is general and can be adapted to PDEs defined in higher-dimensional domains as well. In the following, we will denote $\bar{g}_t^s(\mathbf{x})$ as the value of the function $\bar{g}_t^s$ on all the points in W_n^s, unless stated otherwise.

Unifying Physical Quantities We consider a fixed set $V = \{v_1, v_2, \cdots v_N\}$ of N physical quantities and train our model on the quantities that belong to V for each PDE. The quantities we consider in this paper are velocity (in both x and y directions), pressure, and density, and they are the superset of all quantities for the PDE families we evaluate. This leads to $N = 4$. If a dataset does not use a particular quantity, the entire dimension corresponding to it is set to 0.

[2] Unless stated otherwise, we assume that the value of the function is 0 at the boundary, i.e, $h(y) = 0$ for all $y \in \partial\Omega$.

With the above procedure, we lift every PDE to a unified space so $\bar{g}^s \in \mathbb{R}^{T_s \times N \times n^d}\ \forall s \in S$. To obtain the datasets for forward prediction, we generate input-output pairs via autoregressive teacher-forcing: for each time step $t \in [T_s]$, we use $\bar{g}^s_t$ to predict $\hat{\bar{g}}^s_{t+1}$, yielding $T_s - 1$ pairs of data from a single trajectory. We append the coordinates of each $\mathbf{x}^s_i \in W^s_n$ to the input and maintain an output mask to mask out the zero-padded dimensions when computing the loss.

7.3.2 Unified Architecture

Transformer models have demonstrated success in various domains like natural language [Touvron et al., 2023a], vision [Dosovitskiy et al., 2021a], and audio processing [Lu et al., 2023]. In this work, we explore the potential of transformers for PDE solving. We break down the UPS architecture into 3 parts: an embedding network that transforms the unified representation into sequence features; the model body, consisting of the pretrained LLM layers; and a predictor that generates the prediction (Figure 7.1).

FNO Embedding Network The embedding network plays two roles. First, it projects the PDE $\bar{g}^s_t(\mathbf{x})$ into the LLM's sequential embedding space $\mathbb{R}^{l \times e}$, where l denotes the sequence length of the embedded features and e denotes the LLM's hidden dimension. Second, it should extract key features of the PDE input to enable subsequent transformer layers to make predictions. Therefore, we design a PDE-specific embedding network with FNO layers for feature extraction, a linear layer for dimensionality matching, and a concatenation operator for adding metadata (Figure 7.1).

We use FNO due to its strong empirical performance [Li et al., 2020a, Takamoto et al., 2022] and its ability to extract resolution-invariant features. As we consider maximum two-dimensional PDEs in this work, we use a series of 2D FNO layers with l channels to obtain PDE features in $\mathbb{R}^{l \times n^d}$. Then, to map the FNO output to the LLM's embedding dimension, we apply a pointwise convolution with input channel n^d, output channel e, kernel size 1, stride 1. This yields the desired sequential features $h_{\text{PDE}} \in \mathbb{R}^{l \times e}$.

Since UPS is intended to handle diverse data from various generating sources, we leverage the PDE's metadata in addition to the input dynamics. The motivation is that LLMs can use the textual information to better understand the context and characteristics of different PDEs. To implement this, we specify the metadata in the form "`[PDE family][coefficients]`" which is embedded into sequential features h_{meta} using the LLM's tokenizer and text embedding layer. We then concatenate the meta features and the PDE features to get $h_{\text{mix}} := [h_{\text{meta}}, h_{\text{PDE}}]$. Finally, we apply positional encoding and layer norm to h_{mix}. This will be the input to the subsequent transformer layers.

In Section 7.5.3, we perform various ablation studies on the embedding network. We investigate the effect different hyperparameters, such as the channel dimension l in FNO, and show that incorporating metadata improves both prediction performance and generalization ability of UPS.

Utilizing Pretrained LLMs The main body of a UPS model consists of pretrained transformer layers from an LLM. Thus, we pass h_{mix} to the pretrained transformer layers, which produce the hidden states $\hat{h} \in \mathbb{R}^{l \times e}$. Since there is no causal structure in the spatial dimensions of a PDE, we do not apply autoregressive masking to h_{mix} and allow the embedded features to attend to each other.

Our design provides flexibility for using different LLMs as the model body. We show experiment results with multiple LLMs in Section 7.5.3. While different LLMs have different performance, they are competitive with existing baselines. We also show that adapting from pretrained weights outperforms training the same architecture from scratch, so UPS is especially useful for low-data regimes.

Linear Predictor Finally, we define a prediction head to transform the hidden state of the LLM body $\hat{h}$ to the predicted next step of the input $\hat{\tilde{g}}^s_{t+1}(\mathbf{x}) \in \mathbb{R}^{N \times n^d}$ (we predict all the physical quantities in the set V). This is achieved by averaging over the sequence dimension of $\hat{h}$ to get shape $\mathbb{R}^e$, applying a linear layer to map it to $\mathbb{R}^{Nn^d}$, and reshaping the results to obtain the final prediction $\hat{\tilde{g}}^s_{t+1}(\mathbf{x})$. The linear predictor is shared for all PDEs.

7.4 Full Workflow and Training

We train UPS in two stages. In the first stage, we train the embedding network to align h_{mix} with the LLM's embedding space. This is because LLMs are trained for the text modality, which has distinct characteristics and features from physical processes like fluid dynamics and heat flow. Stage 1 reduces the modality gap to prevent the distortion of pretrained weights. Next, we fine-tune the entire model on a dataset of multiple families of spatiotemporal PDEs.

Stage 1: Embedding Pretraining Intuitively, there is a modality gap between text data used to train general-purpose LLMs and PDEs. Previous work has also shown that directly fine-tuning pretrained LLMs on non-text inputs can result in suboptimal performance [Lu et al., 2022]. To address this, Shen et al. [2023] introduced ORCA, which performs distribution matching before fine-tuning to enable cross-modal adaptation. That is, given a randomly initialized embedding network, we first pretrain it to minimize the distribution distance between the embedding network's output—in our case h_{mix}—and the text embeddings of a external reference NLP dataset, which we denote as h_{LM}. This process makes the cross-modal distribution resemble the text distribution that the LLM is pretrained on. Following the ORCA work, we use the CoNLL-2003 dataset [Sang and Meulder, 2003] as the reference dataset for alignment.

We propose several PDE-specific improvements to the alignment process. First, unlike ORCA which uses an optimal transport (OT) based metric for measuring the distribution distance, we use the maximum mean discrepancy (MMD) distance for UPS. This is because the OT-based metric requires discrete class labels to compute, making it unsuitable for PDEs. In contrast, MMD acts directly on the features h_{mix} and is more computationally efficient. Thus, we define

$$\mathcal{L}_{\text{align}} = \|\mu_{\mathcal{D}_{\text{h}_{\text{mix}}}} - \mu_{\mathcal{D}_{\text{h}_{\text{LM}}}}\|_{L_2} = \mathop{\mathbb{E}}_{\mathcal{D}_{\text{h}_{\text{mix}}}}[k(a, a')] - 2 \mathop{\mathbb{E}}_{\mathcal{D}_{\text{h}_{\text{mix}}}, \mathcal{D}_{\text{h}_{\text{LM}}}}[k(a, b)] - \mathop{\mathbb{E}}_{\mathcal{D}_{\text{h}_{\text{LM}}}}[k(b, b')] \tag{7.2}$$

where $k(a, a') = \exp(\|a - a'\|_2/2)$ denotes the Gaussian kernel; $\mathcal{D}_{\text{h}_{\text{mix}}}$ and $\mathcal{D}_{\text{h}_{\text{LM}}}$ denote the distributions of the PDE embeddings h_{mix} and the reference text embeddings h_{LM}.

Second, to improve the feature extraction ability of the embedding network in the context of our downstream task, we introduce a *task loss* for PDE forward prediction, i.e., the normalized root mean squared (nRMSE) loss between the prediction $\hat{\bar{g}}^s_{t+1}(\mathbf{x})$ and the ground truth $\bar{g}^s_{t+1}(\mathbf{x})$:

$$\mathcal{L}_{\text{task}} = \frac{1}{S}\sum_{s=0}^{S}\frac{1}{T_s}\sum_{t=0}^{T_s-1}\frac{\|\bar{g}^s_{t+1}(\mathbf{x}) - \hat{\bar{g}}^s_{t+1}(\mathbf{x})\|_2}{\|\bar{g}^s_{t+1}(\mathbf{x})\|_2} \tag{7.3}$$

Thus, the final objective for pretraining the embedding network is the joint loss $\mathcal{L}_{\text{emb}} = \mathcal{L}_{\text{align}} + \mathcal{L}_{\text{task}}$. We show in Section 7.5.3 that both objectives are essential to the overall performance of UPS.

Stage 2: Multi-Task Fine-Tuning In contrast to most existing neural PDE solvers, which train a separate model for each dataset, UPS is trained using one large dataset consisting of PDE data from multiple generating sources (all of S). Hence, after learning the embedding network, we fine-tune the entire model (the embedding network, the LLM body, and the linear predictor) using $\mathcal{L}_{\text{task}}$ defined in Equation 7.3. We evaluate the performance of UPS in Section 7.5.1 and find it outperforms existing single-dataset neural operators. We also show that UPS generalizes to unseen PDE families and coefficients (Section 7.5.2)—the zero-shot and few-shot adaptation performance is competitive with models specifically trained on the entire target dataset.

7.5 Experiments

Data We train and evaluate our method using PDEBench [Takamoto et al., 2022]. For training, we combine 7 datasets from different PDE families: Burgers Equation (1D), Advection (1D), Diffusion-Sportion (1D), Shallow-Water (2D), compressible Navier-Stokes (1D and 2D), and incompressible Navier-Stokes (2D). We explicitly hold out two families, 1D and 2D Diffusion-Reaction, to evaluate the generalization ability of UPS. The dataset details can be found in Appendix 14.0.1. We autoregressively generate the predictions and use the scale-independent normalized root mean squared error (nRMSE) as the evaluation metric, defined as follows:

$$\text{nRMSE} = \frac{1}{S_{\text{test}}}\sum_{s=1}^{S_{\text{test}}}\frac{\|\bar{g}^s(\mathbf{x}) - \hat{\bar{g}}^s(\mathbf{x})\|_2}{\|\bar{g}^s(\mathbf{x})\|_2} \tag{7.4}$$

We preprocess all the PDEs by normalizing each dataset along the channel dimension to ensure the scale of $\bar{g}^s_t(\mathbf{x})$ across datasets is similar[3].

Baselines We compare against two sets of baselines: (i) single-family models trained on individual PDE datasets, including the widely used U-Net [Ronneberger et al., 2015], FNO [Li et al., 2020b], the improved version FFNO [Tran et al., 2023], the transformer-based GNOT [Hao et al., 2023b] and OFormer [Li et al., 2023], as well as the cross-modal ORCA [Shen et al., 2023]; (ii) unified models trained on multiple datasets, including MPP [McCabe et al., 2023], DPOT [Hao et al., 2024b], and a unified FNO

[3] We standardize the data by subtracting the mean and dividing by the standard deviation to ensure training stability when using pretrained model weights. For loss computation, we apply an inverse transformation to the outputs to revert them to the original scale. Although data normalization may affect non-linear equations, it is essential to prevent loss explosion during fine-tuning. We leave exploring alternative methods to minimize distortion in non-linear dynamics as future work.

Table 7.1: nRMSEs (lower is better) for in-distribution PDEBench families, with baseline results taken from Takamoto et al. [2022], Shen et al. [2023], McCabe et al. [2023], Hao et al. [2024b]. '-' means that the result is not available. On all datasets, UPS with RoBERTa-Base (UPS-B) achieves the lowest nRMSEs among all smaller models and UPS with RoBERTa-Large (UPS-L) achieves the lowest nRMSEs among all large models. Numbers are bolded for each model size group.

	# Params (sorted)	Advection 1D	Burgers 1D	Diffusion-Sorption 1D	Navier-Stokes 1D	Shallow-Water 2D	Navier-Stokes 2D	Incomp Navier-Stokes 2D
Single-Family								
FNO	466K	0.011	0.042	0.0017	0.068	0.0044	0.36	0.0942
GNOT	1.8M	-	-	-	-	0.0068	0.0373	-
OFormer	1.9M	-	-	-	-	0.0072	0.0521	-
U-Net	7.7M	0.67	0.34	0.15	0.72	0.083	5.1	0.1903
ORCA	125M	0.0098	0.12	0.0016	0.062	0.006	0.3549	0.1529
Unified (Small)								
Unified FNO	466K	0.013	0.0501	0.0041	0.0101	0.0033	0.152	0.1064
MPP-B	116M	-	-	-	-	0.0024	0.0281	-
DPOT-M	122M	-	-	-	-	0.0029	0.0177	-
UPS-B **(Ours)**	149M	**0.0027**	**0.0399**	**0.0009**	**0.0056**	**0.0016**	**0.0153**	**0.0931**
Unified (Large)								
UPS-L **(Ours)**	387M	**0.0022**	**0.0373**	**0.0009**	**0.0045**	**0.0015**	**0.015**	**0.0924**
MPP-L	409M	-	-	-	-	0.0022	0.0208	-
DPOT-L	500M	-	-	-	-	0.0023	0.0158	-

trained using data transformed by our unified representation scheme. We note that MPP and DPOT focus on 2D PDEs and are pretrained on 2D Navier-Stokes, Shallow-Water, and Diffusion-Reaction from PDEBench. Subramanian et al. [2023] is not included as a baseline because its models are pretrained on different PDE families (e.g., Poisson's and Helmholtz equations) not present in PDEBench.

Implementation Details As noted in Section 7.3, UPS is compatible with any pretrained LLM. We present our main results using RoBERTa [Liu et al., 2019] and study other backbones in ablation studies (Table 7.4). We set the embedding FNO channel l to 32. Since the resolution of the 2D datasets in PDEBench is 128, we set the model resolution n to 128 and downsample datasets with higher resolutions. All of our experiments can be run on a single NVIDIA A6000 GPU. See Appendix 14.0.2 for training details. Due to computational constraints, results are based on a single run per network configuration.

7.5.1 Achieving State-of-the-Art Results on PDEBench with Compute Efficiency

We first study the *in-distribution* performance of UPS, i.e., we evaluate UPS on the test splits of the datasets that are used to train UPS, which consists of PDEs that share the same boundary conditions and coefficients with the training samples, but have different initial conditions. The results are shown in Table 7.1. In general, UPS with RoBERTa ranks first on all 7 datasets and improves the state-of-the-art by an order of magnitude on many 1D datasets. We analyze the results in more details below.

Compare with Single-Family Operators We outperform all single-family models like FNO and ORCA, which train a different model for every PDE family. This shows the benefits of learning a versatile neural operator rather than multiple specialized ones, and our model is capable of extracting universal rules when learning to model multiple PDE equations.

Compare with Unified Operators We note that existing unified models like MPP and DPOT do not pretrain or evaluate on 1D problems due to the limitation of their data representation. In contrast, UPS embeds low-dimensional PDEs in high-dimensional spaces and model all PDEs uniformly despite the dimensionality difference, achieving state-of-the-art results on all 1D datasets in PDEBench. As for the 2D problems considered, UPS with RoBERTa-Base outperforms MPP-B and DPOT-M, which have similar model sizes. UPS with RoBERTa-Large outperforms MPP-L and DPOT-L. We emphasize that UPS is trained on significantly fewer trajectories per PDE family (<5K) compared to the baselines. Besides, UPS can be run on a single A6000 for less time while maintaining good performance. This shows the data and compute benefits of adapting from pretrained LLMs.

Since MPP and DPOT focus on 2D problems and use a different set of pretraining datasets from ours, we train a 2D-only UPS on all 2D datasets in PDEBench to provide a more direct comparison. The results are shown in Appendix Table 14.4. Notably, while 2D UPS is still trained with less data (since the other methods use additional datasets like PDEArena [Gupta and Brandstetter, 2022]), our method ranks first on 4 of 8 datasets, outperforming DPOT on 5 of 8 datasets and outperforming MPP on 3 of 4 datasets.

Recall also that we train a 2D unified FNO using the datasets processed by our dimension unification scheme. The unified FNO does not always outperform single-family FNOs, especially on 1D tasks, possibly because the network is 2D, and the relatively simple architecture might not be able to extract shared information across PDE families and leverage it to improve performance. More crucially, UPS outperforms unified FNO on all datasets, showing the efficacy of our LLM-based architecture.

Scaling Up LLM Backbones To study the scaling behavior of our method, we adapt from both RoBERTa-Base (149M parameters) and RoBERTa-Large (387M parameters) and report the results in Table 7.1. The first observation is that the two versions of UPS all outperform baselines of similar sizes, achieving both effectiveness and efficiency. Besides, UPS-Large generally outperforms UPS-Base, which shows that scaling up the backbone has the potential to yield better results.

In addition to prediction errors in Table 7.1, we visualize some of the UPS outputs in Appendix 14.0.4 and show that it is indeed able to capture the key features and dynamics of different PDE families. For efficiency metrics, we report the training compute requirement, FLOPs, and inference time for UPS in Appendix 14.0.2. Compared to existing work, our method has lower FLOPs and shorter inference time. This shows that our method can be deployed in practical environments where both computational efficiency and speed are critical.

7.5.2 Generalizing to Unseen PDEs with Data Efficiency

In this section, we investigate the generalization (*out-of-distribution*) performance of UPS under three scenarios: (i) unseen PDE families, (ii) PDEs belonging to the training families but with different coefficients, and (iii) PDEs with higher-resolution grids. Unless otherwise specified, UPS-B is used.

Unseen PDE Families As mentioned earlier, we hold out the Diffusion-Reaction equations from developing UPS. We first directly evaluate UPS on these two tasks and report the zero-shot transfer performance. Then, we study few-shot transfer by randomly sampling $k \in \{10, 100\}$ trajectories from the training sets of the held-out tasks and use them to fine-tune UPS. Lastly, we report the fine-tuning results

Table 7.2: Zero- and few-shot transfer performance of UPS on unseen PDE families and coefficients. Our few-shot results are competitive with baselines trained with more data. UPS-B refers to UPS with RoBERTa-Base.

		Unseen PDE Families		Unseen Coefficients	
# Samples	Model	1D Diff-React	2D Diff-React	1D Burgers $\nu = 1.0$	2D Navier-Stokes $M = 1, \eta = \zeta = 0.1$
0	UPS-B	**0.0557**	**1.0593**	**0.0566**	**0.103**
	FNO	0.1839	1.2	1.0342	1.4302
	ORCA	0.1818	1.0812	1.6316	1.6399
10	UPS-B	**0.0107**	**0.3327**	**0.0134**	**0.0809**
	FNO	0.1698	0.8193	0.67	0.567
	ORCA	0.1004	0.5376	0.4829	0.1623
100	UPS-B	**0.0034**	0.2508	**0.0022**	**0.0543**
	FNO	0.0037	0.1869	0.0123	0.3962
	ORCA	0.0051	**0.1362**	0.027	0.0898
9K (Full)	UPS-B	**0.0003**	**0.041**	**0.0008**	**0.0191**
	FNO	0.0014	0.12	0.0031	0.098
	ORCA	0.0034	0.082	0.012	0.0287

Table 7.3: UPS with resolution 128 has an nRMSE of 0.0033 for Advection and 0.0931 for incompressible Navier-Stokes. We directly test UPS on higher resolutions.

Test Resolution	256	512	1024
Advection (nRMSE)	0.0057	0.0064	0.0068
Incomp Navier-Stokes (nRMSE)	0.119	0.126	-

with the full training dataset. The results are shown in Table 7.2. As the number of adaptation samples increases, the prediction error decreases. Notably, the 100-shot result of UPS on 1D datasets is better than the baselines trained on 9,000 data, i.e., we use 90x less data to match the performance of single-family operators. This makes UPS useful for low-resource PDE problems where data collection is costly and training models from scratch is challenging. On 2D Diffusion-Reaction, we are slightly worse than pre-trained MPP (0.0292) and DPOT (0.0106) since this dataset is considered as in-distrubution for MPP and DPOT.

Unseen Coefficients UPS also generalizes to PDEs in the same families as the training data but with different coefficients. We verify this by adapting UPS to Burgers Equation with $\nu = 1.0$ (the model is trained on $\nu = 0.001$) and compressible Navier-Stokes with $M = 1, \eta = \zeta = 0.1$ (the model is trained on $M = \eta = \zeta = 0.1$). The last two columns in Table 7.2 shows that while our zero-shot performance is already competitive, the performance after further adaptation outperforms most considered baselines.

Unseen Resolutions Zero-shot resolution refers to training the model on a lower resolution of the input data and evaluating them directly on a higher resolution. PDE solvers with this ability are better equipped to handle real-world scenarios where input data may vary in resolution due to practical constraints or sensor-based limitations. Recall that UPS is trained with n-point discretization W_n^s, and we set $n = 128$ because most 2D datasets in PDEBench has resolution 128. Now, we evaluate the performance of UPS for $n \in \{256, 512, 1024\}$, increasing the resolution of the input PDE. This is achieved by

Table 7.4: Results for the ablation studies. For each set of experiments, only the specified settings are different; all the other hyperparameters and training configurations are the same. Overall, the full UPS-Base workflow (first row for every study) most effectively leverages the pretrained knowledge of LLMs and obtains the best results.

Study No.	Settings	Advection 1D	Burgers 1D	Diff-Sorp 1D	Navier-Stokes 1D	Shallow-Water 2D	Navier-Stokes 2D	Incomp Navier-Stokes 2D
S1	Pretrained LLM	**0.0027**	**0.0399**	**0.0009**	**0.0056**	**0.0016**	**0.0153**	**0.0931**
	Training From Scratch	0.017	0.0546	0.0036	0.0159	0.0032	0.0461	0.1442
S2	Align and Task	**0.0027**	0.0399	**0.0009**	**0.0056**	**0.0016**	**0.0153**	**0.0931**
	Task Only	0.0048	**0.0389**	**0.0009**	0.0065	0.002	0.0184	0.1046
	Align Only	0.0039	0.043	0.0011	0.0063	0.0022	0.0187	0.1092
	No Embedding Pretraining	0.0049	0.0436	0.0019	0.0072	0.0024	0.0197	0.1079
S3	Concatenate Pretrained Text	**0.0027**	**0.0399**	**0.0009**	**0.0056**	**0.0016**	**0.0153**	**0.0931**
	Cross-Attention Pretrained Text	0.003	0.0420	**0.0009**	0.0065	0.0023	0.0189	0.1082
	Concatenate One-Hot	0.0029	0.0447	0.0011	0.006	0.0018	0.0198	0.095
	Concatenate Learned Embeddings	0.0041	0.0474	0.0014	0.0119	0.0036	0.0295	0.1103
	No Meta Information	0.0122	0.0453	0.001	0.0091	0.0026	0.0238	0.1171
S4	RoBERTa-Base	**0.0027**	0.0399	**0.0009**	**0.0056**	**0.0016**	0.0153	0.0931
	Flan-T5-Base	0.0094	0.0404	0.0076	0.0098	0.0028	0.037	0.1166
	CLIP-Base	0.0046	**0.0321**	0.0018	0.0063	0.0019	**0.0151**	**0.0905**
S5	$l = 32$	0.0027	**0.0399**	**0.0009**	**0.0056**	**0.0016**	**0.0153**	**0.0931**
	$l = 20$	**0.0024**	0.0423	**0.0009**	0.0068	0.0022	0.0157	0.1043
	$l = 8$	0.0032	0.0429	**0.0009**	0.0071	0.0024	0.0195	0.1064

downsampling the higher-resolution inputs to make them compatible with UPS and then upsampling the output prediction to the desired resolution. We do not fine-tune the model at all.

We report the resolution generalization performance for 1D Advection Equation and 2D incompressible Navier-Stokes in Table 7.3. Although the nRMSEs for both PDEs slightly increase compared to the nRMSE for the training resolution, they outperform all baselines in Table 7.1. Since the numbers are similar across columns, UPS generalizes to higher resolutions in a zero-shot manner.

7.5.3 Ablation Studies

We perform five sets of studies to ablate various design decisions in UPS. S1-S4 demonstrate why adapting from pretrained LLMs is beneficial, while S5 is related to the FNO embedding network.

S1: Pretrained LLMs vs. Training From Scratch Compared to existing single-family models like FNO, UPS uses a transformer-based architecture with more parameters and reuses the pretrained LLM weights for the model body. To show that our results are not solely attributed to the model size and that cross-modal adaptation is important, we evaluate the model's performance when we train a transformer model *from scratch* using the same PDE datasets without doing anything more complicated. As shown in Table 7.4, training from scratch results in much worse performance than UPS, showing the benefits of adapting a pretrained LLM.

S2: Cross-Modal Alignment We also test the importance of the two objectives used in stage 1, i.e., alignment loss with MMD, and task loss with nRMSE. We study three settings: (i) using only $\mathcal{L}_{\text{align}}$ for stage 1 as in Shen et al. [2023]; (ii) using only $\mathcal{L}_{\text{task}}$ for stage 1; and (iii) removing stage 1 from our workflow entirely. As shown in Table 7.4, while removing any objective reduces the performance across all datasets, removing the task loss has a more significant negative effect. Meanwhile, removing the entire

stage of embedding pretraining hurts prediction accuracy. This shows that simply fine-tuning the LLM without considering the modality gap or learning to extract PDE features is ineffective.

S3: Incorporating Text-Form Metadata UPS leverages the PDE's metadata by combining its text embeddings with the learned PDE embeddings. To study whether incorporating such metadata is helpful and identify an optimal approach, we compare our workflow with two alternatives: (i) we do not use metadata, so $h_{\text{mix}} := h_{\text{PDE}}$; (ii) we use metadata, but instead of concatenating features from two modalities, we apply a cross-attention mechanism: $h_{\text{mix}} := \text{softmax}(\frac{QK^T}{\sqrt{e}})V$, where $Q = W_Q h_{\text{PDE}}$, $K = W_K h_{\text{meta}}$, and $V = W_V h_{\text{meta}}$. To further investigate whether the pretrained text embeddings contribute beyond merely labeling the PDE type, we also study alternative embedding strategies that do not leverage language pretraining: (iii) we replace the pretrained text embeddings of the PDE meta information with one-hot encoded vectors representing each PDE type. This setting serve as a baseline to assess the impact of merely labeling the PDE types without any semantic understanding; (iv) we also test a setting where PDE types were embedded using a randomly initialized embedding layer that was trained from scratch along with the rest of the network, i.e., each new token represents a PDE family.

The results are shown in Table 7.4. UPS outperforms the non-metadata baseline, demonstrating the effect of incorporating metadata as a textual form of domain knowledge, which LLMs are able to understand. The results also suggest that feature concatenation is better than cross-modal attention, possibly because the latter is harder to optimize. Lastly, the setting utilizing pretrained text embeddings consistently outperforms the one-hot and embedding-from-scratch settings. In terms of learning dynamics, we also observe that using pretrained embeddings demonstrated faster convergence compared to the alternative strategies. This suggests that the pretrained semantic knowledge in the LLM indeed contributes to processing the PDE data, not just in labeling PDE types but might also in understanding the underlying physical phenomena. However, we leave studying the the exact mechanism of cross-modal transfer and the optimal combination of metadata and PDE data as a future direction.

S4: Other LLMs/VLMs To study whether UPS applies to other pretrained models, we further investigate Flan-T5 [Chung et al., 2022] and the vision language model CLIP [Radford et al., 2021]. In particular, for CLIP, we use its text model to encode the metadata and its vision model to process the PDE data. The results are reported in Table 7.4. Since these models are trained using the same datasets and optimizer configuration as RoBERTa, the results are not fully optimized. Nonetheless, their performance is competitive with existing baselines, and CLIP further outperforms RoBERTa on 3 tasks. This shows the compatibility of UPS with diverse pretrained backbones. A future direction is to study whether optimizing the training hyperparameters for each pretrained model—especially VLMs like CLIP that are trained for an additional vision modality—could improve downstream performance.

S5: FNO Embedder & Target Sequence Length As discussed in Section 7.3, the channel l of the FNO layers in the embedding network determines the sequence length of the PDE features that will be fed into the transformer layers. To study how this hyperparameter affects learning outcomes, we vary $l \in \{8, 20, 32\}$ and report the results in Table 7.4. In general, increasing l improves the size and capacity of the embedding network, as well as the expressivity of the PDE features. This leads to lower prediction error. However, using too many parameters for the embedding network may result in a trade-off between effectiveness and efficiency. For instance, we also experimented with $l = 64$ (Appendix 14.0.3) and

find that the longer sequence length leads to slight performance improvements but with much higher computational costs. Thus, we opt for $l = 32$ in our main experiments.

7.6 Conclusion and Future Work

In this paper, we present UPS, a method for adapting pretrained LLMs to unified time-evolution operators that predict the next state of a PDE from the current state. UPS applies to a diverse set of PDE families defined over one- and two-dimensional domains, with varying initial conditions, boundary conditions, coefficients, and resolutions. To train UPS, we develop a two-stage cross-modal adaptation protocol that first pretrains a FNO-based embedding network and aligns its hidden representations with the LLM's embedding space, and then fine-tunes the entire model on a dataset containing diverse families of PDEs. Since UPS is adapted from pretrained models, it requires fewer training samples and compute than previous approaches for training unified PDE solvers from scratch. We show that UPS achieves state-of-the-art performance across multiple datasets from PDEBench and is capable of zero- and few-shot transfer to different PDE families, coefficients, and resolutions.

We identify several future directions based on our work. First, we can validate our method on a broader range of PDEs with higher-order temporal derivatives or three-dimensional domains. Meanwhile, to seek a truly general foundation model for PDE, we aim to extend the types of tasks that UPS can solve. Currently, UPS is only applicable to forward prediction. It is important to study inverse problems of parameter estimation for different PDEs as well. For an impact statement, see Appendix **??**.

Part IV

Graph Neural Networks: Architectures and Theory

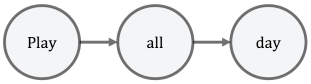
8 Chimera: State Space Models Beyond Sequences

Abstract: *Powerful deep learning methods based on Transformers are used to model diverse data modalities such as sequences, images, and graphs. These methods typically use off-the-shelf modules like self-attention, which are domain-agnostic and treat data as an unordered set of elements. To improve performance, researchers employ inductive biases—such as position embeddings in sequences and images, and random walks in graphs—to inject the domain structure, or* topology *into the model. However, these inductive biases are carefully engineered heuristics that must be designed for each modality, requiring significant research effort. In this work, we propose* Chimera*, a unified framework that mathematically generalizes state space models to incorporate the topological structure of data in a principled way. We demonstrate that our method achieves state-of-the-art performance across domains including language, vision, and graphs. Chimera outperforms BERT on the GLUE benchmark by 0.7 points, surpasses ViT by 2.6% on ImageNet-1k classification accuracy, and outperforms all baselines on the Long Range Graph Benchmark with a 12% improvement on PascalVOC. This validates Chimera's methodological improvement which allows it to directly capture the underlying topology, providing a strong inductive bias across modalities. Furthermore, being topologically aware enables our method to achieve a linear time complexity for sequences and images, in contrast to the quadratic complexity of attention.*

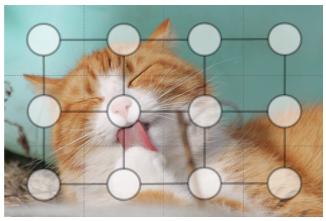
8.1 Introduction

Real-world data is heterogeneous, ranging from sequential language data to high-dimensional image data, and structured data of proteins and molecules. Despite this heterogeneity, many domains exhibit an inherent *topology* that encodes the neighborhood of each element (node) of the data. For instance, language and audio have a directed line graph topology, where each node (token) is arranged sequentially (Fig 8.1a). Similarly, images possess an undirected grid-graph topology, where each node (image patch) is connected to its immediate local neighbors in a grid (Fig 8.1b). Structured data like proteins have predefined nodes (atoms) and edges (bonds), which constitute their topology (Fig 8.1c).

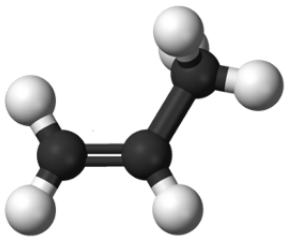
Typical approaches to model data build upon Transformers [Vaswani et al., 2017a] with self-attention at their core [Devlin et al., 2019, Dosovitskiy et al., 2021b, Rampášek et al., 2022]. However, since self-attention is permutation invariant, it treats data as an unordered set of elements and completely disregards the data's topology. To address this, significant research effort has focused on developing domain-specific heuristics, such as position embeddings [Su et al., 2023, Devlin et al., 2019], and random walks [Behrouz and Hashemi, 2024, Wang et al., 2024], to serve as the inductive bias for the underlying topology. However, developing these heuristics requires navigating a large search space for each domain. For instance,



(a) Language (Line Graph)



(b) Images (Grid Graph)



(c) Molecules (General Graph)

Figure 8.1: Real-world data exhibits inherent topology: (a) language follows a directed line graph, (b) images a grid graph, and (c) structured data like molecules have explicit graph topology.

RoPE embeddings [Su et al., 2023] work well in language [Touvron et al., 2023b]; in vision, absolute position embeddings are widely used [Dosovitskiy et al., 2021b, Heo et al., 2024]; Moreover, given the lack of systematic underpinnings, it is unclear how effectively they capture the underlying topology.

In this paper, we consider the following problem: *"Can we develop a principled method that captures the underlying data topology, and achieves state-of-the-art performance across domains?"*. We propose *Chimera*, a domain-agnostic framework built on recent State Space Models (SSMs)—Mamba-2 [Dao and Gu, 2024a], RetNet [Sun et al., 2023], Linear Attention (LA) [Katharopoulos et al., 2020]—that mathematically generalizes SSMs to *any topology* and achieves state-of-the-art performance across diverse domains including language, images, and graphs. These consistently superior results validate Chimera's methodological improvement which allows it to directly capture the underlying topology, providing a strong inductive bias across various modalities. This contrasts with existing approaches that instead apply attention or SSMs as a black box to "flattened data", supplemented by heuristics. Furthermore, being topologically aware allows Chimera to leverage the simpler topology of line and grid graphs to avoid "unnecessary computation", thus reducing its computational cost to linear in the number of nodes. This recovers the linear complexity of SSMs while maintaining strong performance.

To derive Chimera, we consider SSMs for causal language modeling and formally show that SSMs *inherently capture the underlying directed line graph topology* through their recurrence structure (Sec 8.3.2). For this, we leverage the Structured Masked Attention (SMA) representation [Dao and Gu, 2024a]: Multiple methods including Mamba-2, RetNet, LA are SSMs, and these SSMs are equivalent to the matrix $\mathbf{M} = \mathbf{L} \odot (\mathbf{Q}\mathbf{K}^T)$ acting on the input, where $\mathbf{Q}$ and $\mathbf{K}$ are the query and key matrices respectively, and $\mathbf{L}$ is a (data dependent) mask matrix. This mask matrix $\mathbf{L}$ is analogous to the causal masked attention matrix used in Transformers. We show that for SSMs, the mask matrix $\mathbf{L}$ can be represented as the resolvent of the adjacency matrix, $\mathbf{A}$, of a directed line graph, i.e., $\mathbf{L} = (\mathbf{I} - \mathbf{A})^{-1} = \sum \mathbf{A}^i$, where $\mathbf{I}$ is the identity matrix. Thus, $\mathbf{L}$ characterizes a specific SSM model and is also equivalent to the resolvent of the adjacency matrix, connecting SSMs and the underlying topology.

We extend this result to generalize SSMs to any topology. Specifically, we appropriately parameterize the adjacency matrix $\mathbf{A}$, and compute the SMA matrix $\mathbf{M} = \mathbf{L} \odot (\mathbf{Q}\mathbf{K}^T)$, where $\mathbf{L} = (\mathbf{I} - \mathbf{A})^{-1}$. Intuitively, $\mathbf{A}_{ij}$ captures the influence between neighbor i and j, and the resolvent then accumulates the influence between each pair of nodes through all possible paths between them, thus capturing the underlying topology. We present a detailed scheme to parameterize $\mathbf{A}$

Central to Chimera is the computation of the mask matrix whose naive implementation incurs cubic cost. To avoid this, Chimera leverages structure in the topology to significantly speed up this calculation. Specifically, for the class of directed acyclic graphs (DAGs), the resolvent operation can be computed in linear time. This is especially useful for topologies like undirected line graphs and grid graphs, which can be canonically decomposed into multiple DAGs: An undirected line graph decomposes into two directed line graphs (Fig 8.4), while a grid graph divides into four directed grid graphs (Fig 8.5). This allows us to implement Chimera in linear time—recovering the complexity of SSMs—while preserving the underlying topology. We further show that for general graphs, we can efficiently compute the finite sum approximation of the resolvent, capturing the global topological structure while achieving performance competitive with state-of-the-art baselines. Overall, we make the following contributions:

- We propose Chimera, a unified framework that generalizes SSMs to any data topology.
- We introduce a technique that leverages the underlying data topology using DAGs to improve the efficiency of Chimera, achieving linear time complexity for sequences and images.
- We validate that Chimera consistently achieves state-of-the-art results across diverse domains including language, images, and graphs: It outperforms BERT [Devlin et al., 2019] by a GLUE score [Wang et al., 2019] of 0.7, surpasses ViT [Dosovitskiy et al., 2021b] on ImageNet-1k [Deng et al., 2009] classification by 2.6%, and outperforms strong baselines on the Long Range Graph Benchmark (LRGB) [Dwivedi et al., 2022], notably increasing PascalVOC's F1 score by 12% .

8.2 Preliminaries

In this section, we introduce State Space Models (SSMs), which are recurrent models designed to process sequential data, such as language and audio. We first formulate SSMs in their recurrent form and then introduce the Structured Masked Attention (SMA) [Dao and Gu, 2024a] representation that interprets this recurrence as a matrix $\mathbf{M}$ acting on the input $\mathbf{X}$. In the subsequent section, we use the SMA representation to show that SSMs inherently operate on a directed line graph topology.

8.2.1 Overview of State Space Models

SSMs, such as Mamba-2 [Dao and Gu, 2024a], Linear Attention (LA) [Katharopoulos et al., 2020], RetNet [Sun et al., 2023], are recurrent sequence-to-sequence models that feature a linear hidden-state transition function. This linearity enables a hardware-efficient, vectorized implementation of SSMs, allowing them to scale effectively. Furthermore, this transition function is typically data-dependent which is known to improve model performance [Hwang et al., 2024].

Formally, let $\boldsymbol{X} \in \mathbb{R}^{T\times D}$ denote the input sequence of T tokens, where each token has D channels. Let the size of the hidden state be d. Let $\boldsymbol{Y} \in \mathbb{R}^{T\times D}$ be the output of the sequence-to-sequence model. Then, SSMs begin by computing the following matrices:

$$\boldsymbol{B} = f_B(\boldsymbol{X}) \in \mathbb{R}^{T\times d}, \quad \boldsymbol{C} = f_C(\boldsymbol{X}) \in \mathbb{R}^{T\times d}, \quad \boldsymbol{V} = f_V(\boldsymbol{X}) \in \mathbb{R}^{T\times d}, \tag{8.1}$$

where f_B, f_C, f_V are model specific data dependent functions. For instance, in Mamba-2 each of these functions is a composition of a linear projection of $\boldsymbol{X}$ along the channel dimension, followed by a short convolution layer along the sequence dimension and a Swish activation function [Ramachandran et al., 2017]. In Dao and Gu [2024a], it was shown that we can view the $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{V}$ matrices as analogs of the key, query, and value matrices in self-attention.

Let $\boldsymbol{v}^i \in \mathbb{R}^T$ denote the input corresponding to channel i (i.e., $\boldsymbol{v}^i = \boldsymbol{V}[:, i]$). For any time t, define $\boldsymbol{B}_t = \boldsymbol{B}[t, :]$, $\boldsymbol{C}_t = \boldsymbol{C}[t, :]$, $y_t^i = \boldsymbol{Y}[t, i]$ and $v_t^i = \boldsymbol{v}^i[t]$. Then, the model computes a recurrence, which is a function from $\boldsymbol{B}, \boldsymbol{C}, \Delta, \boldsymbol{V}$ to the output $\boldsymbol{Y}$, starting with the hidden state $\boldsymbol{h}_{-1}^i = \boldsymbol{0} \in \mathbb{R}^d$ as,

$$\boldsymbol{h}_t^i = a_t \boldsymbol{h}_{t-1}^i + b_t \boldsymbol{B}_t v_t^i, \tag{8.2}$$

$$y_t^i = \boldsymbol{C}_t^T \boldsymbol{h}_t^i, \tag{8.3}$$

where a_t, b_t are model-specific parameters that characterize the SSM. For instance, Linear Attention sets $a_t = b_t = 1$, RetNet chooses $a_t = \gamma, b_t = 1$ for some learnable parameter γ. In contrast, Mamba-2 sets a_t, b_t in a data-dependent manner to implicitly encode a gated memory mechanism known as *selectivity* or the *selection mechanism*. This mechanism allows the model to select and propagate important tokens across long sequences. Specifically, define,

$$\Delta = f_\Delta(\boldsymbol{X}) \in \mathbb{R}^T, \quad a_t = \exp(-\Delta_t) \in \mathbb{R}, \quad b_t = \Delta_t \in \mathbb{R}, \tag{8.4}$$

where Δ is the selectivity matrix, and f_Δ like f_B, f_C, f_V is a data-dependent function. The selection mechanism operates as follows: for an important token, Δ_t is large, and the model gives more weight to token t while reducing the contribution of the previous hidden state. Conversely, for an unimportant token, Δ_t is small and the model retains most of the past hidden state, with minimal contribution from token t. This allows Mamba-2 to retain important tokens through long recurrences.

8.2.2 SSM in the Structured Masked Attention Representation

In Dao and Gu [2024a], the authors introduced the Structured Masked Attention (SMA) representation, which computes the same function as the SSM recurrence (Eq. 8.3) described in the previous section but instead interprets the function computation as a matrix $\boldsymbol{M}$ acting on the value matrix $\boldsymbol{V}$.[1] They demonstrate that such an $\boldsymbol{M}$ is a function of $\boldsymbol{B}, \boldsymbol{C}, \Delta$ matrices (defined above) and can be expressed as $\boldsymbol{M} = \boldsymbol{L} \circ \boldsymbol{C}\boldsymbol{B}^T$, where $\boldsymbol{L}$ is a data-dependent mask matrix derived from the Δ matrix.

Formally, define $\bar{\boldsymbol{B}}_t = b_t \boldsymbol{B}_t$, and recall from Section 8.2.1 that $b_t = \Delta_t$, $a_t = \exp(-\Delta_t)$ for Mamba-2; $b_t = 1$, $a_t = \gamma$ for RetNet; and $b_t = 1$, $a_t = 1$ for Linear Attention. Then the output $\boldsymbol{Y}$ computed by the recurrence (Eq. 8.3) can be vectorized as,

$$\boldsymbol{Y} = \boldsymbol{M}\boldsymbol{V} = (\boldsymbol{L} \odot \boldsymbol{C}\bar{\boldsymbol{B}}^T)\boldsymbol{V}, \tag{8.5}$$

[1] Note that not all SSMs have an SMA representation, but you focus throughout this paper on ones that do (LA, RetNet, Mamba-2) and use we will use "SSMs" to refer specifically to this restricted class.

where the structured mask matrix $\boldsymbol{L}_{ij} = \mathbf{1}[i \geq j]\, \Pi_{j<k\leq i} a_k$, for all i, j,

$$\boldsymbol{L} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ a_1 & 1 & 0 & \cdots & 0 \\ a_1 a_2 & a_2 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_1 a_2 \cdots a_{T-1} & a_2 a_3 \cdots a_{T-1} & a_3 a_4 \cdots a_{T-1} & \cdots & 1 \end{bmatrix}. \tag{8.6}$$

The SMA representation is useful because, as we will demonstrate in Section 8.3, it neatly isolates the effect of the underlying topology within the recurrence computation into the mask matrix $\boldsymbol{L}$. This property allows us to generalize SSMs to arbitrary topologies by appropriately formulating the mask $\boldsymbol{L}$.

8.3 Chimera: Building Models for Any Topology

In this section, we introduce Chimera, a unified framework that generalizes SSMs to any arbitrary topology, enabling the development of performant models across diverse domains. Existing approaches such as Behrouz and Hashemi [2024], Devlin et al. [2019], Liu et al. [2021], treat attention and SSMs as black-box modules operating on fixed topologies such as sets or sequences and rely on heuristics to incorporate structural information. In contrast Chimera opens up this black box and mathematically adapts it to handle any topology.

To motivate Chimera, we first analyze the setting of SSMs applied to the causal language modeling task. We show that the recurrence in SSMs naturally operates on a directed line graph topology. To formalize this result, we first define the resolvent of a linear operator and interpret its action when this operator is a weighted adjacency matrix of a topology.

8.3.1 Resolvent Of An Adjacency Matrix Accumulates Influence

A graph topology consists of a set of nodes $\mathcal{V}$ that represent data elements, and edges $\mathcal{E}$ that encode the underlying topological structure. We conceptualize the associated adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ as *capturing the influence between neighboring nodes*. Specifically, $\boldsymbol{A}_{ij}$ is the influence that node j has on node i, for each edge (i, j). The natural desideratum then is to extend the notion of influence to all node pairs by incorporating the graph's structure, accounting for all possible paths between them. To model this cumulative influence, we introduce the concept of the resolvent of a linear operator

Definition 30 (Resolvent of a Linear Operator [Reed and Simon, 1980])**.** *Let $\boldsymbol{A} \in \mathbb{R}^{T\times T}$ be a linear operator, $\boldsymbol{I}$ the identity operator, and λ a complex number. Then, the resolvent operator is defined as:*

$$R(\lambda, \boldsymbol{A}) = (\lambda \boldsymbol{I} - \boldsymbol{A})^{-1}, \tag{8.7}$$

which exists for all complex numbers λ that are not in the spectrum of $\boldsymbol{A}$, i.e., $\lambda \notin \sigma(\boldsymbol{A})$. In this work, we set $\lambda = 1$ to remain in the field of real numbers, and this is done without loss of generality, as any choice of λ is equivalent upto scaling of the model.

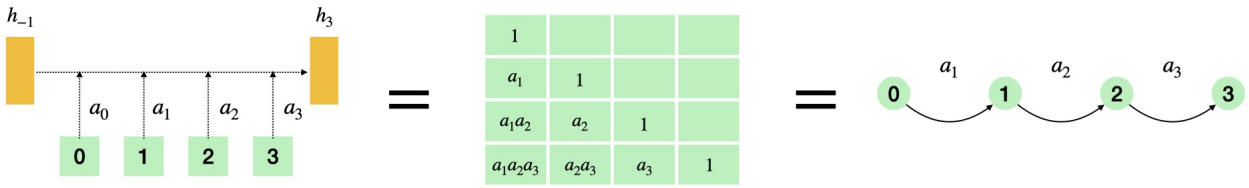


Figure 8.2: SSMs inherently operate on a directed line graph: SSMs modeling a sequence of tokens (left), the structured mask matrix (center), Chimera on a directed line graph (right)

We now demonstrate how the resolvent operator captures the influence between any two nodes in the graph. Observe that the resolvent operation can be expanded using the Liouville-Neumann series if the operator norm of the adjacency matrix, $\|\boldsymbol{A}\|$, is less than 1,

$$R(1, \boldsymbol{A}) = (\boldsymbol{I} - \boldsymbol{A})^{-1} = \sum_{k=0}^{\infty} \boldsymbol{A}^k. \tag{8.8}$$

Intuitively, each term $\boldsymbol{A}^k$ in this expansion represents the influence between any two nodes i and j through all paths of length exactly k connecting them. This is formalized in Proposition 4.

Proposition 4 ($\boldsymbol{A}^k$ accumulate influence through paths of length k)**.** *Given the weighted adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{T\times T}$ of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}| = T$, the $(i, j)^{th}$ entry of $\boldsymbol{A}^k$ is:*

$$(\boldsymbol{A}^k)_{ij} = \sum_{p_1, p_2, \ldots, p_{k-1}} \boldsymbol{A}_{ip_1} \boldsymbol{A}_{p_1p_2} \cdots \boldsymbol{A}_{p_{k-1}j},$$

where $(p_1, \ldots, p_{k-1})$ is an ordered sequence of vertices forming a path of length k from node i to j.

Therefore, the series $(\boldsymbol{I} - \boldsymbol{A})^{-1}$ (Eq. 8.8) sums up the influence of node i on node j over all possible paths and path lengths. Additionally, we also note that Eq. 8.8 provides a sufficient condition for the existence of the resolvent: the series converges when the operator norm of $\boldsymbol{A}$ is less than one.

8.3.2 SSMs operate on a Directed Line Graph

We now show that SSMs naturally operate on a directed line graph. Specifically, let $\mathcal{V}$ be the set of tokens, and $\mathcal{E}$ be the edges connecting token t to the next token $t + 1$. The weighted adjacency matrix is defined as $\boldsymbol{A}_{s,t} = \mathbf{1}_{[t=s+1]} a_t$, where a_t is the method-specific parameter described in Section 8.2.2.

We recall from Section 8.2.2 that SSMs can be represented as the SMA matrix $\mathbf{M} = \mathbf{L} \odot (\mathbf{C}\mathbf{B}^T)$. We make the key observation that $\boldsymbol{L}$ *is precisely the resolvent of* $\boldsymbol{A}$, that is $\boldsymbol{L} = (\boldsymbol{I} - \boldsymbol{A})^{-1}$. This ties SSMs to the directed line graph topology, with the mask matrix encoding the topology (Fig 8.2).

Proposition 5. *Under the notation established in Section 8.2, consider a weighted directed graph $\mathcal{G}$ with nodes $\mathcal{V} = \{0, \cdots, T-1\}$, edges $\mathcal{E} = \{(i-1, i) | i \in \mathcal{V}, i > 0\}$, and the edge weights $\mathcal{W} = \{w_{i-1 \to i} = a_i | i \in \mathcal{V}, i > 0\}$. Let $\boldsymbol{A}$ be the weighted adjacency matrix of incoming edges,*

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ a_1 & 0 & 0 & \cdots & 0 \\ 0 & a_2 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 \cdots 0 & 0 \cdots 0 & 0 & a_{T-1} & 0 \end{bmatrix}, \tag{8.9}$$

then $\boldsymbol{L} = \sum_{i=0}^{\infty} \boldsymbol{A}^i = (\boldsymbol{I} - \boldsymbol{A})^{-1}$, and consequently, $\boldsymbol{y} = ((\boldsymbol{I} - \boldsymbol{A})^{-1} \odot \boldsymbol{C}\bar{\boldsymbol{B}}^T)\boldsymbol{V}$.

We can interpret this result intuitively: in a directed line graph, there is exactly one path between tokens i,j with $i < j$, and the corresponding entry in $\boldsymbol{L}$, $\boldsymbol{L}_{ij} = \prod_{i \geq k > j} a_k$, reflects the cumulative influence of the intervening tokens along this path. Furthermore, $\boldsymbol{L}_{ij} = 0$ for $i < j$ restricts influence in the forward direction, ensuring that the model remains causal. This shows that SSMs inherently operate on a directed line graph with the $\boldsymbol{L}$ matrix encoding the topology.

8.3.3 Generalizing SSMs to Arbitrary Topologies

We now build on Proposition 5 to generalize SSMs to arbitrary topologies. Specifically, we compute the resolvent of an "appropriately parameterized" adjacency matrix, $\boldsymbol{A}$, and model the output in the SMA representation as $((\boldsymbol{I} - \boldsymbol{A})^{-1} \odot (\boldsymbol{C}\bar{\boldsymbol{B}}^T))\boldsymbol{V}$. In this section, we focus on the parameterization of $\boldsymbol{A}$ for arbitrary topologies and ensuring the numerical stability of the method, particularly in cases of non-invertibility or poor conditioning of $\boldsymbol{I} - \boldsymbol{A}$.

Formally, consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}| = T$ nodes, where each node has D channels. Let d denote the generalized hidden state size. For each node, we compute the following matrices,

$$\boldsymbol{B} = f_B(\boldsymbol{X}) \in \mathbb{R}^{T \times d}, \ \boldsymbol{C} = f_C(\boldsymbol{X}) \in \mathbb{R}^{T \times d}, \ \boldsymbol{V} = f_V(\boldsymbol{X}) \in \mathbb{R}^{T \times d}, \ \Delta = f_\Delta(\boldsymbol{X}) \in \mathbb{R}^T, \tag{8.10}$$

where the functions f_B, f_C, $f_V(\boldsymbol{X})$, f_Δ are linear projections applied to the input, followed by a local graph convolution over neighboring nodes and a Swish activation as chosen in Mamba-2. Our parameterization is inspired by Mamba-2 [Dao and Gu, 2024a]—one of the latest iterations of SSMs—as it features selectivity, which allows it to effectively model long-range dependencies. However, we note that our approach can generalize any SSM with an SMA representation.

We parameterize the $\boldsymbol{A}$ matrix for each edge $(i, j) \in \mathcal{E}$ as,

$$\boldsymbol{A}_{ij} = \frac{\exp(-\Delta_i) + \exp(-\Delta_j)}{2}, \tag{8.11}$$

to incorporate context from both ends of the edge (i, j). To add directionality to the edge representation and to further increase the representational power of our model, we can also maintain two (different) Δ's such that $\boldsymbol{A}_{ij} = (\exp(-\Delta_i^{(1)}) + \exp(-\Delta_j^{(2)}))/2)$.

Note that the matrix $\boldsymbol{I} - \boldsymbol{A}$ may be either non-invertible or poorly conditioned, which could hinder the stable training of the model. To address this, we introduce a data-dependent normalization parameter $\Psi = f_\Psi(\boldsymbol{X}) \in \mathbb{R}^T$, computed similarly to Δ, and perform a row-wise normalization of the adjacency matrix using Ψ. Specifically, for each row $i \in [T]$, we apply:

$$\boldsymbol{A}[i,:] = \frac{\gamma \boldsymbol{A}[i,:]}{\mathbf{1}^T \boldsymbol{A}[i,:] + \exp(-\Psi_i)},$$

where γ is a scaling hyperparameter. In the following proposition, we show that this normalization guarantees the convergence of the Neumann series for the adjacency matrix $\boldsymbol{A}$.

Proposition 6. *Under Gaussian initialization, the row-wise normalization strategy ensures that* $\|\boldsymbol{A}\| < 1$ *and* $\|(\boldsymbol{I} - \boldsymbol{A})^{-1}\|$ *is bounded with probability greater than* $1 - \Phi(\frac{-1}{\gamma})$.

We provide the proof for this proposition in Appendix 15.1.1. Finally, we compute the resolvent matrix $\boldsymbol{L} = (\boldsymbol{I} - \boldsymbol{A})^{-1}$ and the output $\boldsymbol{y}$ as $(\boldsymbol{L} \odot \boldsymbol{C}\bar{\boldsymbol{B}}^T)\boldsymbol{V}$.

8.4 Chimera with improved efficiency

While Chimera works with arbitrary graph topologies, directly computing the resolvent incurs a cubic cost in the number of nodes. However, we show that we can significantly reduce this computational cost when the underlying topology is more structured. Specifically, we consider the class of directed acyclic graphs (DAGs), a generalization of directed line graphs, and show that the resolvent can be computed in linear time, matching the complexity of SSMs like Mamba-2.

8.4.1 Chimera on DAGs

We tailor Chimera to DAGs with a specialized normalization scheme and an algorithm to compute the output in linear time. Our choice of DAGs is motivated by the fact that topologies such as undirected line and grid graphs can be canonically decomposed into DAGs: a line graph divides into two directed line graphs (Fig 8.4) and a grid graph divides into four directed grid graphs (Fig 8.5). This decomposition enables Chimera to operate efficiently with a linear complexity while preserving topology.

Formally, consider a DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}| = T$ nodes, each with D channels and a hidden state size of d. For any node i, let $p(i)$ be the set of its parents. Let $\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{V}, \Delta$ be the input projections as defined in Section 8.3. We define the adjacency matrix $\boldsymbol{A}$ as $\boldsymbol{A}_{ij} = \exp(-\Delta_i[j])$ for each $(i,j) \in \mathcal{E}$, and set $\bar{\boldsymbol{B}}_i = \Delta_i \boldsymbol{B}_i$ for each node i. We first show that the resolvent $(\boldsymbol{I} - \boldsymbol{A})^{-1}$ exists.

Proposition 7. *For a DAG,* $\boldsymbol{A}$ *is nilpotent, that is* $\boldsymbol{A}^T = \mathbf{0}$. *Therefore, the inverse* $(\boldsymbol{I} - \boldsymbol{A})^{-1}$ *exists and is given by the finite sum:*

$$\boldsymbol{L} = (\boldsymbol{I} - \boldsymbol{A})^{-1} = \sum_{t=0}^{T-1} \boldsymbol{A}^t. \tag{8.12}$$

As in previous sections, we compute the output of the model as $\boldsymbol{y} = (\boldsymbol{L} \odot (\boldsymbol{C}\bar{\boldsymbol{B}}^T))\boldsymbol{V}$. Furthermore, this method admits an equivalent recurrent view (Prop. 8).

Proposition 8. *Our method computes the following recurrence on each channel $\boldsymbol{v}$ of $\boldsymbol{V}$:*

$$\boldsymbol{h}_i = \sum_{j \in p(i)} \boldsymbol{A}_{ij}\boldsymbol{h}_j - \bar{\boldsymbol{B}}_i v_i, \qquad y_i = \boldsymbol{C}_i^T \boldsymbol{h}_i, \tag{8.13}$$

where $\boldsymbol{h}_l = \boldsymbol{0}$ for all leaf nodes l.

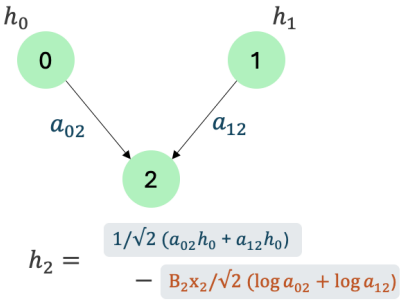


Figure 8.3: Recurrence on DAGs

Observe that while the resolvent always exists, its entries can become exceedingly large which can cause numerical instabilities. Recall from Section 8.3.1 that each $\boldsymbol{L}_{ij}$ represents the cumulative sum of all paths from node j to i, and in the worst case, the number of such paths grows exponentially with distance. To address this, we introduce a normalization scheme that is built directly into the recurrence:

Proposition 9. *The normalized method computes the following recurrence:*

$$\boldsymbol{h}_i = \frac{1}{\sqrt{|p(i)|}} \sum_{j \in p(i)} (\boldsymbol{A}_{ij}\boldsymbol{h}_j - \ln(\boldsymbol{A}_{ij})\boldsymbol{B}_i v_i), \tag{8.14}$$

$$y_i = \boldsymbol{C}_i^T \boldsymbol{h}_i. \tag{8.15}$$

This normalization ensures that $Var(\boldsymbol{C}_i^T \boldsymbol{h}_i) \leq 1$ under the assumption that the vectors $\{\boldsymbol{B}_i v_i, \boldsymbol{C}_i\}_i$ are i.i.d. Gaussians, that is $\boldsymbol{B}_i v_i, \boldsymbol{C}_i \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_d)$.

The proof follows by induction on the time step t, where at each time step, we ensure that the output variance is bounded by 1, $\text{Var}(\boldsymbol{C}_i^T \boldsymbol{h}_i) \leq 1$, which guarantees that the output remains a well-behaved random variable. We provide the detailed proof in Appendix 15.1.2. To incorporate this normalization in the SMA representation, we define,

$$\bar{\boldsymbol{A}} = \frac{1}{\sqrt{|p(i)|}}\boldsymbol{A}, \quad \bar{\boldsymbol{B}} = \frac{\ln(\boldsymbol{A}_{ij})}{\sqrt{|p(i)|}}\boldsymbol{B}, \quad \boldsymbol{L} = (\boldsymbol{I} - \bar{\boldsymbol{A}})^{-1}, \tag{8.16}$$

and compute the output $\boldsymbol{y} = (\boldsymbol{L} \odot (\boldsymbol{C}\bar{\boldsymbol{B}}^T))\boldsymbol{V}$.

Chimera is efficient on DAGs

Finally, we highlight that DAGs are a particularly important case of Chimera because of additional efficiency benefits, both theoretically and through optimized implementations.

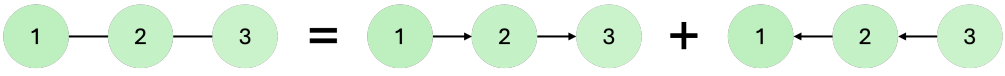


Figure 8.4: The undirected line graph structure (Left). The canonical DAG decomposition (Right)

Linear-time Complexity The intuition for the linear complexity is that the resolvent operation for DAGs is *finite* because of the lack of cycles. From the adjacency matrix perspective, $\boldsymbol{A}$ is nilpotent, i.e. $\boldsymbol{A}^k = 0$, where k is the diameter of the graph (Prop 7). Since Chimera can be equivalently viewed as a recurrence on the DAG, the resolvent operation converges after one pass through the graph in the topological order which takes linear time.

Proposition 10. *The Chimera structured mask matrix L can be computed in $O(|\mathcal{V} + |\mathcal{E}|)$ complexity where $|\mathcal{V}|, |\mathcal{E}|$ is the number of vertices and edges of the graph, respectively.*

The proof is provided in Appendix 15.1.3. We note that the linear-time complexity of Mamba can be seen as a special case of Proposition 10 specialized to the directed line graph, where both $|\mathcal{V}|$ and $|\mathcal{E}|$ is equal to the sequence length.

Improving Efficiency Through Matrix Multiplications Finally, we note that on modern hardware accelerators such as GPUs and TPUs, various computational algorithms can have different efficiency tradeoffs. For example, on directed line graphs, the naive computation of SSMs and RNNs as a recurrence is not parallelizable and is inefficient in practice [Gu and Dao, 2023a]. In the case of DAGs, we present a technique to reduce both the forward and backward pass for Chimera to leverage only matrix multiplications which are heavily optimized on modern accelerators.

Theorem 10. *In case of Chimera on DAGs, the forward pass can be computed with $O(\log(dia(\mathcal{G})))$ matrix multiplications where $dia(\mathcal{G})$ is the diameter of the graph (i.e. length of the longest path), and the backward pass can be computed with $O(1)$ matrix multiplications.*

Backward pass. The local update rule of backpropagation requires applying the chain rule through the matrix inverse operation, in particular, using the following identity applied to $\boldsymbol{Y} = (\boldsymbol{I} - \boldsymbol{A})$,

$$\frac{\partial \boldsymbol{Y}^{-1}}{\partial \theta} = -\boldsymbol{Y}^{-1} \frac{\partial \boldsymbol{Y}}{\partial \theta} \boldsymbol{Y}^{-1} \tag{8.17}$$

Because $\boldsymbol{Y}^{-1}$ is already computed in the forward pass, it can be cached, and then the marginal cost of the local backpropagation is simply two extra matrix multiplications.

Forward pass. To compute $\boldsymbol{L} = (\boldsymbol{I} - \boldsymbol{A})^{-1}$ more efficiently for DAGs, we leverage the equivalence of Neumann series to the series $\boldsymbol{L} = \boldsymbol{I} + \boldsymbol{A} + \boldsymbol{A}^2 + \cdots$, which comes to a finite sum for DAGs due to the nilpotence of $\boldsymbol{A}$ matrix. We compute this sum more efficiently using the "squaring trick" as,

$$(\boldsymbol{I} - \boldsymbol{A})^{-1} = (\boldsymbol{I} + \boldsymbol{A})(\boldsymbol{I} + \boldsymbol{A}^2)(\boldsymbol{I} + \boldsymbol{A}^4) \cdots (\boldsymbol{I} + \boldsymbol{A}^k), \tag{8.18}$$

where k is the smallest power of 2 larger than the graph diameter $\text{dia}(\mathcal{G})$. This can be computed using $O(\log(\text{dia}(\mathcal{G})))$ matrix multiplications to compute the powers of $\boldsymbol{A}$ for powers-of-two exponents, and then $O(\log(\text{dia}(\mathcal{G})))$ matrix multiplications to multiply together the right-hand side.

Approximate Chimera for General Topology

While DAGs allow for efficient computation in structured domains like images and language, directly computing the resolvent $\boldsymbol{L}$ for general graph topology remains computationally expensive. To address this, we use a finite-sum relaxation of the resolvent operator and truncate its corresponding Neumann series sum (Eq. 8.8) at some maximum power $k \in \mathbb{N} > 0$. Specifically, let $\boldsymbol{A}$ be the adjacency matrix of the graph topology defined in Section 8.3.3, then,

$$\boldsymbol{L} = \sum_{i=0}^{\infty} \boldsymbol{A}^i \approx \hat{\boldsymbol{L}} = \sum_{i=0}^{k} \boldsymbol{A}^i. \tag{8.19}$$

We choose $k = \text{diam}(\mathcal{G})$, the diameter of the graph, to ensure that $\hat{\boldsymbol{L}}$ has access to the global structure of the graph, that is, it includes contributions from every edge and node in the graph.

Proposition 11. *If $k \geq dia(\mathcal{G})$, then for any pair of nodes (i, j), if $\boldsymbol{L}_{ij} > 0$ in the original method, then $\hat{\boldsymbol{L}}_{ij} > 0$ in the finite-sum relaxation.*

As in Section 8.4.1, we can compute this approximation efficiently using the squaring trick:

$$\hat{\boldsymbol{L}} = (\boldsymbol{I} + \boldsymbol{A})(\boldsymbol{I} + \boldsymbol{A}^2)(\boldsymbol{I} + \boldsymbol{A}^4) \cdots (\boldsymbol{I} + \boldsymbol{A}^p), \tag{8.20}$$

where p is the smallest power of 2 larger than or equal to the graph diameter $\text{dia}(\mathcal{G})$. This reduces the computational cost of the method to $O(\log(\text{dia}(\mathcal{G})))$ matrix multiplications.

8.5 Experiments

In this section, we will demonstrate that *directly incorporating topology is a powerful inductive bias for diverse domains* such as language, images and graphs, eliminating the need for domain-specific heuristics. Chimera consistently achieves state-of-the-art performance in these domains. On language, it outperforms BERT on the GLUE benchmark [Wang, 2018] by a GLUE score of 0.7. On images, it surpasses ViT models on the ImageNet-1k classification [Deng et al., 2009] task by 2.6%. On general graphs, Chimera outperforms strong baselines on the Long Range Graph Benchmark [Dwivedi et al., 2021] which highlights our method's ability to model long range interactions on graphs. Notably, our method improves upon PascalVOC dataset's F1 score by over 12%.

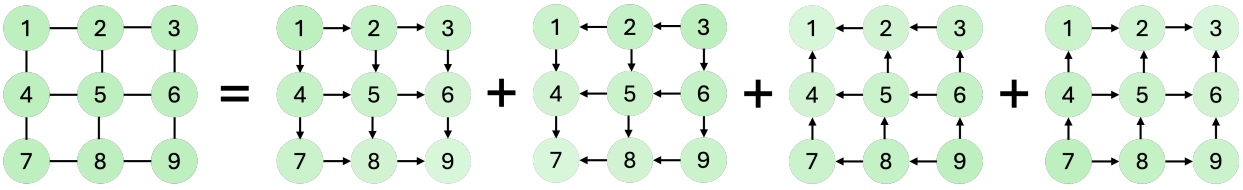


Figure 8.5: Grid graph (left). The canonical 2D-DAG decomposition of the grid graph (right). These graphs are sufficient to capture the influence between all pairs of nodes in the undirected grid graph.

8.5.1 Masked Language Modeling

We evaluate Chimera on bidirectional language modeling, which has a line graph topology (Fig. 8.4). We test two Chimera variants: the general method[2] (Sec. 8.3) applied to an undirected line graph, and the DAG method (Sec. 8.4.1), applied to the canonical DAG decomposition of undirected line graphs into two directed line graphs and summing the resolvents of both DAGs (Fig. 8.4). Both methods are trained on the Masked Language Modeling (MLM) [Devlin et al., 2019] task on the C4 dataset [Raffel et al., 2020a] for 70k steps, following the recipe used in M2 [Fu et al., 2023]. The models are then fine-tuned on the GLUE benchmark. We refer the reader to Appendix 15.3 for details.

Table 8.1: Comparing Chimera on the undirected line graph (UG), and on DAG decomposed directed line graphs (DAG) with other state-of-the-art models including M2 [Fu et al., 2023], MLP-Mixer [Tolstikhin et al., 2021], FNet [Lee-Thorp et al., 2022], BERT [Devlin et al., 2019] on GLUE benchmark

Method	#Params	Pretrain		GLUE Tasks								GLUE Avg
		$\mathcal{L}_{ce}$	Acc (%)	MNLI	QNLI	QQP	RTE	SST2	MRPC	COLA	STS	
BERT-Base	110M	1.59	67.3	84.1	**89.8**	**91.2**	77.2	91.2	87.5	54.6	**88.9**	83.2
MLP-Mixer	112M	1.77	63.5	77.2	82.4	87.6	67.3	90.5	86.5	43.0	85.2	77.5
FNet	112M	1.94	61.3	74.9	82.1	85.7	63.6	87.6	86.4	42.7	83.1	75.8
M2	116M	1.65	65.9	80.5	86.0	87.0	69.3	92.3	89.2	56.0	86.9	80.9
Chimera (UG)	110M	1.49	68.5	83.63	88.98	89.32	73	93.67	89.4	56.95	88.82	82.97
Chimera (DAG)	110M	1.46	**68.9**	**84.11**	89.78	89.77	**77.98**	**93.69**	**90.36**	**57.08**	88.68	**83.93**

From Table 8.1, observe that while BERT outperforms other linear baselines such as M2, MLP-Mixer, FNet it does so with an additional quadratic cost. In contrast, Chimera achieves the best of both worlds, incurring a linear time complexity while achieving state-of-the-art performance. This capability arises from two key factors: first, our parameterization of the adjacency matrix allows the model to effectively modulate the influence between tokens in the sequence, leading to strong performance. Second, the structured nature of the adjacency matrix enables a fast, linear-time resolvent operation, improving the method's computational efficiency. Additionally, note that our undirected graph (UG) variant performs competitively with BERT while surpassing other recent baselines with a linear time complexity.

8.5.2 ImageNet-1k Classification

We evaluate Chimera on the ImageNet-1k [Deng et al., 2009] classification task that has a grid graph topology. We compare Chimera applied to the 2D-DAG decomposition (Figure 8.5) topology against state-of-the-art ViT based models, specifically we use ViT-B which has 88M parameters. We also compare against other latest linear time baselines like Hyena [Poli et al., 2023], S4 [Gu et al., 2022a] in Table 8.2. We note that *all these baselines flatten the image into a 1D sequence and apply 1D sequence models, and do not take into account the underlying topology*. For our experiments, we simply replace the SSD layer in the Mamba block introduced in Dao and Gu [2024a] with Chimera, and use the ViT-B training recipe with no additional hyperparameter tuning.

[2] We use a slightly modified normalization scheme for the undirected line graph method to allow for larger selectivity values in the adjacency matrix. See Appendix 15.2.1 for details

Table 8.2: Top-1, Top-5 accuracies of various methods on ImageNet-1K.

Method (88M)	Top-1 (%)		Top-5 (%)	
	Acc	Acc_{EMA}	Acc	Acc_{EMA}
ViT-B	78.8	80.6	94.2	95.2
S4-ViT-B	79.4	80.4	94.2	95.1
Hyena-ViT-B	78.4	76.4	94.0	93.0
Chimera-ViT-B	**81.4**	**82.1**	**95.4**	**95.9**

Table 8.3: Ablation: Comparing 2D grid structure with 1D flattening of patches.

Method (22M)	Top-1 (%)		Top-5 (%)	
	Acc	Acc_{EMA}	Acc	Acc_{EMA}
Fwd (1D)	73.8	73.8	91.6	91.6
Fwd & Rev (1D)	76.5	75.6	93.4	92.8
2D DAG	**77.8**	**76.7**	**93.9**	**93.5**

Table 8.2 shows that Chimera's 2D-DAG decomposition outperforms ViT by 2.6%. We note that our method does not require any additional position embeddings which are still an active area of research for ViT [Heo et al., 2024]. Furthermore, we outperform methods such as Hyena [Poli et al., 2023] by 3%, and S4 [Gu et al., 2022a] by 2% that linearize the data and then apply an SSM on it.

To demonstrate the importance of incorporating topology, we perform an ablation where we progressively degrade the grid-graph structure, observing a monotonic drop in performance. We consider three topologies: **2D DAG** is the 2D DAG decomposition that retains the grid structure (Fig 8.5, right); **Fwd & Rev (1D)** flattens the grid into a 1D sequence with bidirectional edges like ViT (Fig 8.6, top); **Fwd (1D)** is a 1D graph with only forward edges (Fig 8.6, bottom). We observe from Table 8.3 that as the topology is lost, the accuracy drops from 77.8% (2D-DAG) to 76.5% (Fwd & Rev) to 73.8% (Fwd).

8.5.3 Long Range Graph Benchmark

We evaluate Chimera on the Long Range Graph Benchmark (LRGB) [Dwivedi et al., 2022]. This benchmark comprises tasks designed to challenge models in their ability to effectively capture both local and long-range interactions within graph structures. We compare against convolution-based (GCN Kipf and Welling [2016], GatedGCN Bresson and Laurent [2017]), Transformer-based (GraphGPS Rampášek et al. [2022]), Mamba-based (Graph-Mamba Wang et al. [2024], Graph Mamba Behrouz and Hashemi [2024]), and other baselines like GINE Hu et al. [2019], as well as their hyperparameter tuned versions introduced in Tönshoff et al. [2023]. These baselines incorporate topology using a variety of techniques: convolution ones use local aggregation, transformer ones use local and global aggregation via position embeddings, and Mamba ones use "data flattening" along with random walks, position embeddings, and

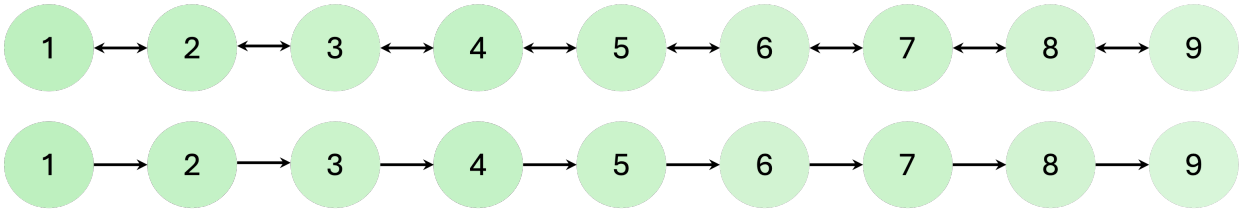


Figure 8.6: Progressively destroying the 2D grid graph topology. *Fwd & Rev* (top): 1D flattened grid with bidirectional edges. *Fwd* (bottom): 1D flattened grid graph with only forward edges.

Table 8.4: Evaluation of Chimera on LRGB Tasks [Dwivedi et al., 2022]. The first section shows the best performing numbers cited in the papers that introduce the given baselines. The second section shows the result of better hyperparameter tuned baselines introduced by Tönshoff et al. [2023]. Finally, we also compare with other baselines that use SSMs as a blackbox replacement for a Transformer.bolding seems inconsistent (see 4th column)

Method (< 500k params)	Peptides-Func	Peptides-Struct	PascalVOC-SP	COCO-SP
	AP (↑)	MAE (↓)	F1 (↑)	F1 (↑)
GCN [Kipf and Welling, 2016]	0.5930±0.0023	0.3496±0.0013	0.1268±0.0060	0.0841±0.0010
GINE [Hu et al., 2019]	0.5498±0.0079	0.3547±0.0045	0.1265±0.0076	0.1339±0.0044
Gated-GCN [Bresson and Laurent, 2017]	0.5864±0.0077	0.3420±0.0013	0.2873±0.0219	0.2641±0.0045
SAN+LapPE [Kreuzer et al., 2021]	0.6384±0.0121	0.2683±0.0043	0.3230±0.0039	0.2592±0.0158
Exphormer [Shirzad et al., 2023]	0.6527±0.0043	0.2481±0.0007	0.3975±0.0037	0.3430±0.0108
GPS+BigBird [Rampášek et al., 2022]	0.5854±0.0079	0.2842±0.0130	0.2762±0.0069	0.2622±0.0008
GraphGPS+Transformer [Rampášek et al., 2022]	0.6575±0.0049	0.2510±0.0015	0.3689±0.0131	0.3774±0.0150
GCN [Tönshoff et al., 2023]	0.6860 ± 0.0050	**0.2460 ± 0.0007**	0.2078 ± 0.0031	0.1338 ± 0.0007
Gated-GCN [Tönshoff et al., 2023]	0.6765 ± 0.0047	0.2477 ± 0.0009	0.3880 ± 0.0040	0.2922 ± 0.0018
GINE [Tönshoff et al., 2023]	0.6621 ± 0.0067	0.2473 ± 0.0017	0.2718 ± 0.0054	0.2125 ± 0.0009
GraphGPS+Transformer [Tönshoff et al., 2023]	0.6534 ± 0.0091	0.2509 ± 0.0014	0.4440 ± 0.0054	0.3884 ± 0.0055
Graph-Mamba [Wang et al., 2024]	0.6739 ± 0.0087	0.2478 ± 0.0016	0.4191 ± 0.0126	0.3960 ± 0.0175
Graph Mamba [Behrouz and Hashemi, 2024]	**0.7071 ± 0.0083**	0.2473 ± 0.0025	0.4393 ± 0.0112	0.3974 ± 0.0101
Chimera (Ours)	**0.7021 ± 0.003**	**0.2460 ± 0.0002**	**0.496 ± 0.007**	**0.3977 ± 0.016**

Table 8.5: Ablation: Chimera with approximate resolvent is competitive with the Transformer baseline.

Method	Peptides-Func	Peptides-Struct	PascalVOC-SP	COCO-SP
	AP (↑)	MAE (↓)	F1 (↑)	F1 (↑)
GraphGPS+Transformer	0.6534 ± 0.0091	<u>0.2509</u> ± 0.0014	0.4440 ± 0.0054	<u>0.3884</u> ± 0.0055
Chimera (Approx)	<u>0.6709</u> ± 0.0089	0.2521 ± 0.0006	<u>0.4508</u> ± 0.0367	0.3709 ± 0.0009
Chimera (Ours)	**0.7021 ± 0.003**	**0.2460 ± 0.0002**	**0.496 ± 0.007**	**0.3977 ± 0.016**

local encodings. The diversity of these methods highlights the significant research effort dedicated to heuristics to incorporate topology, in contrast to our unified approach.

We show that Chimera achieves state-of-the-art results across all LRGB tasks (Table 8.4). Notably, we observe that on tasks such as Peptides-Func and Peptides-Struct, where convolution-based models typically outperform transformers, Chimera outperforms or matches their performance. Furthermore, on tasks like PascalVOC and COCO where transformers do well, Chimera consistently surpasses all baselines, with a more than 12% improvement on PascalVOC. This validates our grounded approach which effectively captures both local and global information.

In Table 8.5, we evaluate the approximate variant of Chimera with a finite-sum relaxation (Sec 8.4.1) that truncates the Neumann series at the average graph diameter of the graph. We show that the approximation variant matches the strong transformer baseline of GraphGPS, however fully leveraging the entire graph structure in Chimera provides clear performance benefits.

8.6 Conclusion and Future Work

In this work, we propose Chimera, a unified framework that mathematically generalizes State Space Models (SSMs) to incorporate the underlying data topology. Unlike previous approaches that rely on carefully engineered heuristics and treat attention and SSMs as black boxes, our method breaks open this black box by providing a principled, domain-agnostic framework for modeling diverse data modalities. We show that Chimera achieves state-of-the-art performance across domains including language, vision, and graph tasks, consistently surpassing highly tuned domain-specific baselines, which validates our premise and the proposed solution. Furthermore, we also show that for structured domains like sequences and images, Chimera has an efficient linear complexity by leveraging our DAG decomposition technique, recovering the complexity of SSMs like Mamba-2.

Our work is the first step toward developing unified models for diverse data modalities. We believe that extending the DAG decomposition technique to general graphs to achieve linear complexity is an exciting direction for future work. Furthermore, we hope that the research community applies Chimera to more domains with an inherent underlying topology, and establishes Chimera as a strong baseline for further research in those domains.

9 Towards Characterizing the Value of Edge Embeddings in Graph Neural Networks

Abstract: *Graph neural networks (GNNs) are the dominant approach to solving machine learning problems defined over graphs. Despite much theoretical and empirical work in recent years, our understanding of finer-grained aspects of architectural design for GNNs remains impoverished. In this paper, we consider the benefits of architectures that maintain and update edge embeddings. On the theoretical front, under a suitable computational abstraction for a layer in the model, as well as memory constraints on the embeddings, we show that there are natural tasks on graphical models for which architectures leveraging edge embeddings can be much shallower. Our techniques are inspired by results on time-space tradeoffs in theoretical computer science. Empirically, we show architectures that maintain edge embeddings almost always improve on their node-based counterparts—frequently significantly so in topologies that have "hub" nodes.*

9.1 Introduction

Graph neural networks (GNNs) have emerged as the dominant approach for solving machine learning tasks on graphs. Over the span of the last decade, many different architectures have been proposed, both in order to improve different notions of efficiency, and to improve performance on a variety of benchmarks. Nevertheless, theoretical and empirical understanding of the impact of different architectural design choices remains elusive.

One previous line of work [Xu et al., 2018] has focused on characterizing the representational limitations stemming from the *symmetry-preserving* properties of GNNs when the node features are not informative (also called "anonymous GNNs") — in particular, relating GNNs to the Weisfeiler-Lehman graph isomorphism test [Leman and Weisfeiler, 1968]. Another line of work [Oono and Suzuki, 2019] focuses on the potential pitfalls of the *(over)smoothing effect* of deep GNN architectures, with particular choices of weights and non-linearities, in an effort to explain the difficulties of training deep GNN models. Yet another [Black et al., 2023] focuses on training difficulties akin to vanishing introduced by *"bottlenecks"* in the graph topology.

In this paper, we focus on the benefits of maintaining and updating *edge embeddings* over the course of the computation of the GNN. More concretely, a typical way to parametrize a layer l of a GNN [Xu et al., 2018] is to maintain, for each node v in the graph, a node embedding $h_v^{(l)}$, which is calculated as

$$a_v^{(l+1)} = \text{AGGREGATE}\left(h_u^{(l)} : u \in N_G(v)\right) \qquad h_v^{(l+1)} = \text{COMBINE}\left(a_v^{(l+1)}, h_v^{(l)}\right) \tag{9.1}$$

where $N_G(v)$ denotes the neighborhood of vertex v. These updates can be viewed as implementing a (trained) message-passing algorithm, in which nodes pass messages to their neighbors, which are then aggregated and combined with the current state (i.e., embedding) of a node. The initial node embeddings $h_v^{(0)}$ are frequently part of the task specification (e.g., a vector of fixed features that can be associated with each node). When this is not the case, they can be set to fixed values (e.g., the all-ones vector) or random values.

But a more expressive way to parametrize a layer of computation is to maintain, for each *edge* e, an edge embedding $h_e^{(l)}$ which is calculated as:

$$a_e^{(l+1)} = \text{AGGREGATE}\left(h_a^{(l)} : a \in M_G(e)\right) \qquad h_e^{(l+1)} = \text{COMBINE}\left(a_e^{(l+1)}, h_e^{(l)}\right) \tag{9.2}$$

where $M_G(e)$ denotes the "neighborhood" of edge e: that is, all edges a that share a vertex with e[1].

This paradigm is at least as expressive as Equation 9.1: we can simulate a layer of Equation 9.1 by designating the embedding of an edge to be the concatenation of the node embeddings of its endpoints, and noticing that $M_G(e)$ includes all the neighbors of both endpoints of e. In particular, if a task has natural initial node embeddings, then their concatenations along edges can be used as initial edge embeddings. Additionally, there may be tasks where initial features are most naturally associated with edges (e.g., attributes of the relationship between two nodes) — or the final predictions of the network are most naturally associated with edges (e.g., in link prediction, where we want to decide which potential links are true links).

GNNs that fall in the general paradigm of Equation 9.2 have been used for various applications – including link prediction [Cai et al., 2021, Liang and Pu, 2023] as well as reasoning about relations between objects [Battaglia et al., 2016], molecular property prediction [Gilmer et al., 2017, Choudhary and DeCost, 2021], and detecting clusters of communities in graphs [Chen et al., 2017] – with robust empirical benefits. These approaches instantiate the edge-based paradigm in a plethora of ways. However, it is difficult to disentangle to what degree performance improvements come from added information from domain-specific initial edge embeddings, versus properties of the particular architectural choices for the aggregation functions in Equation 9.2, versus inherent benefits of the edge-based paradigm itself (whether representational, or via improved training dynamics).

We focus on *theoretically and empirically* quantifying the added *representational* benefit from maintaining edge embeddings. Viewing the GNN as a computational model, we can think of the intermediate embeddings as a "scratch pad". Since we maintain more information per layer compared to the node-based paradigm Equation 9.1, we might intuitively hope to be able to use a shallower edge embedding model. However, formally proving depth lower bounds both for general neural networks [Telgarsky, 2016] and for specific architectures [Sanford et al., 2024b,a] frequently requires non-trivial theoretical insights – as is the case for our question of interest. In this paper, we show that:

[1]The graph is assumed to be undirected, as is most common in the GNN literature.

- *Theoretically*, for certain graph topologies, edge embeddings can have substantial *representational* benefits in terms of the depth of the model, when the amount of memory (i.e., total bit complexity) per node or edge embedding is bounded. Our results illuminate some subtleties of using particular lenses to understand design aspects of GNNs: for instance, we prove that taking memory into account reveals depth separations that the classical lens of invariance [Xu et al., 2018] alone cannot.

- *Empirically*, when given the same input information, edge-based models almost always lead to performance improvements compared to their node-based counterpart — and often by a large margin if the graph topology includes "hub" nodes with high degree.

9.2 Overview of results

9.2.1 Representational benefits from maintaining edge embeddings.

Our theoretical results elucidate the representational benefits of maintaining edge embeddings. More precisely, we show that there are natural tasks on graphs that can be solved by a *shallow* model maintaining constant-size edge embeddings, but can only be solved by a model maintaining constant-size node embeddings if it is much *deeper*.

To reason about the impact of depth on the representational power of edge-embedding-based and node-embedding-based architectures, we introduce two *local computation models*. In the node-embedding case, we assume each node of the graph G supports a processor that maintains a state with a *fixed amount of memory*. In one round of computation, each node receives messages from the adjacent nodes, which are aggregated by the node into a new state. In this abstraction, we think of the memory of the processor as the total bits of information each embedding can retain, and we think of one round of the protocol as corresponding to one layer of a GNN. The edge-embedding case is formalized in a similar fashion, except that the processors are placed on the edges of the graph, and two edge processors are "adjacent" if the edges share a vertex in common. In both cases, the input is distributed across the edges of the graph, and is only locally accessible.

With this setup in mind, our first result focuses on *probabilistic inference* on graphs, specifically, the task of maximum a-posteriori (MAP) estimation in a pairwise graphical model on a graph $G = (V, E)$. For this task, given edge attributes describing the pairwise interactions $\phi_{\{a,b\}}$, the goal is to compute $\arg\max_{x\in\{0,1\}^V} p_\phi(x)$, where $p_\phi(x) \propto \exp\left(\sum_{\{a,b\}\in E} \phi_{\{a,b\}}(x_a, x_b)\right)$.

Theorem (Informal)**.** *Consider the task of using a GNN to calculate MAP (maximum a-posteriori) values in a pairwise graphical model, in which the pairwise interactions are given as input embeddings to a node-embedding or edge-embedding architecture. Then, there exists a graph with $O(n)$ vertices and edges, such that:*

- *Any node message-passing protocol with T rounds and $O(1)$ bits of memory per node processor requires $T = \Omega(\sqrt{n})$.*

- *There is an edge message-passing protocol with $O(1)$ rounds and $O(1)$ bits of memory.*

The proof techniques are of standalone interest: the lower bound on node message-passing protocols is inspired by tracking the "flow of information" in the graph, reminiscent of graph pebbling techniques used to prove time-space tradeoffs in theoretical computer science [Grigor'ev, 1976, Abrahamson, 1991]. The formal result is Theorem 11, and the proof sketch is included in Section 9.5.

The view from symmetry. Above, we are not imposing any *symmetry constraints* – that is, invariance of the computation at a node or edge to its identity and the identities of its neighbors. Indeed, the edge message-passing protocol constructed above is highly non-symmetric. However, we show there is a (different, but also natural) task where *even symmetric* edge message-passing protocols achieve a better depth/memory tradeoff than node message-passing protocols. We state the informal result below; the formal result is Theorem 12.

Theorem (Informal)**.** *Let n be a positive integer. There is a graph G with $O(n)$ vertices and $O(n)$ edges, and a computational task on G, such that:*

- *Any node message-passing protocol with T rounds and $O(1)$ bits of memory per node processor requires $T = \Omega(\sqrt{n})$ to solve this task.*
- *There is a symmetric edge message-passing protocol that solves this task with $O(1)$ rounds and $O(1)$ bits of memory.*

Importance of the memory lens. The memory constraints are crucial for the results above. Without memory constraints, we can show that the node message-passing architecture can simulate the edge message-passing architecture, while only increasing the depth by 1 (Proposition 12). Moreover, the *symmetric* node message-passing architecture can simulate the *symmetric* edge message-passing architecture, again while only increasing the depth by 1. We state the informal result below; the formal result is Theorem 13.

Theorem (Informal)**.** *For any graph G, any symmetric edge message-passing protocol on G with T rounds can be represented by a symmetric node message-passing protocol with $T + 1$ rounds.*

We note that unlike prior work that focuses on understanding the representational power of GNN architectures under symmetry constraints [Xu et al., 2018] — which requires that the initial node features are the same for all nodes — our simulation theorem above holds for arbitrary choices of initial node features.

We view this as evidence that many fine-grained properties of architectural design for GNNs cannot be adjudicated by solely considering them through the lens of symmetries of the network.

9.2.2 Empirical benefits of edge-based architectures.

The theory, while only characterizing representational power, suggests that architectures that maintain edge embeddings should have strictly better performance compared to their node embedding counterparts. We verify this in both real-life benchmarks and natural synthetic sandboxes.

First, we consider several popular GNN benchmarks (inspired by both predicting molecular properties, and image-like data), and show that equalizing for all other aspects of the architecture (e.g., depth, dimensionality of the embeddings) — the accuracy the edge-based architectures achieve is at least as good as their node-based counterparts. Note, the goal of these experiments is *not* to propose a new architecture — there are already a variety of (very computationally efficient) GNNs that in some manner maintain edge embeddings. The goal is to confirm that — all other things being equal — the representational advantages of edge-based architectures do not introduce additional training difficulties. Details are included in Section 9.8.1.

Next, we consider two synthetic settings to stress test the performance of edge-based architectures. Inspired by the graph topology that provides a theoretical separation between edge and node-based protocols (Theorem 11 and Theorem 12), we consider graphs in which there is a hub node, and tasks that are "naturally" solved by an edge-based architecture. Precisely, we consider a star graph, in which the labels on the leaves are generated by a "planted" edge-based architecture with randomly chosen weights. The node-based architecture, on the other hand, has to pass messages between the leaves indirectly through the center of the star. Empirically, we indeed observe that the performance of edge-based architectures is significantly better. Details are included in Section 9.8.2.

Finally, again inspired by the theoretical setting in Theorem 11, we consider probabilistic inference on *tree graphs* — precisely, learning a GNN that calculates node marginals for an Ising model, a pairwise graphical model in which the pairwise interactions are just the product of the end points. An added motivation for this setting is the fact that belief propagation — a natural algorithm to calculate the marginals — can be written as an edge-based message-passing algorithm. Again, empirically we see that edge-based architectures perform at least as well as node-based architectures. This advantage is maintained even if we consider "directed" versions of both architectures, in which case embeddings are maintained to be sent along each direction of the edge, and the message for the outgoing direction of an edge depends only on the embeddings corresponding to the incoming directions of the edges. Details are included in Section 9.8.3.

9.3 Related Works

The symmetry lens on GNNs: The most extensive theoretical work on GNNs has concerned itself with the representational power of different GNN architectures, while trying to preserve equivariance (to permuting the neighbors) of each layer. [Xu et al., 2018] connected the expressive power of such architectures to the Weisfeiler-Lehman (WL) test for graph isomorphism. Subsequent works [Maron et al., 2019, Zhao et al., 2021] focused on strengthening the representational power of the standard GNN architectures from the perspective of symmetries—more precisely, to simulate the k-WL test, which for k as large as the size of the graph becomes as powerful as testing graph isomorphism. Our work suggests that this perspective may be insufficient to fully understand the representational power of different architectures.

GNNs as a computational machine: Two recent papers [Loukas, 2019, 2020] considered properties of GNNs when viewed as "local computation" machines, in which a layer of computation allows a

node to aggregate the current values of the neighbors (in an arbitrary fashion, without necessarily considering symmetries). Using reductions from the CONGEST model, they provide lower bounds on width and depth for the standard node-embedding based architecture. However, they do not consider architectures with edge embeddings, which is a focus of our work.

Communication complexity methods to prove representational separations: Tools from distributed computation and communication complexity have recently been applied not only to understand the representational power of GNNs [Loukas, 2019, 2020], but also the representational power of other architectures like transformers [Sanford et al., 2024b,a]. In particular, [Sanford et al., 2024a] draws a connection between number of rounds for a MPC (Massively Parallel Computation) protocol, and the depth of attention-based architectures.

GNNs for inference and graphical models: The paper [Xu and Zou, 2023] considers the approximation power of GNNs for calculating marginals for pairwise graphical models, if the family of potentials satisfies strong symmetry constraints. They do not consider the role of edge embeddings or memory.

9.4 Setup

Notation. We will denote the graph associated with the GNN as $G = (V, E)$, denoting the vertex set as V and the edge set as E. The graph induces adjacency relations on both edges and nodes, namely for $v, v' \in V$ and $e, e' \in E$, we have: $v \sim v'$ if $\{v, v'\} \in E$; $v \sim e$ if $e = \{u, v\}$ for some $u \in V$; and $e \sim e'$ if e, e' share at least one vertex. For all graphs considered in this paper, we assume that $\{v, v\} \in E$ for all $v \in V$, so that adjacency is reflexive. We then define adjacency functions $N_G : V \cup E \to V$ and $M_G : V \cup E \to E$ as $N_G(a) := \{v \in V : a \sim v\}$ and $M_G(a) := \{e \in E : a \sim e\}$.

Local memory-constrained computation. In order to reason about the required depth with different architectures, we will define a mathematical abstraction for one layer of computation in the GNN. We will define two models for local computation, one for each of the edge-embedding and node-embedding architecture. Unlike much prior work on GNNs and distributed computation, we will also have *memory* constraints — more precisely, we will constrain the bit complexity of the node and edge embeddings being maintained.

In both models, there is an underlying graph $G = (V, E)$, and the goal is to compute a function $g : \Phi^E \to \{0, 1\}^V$, where Φ is the fixed-size *input alphabet*,via several rounds of message-passing on the graph G. This domain of g is Φ^E because in *both* models, the inputs are given on the edges of the graph — the node model will just be unable to store any *additional* information on the edges. As we will see in Section 9.5, this is a natural setup for probabilistic inference on graphs.

In both models, a protocol is parametrized by the number of rounds T required, and the amount of memory B required per local processor. For notational convenience, for $B \in \mathbb{N}$ we define $\mathcal{X}_B := \{0, 1\}^B$,

i.e. the length-B binary strings. Recall that $N_G(v)$, $M_G(v)$ denote the sets of vertices and edges adjacent to vertex v in graph G, respectively.

Definition 31 (Node message-passing protocol)**.** *Let $T, B \in \mathbb{N}$ and let $G = (V, E)$ be a graph. A* node message-passing protocol *P on graph G with T rounds and B bits of memory is a collection of functions $(f_{t,v})_{t\in[T],v\in V}$ where $f_{t,v} : \mathcal{X}_B^{N_G(v)} \times \Phi^{M_G(v)} \to \mathcal{X}_B$ for all t, v. For an* input *$I \in \Phi^E$, the* computation *of P at a round $t \in [T]$ is the map $P_t(\cdot; I) : V \to \mathcal{X}_B$ defined inductively by*

$$P_t(v; I) := f_{t,v}((P_{t-1}(v'; I))_{v'\in N_G(v)}, (I(e))_{e\in M_G(v)})$$

where $P_0 \equiv 0$. We say that P computes *a function $g : \Phi^E \to \{0, 1\}^V$ on inputs $\mathcal{I} \subseteq \Phi^E$ if $P_T(v; I)_1 = g(I)_v$ for all $v \in V$ and all $I \in \mathcal{I}$.*

In words, the value computed by vertex v at round t is some function of the previous values stored at the neighbors $v' \in N_G(v)$, as well as possibly the problem inputs on the edges adjacent to v (i.e. $(I(e))_{e\in M_G(v)}$). Note that $P_t(v; I)$ may indeed depend on $P_{t-1}(v; I)$, due to our convention that $v \in N_G(v)$. We can define the edge message-passing protocol analogously:

Definition 32 (Edge message-passing protocol)**.** *Let $T, B \in \mathbb{N}$ and let $G = (V, E)$ be a graph. An* edge message-passing protocol *P on graph G with T rounds and B bits of memory is a collection of functions $(f_{t,e})_{t\in[T],e\in E}$ where $f_{t,e} : \mathcal{X}_B^{M_G(e)} \times \Phi \to \mathcal{X}_B$ for all t, e, together with a collection of functions $(\tilde{f}_v)_{v\in[V]}$ where $\tilde{f}_v : \mathcal{X}_B^{M_G(v)} \to \{0, 1\}$. For an* input *$I \in \Phi^E$, the* computation *of P at a timestep $t \in [T]$ is the map $P_t(\cdot; I) : E \to \mathcal{X}_B$ defined inductively by:*

$$P_t(e; I) := f_{t,e}((P_{t-1}(e'; I))_{e'\in M_G(e)}, I(e))$$

where $P_0 \equiv 0$. We say that P computes *a function $g : \Phi^E \to \{0, 1\}^V$ on inputs $\mathcal{I} \subseteq \Phi^E$ if*

$$\tilde{f}_v((P_T(e; I))_{e\in M_G(v)}) = g(I)_v$$

for all $v \in V$ and all $I \in \mathcal{I}$.

Remark 6 (Relation to distributed computation literature)**.** *These models are very related to classical models in distributed computation like LOCAL [Linial, 1992] and CONGEST [Peleg, 2000]. However, the latter models ignore memory constraints, so we cannot usefully port lower and upper bounds from this literature.*

Remark 7 (Computational efficiency)**.** *In the definitions above, we allow the update rules $f_{t,v}$, $f_{t,e}$ to be arbitrary functions. In particular, a priori they may not be efficiently computable. However, our results showing a function can be implemented by an edge message-passing protocol (Theorem 11, Part 2 and Theorem 12, Part 2) in fact use simple functions (computable in linear time in the size of the neighborhood), implying the protocol can be implemented in parallel (with one processor per node/edge respectively) with parallel time complexity $O(TB \cdot \max_v |M_G(v)|)$. On the other hand, for the results showing a function cannot be implemented by a node message-passing protocol (Theorem 11, Part 1 and Theorem 12, Part 1), we prove an impossibility result for a* stronger *model (one in which the computational complexity of $f_{t,v}$ is unrestricted) — which makes our results only* stronger.

Symmetry-constrained protocols. Typically, GNNs are architecturally constrained to respect the symmetries of the underlying graph. Below we formalize the most natural notion of symmetry in our models of computation. Note, our abstraction of a round in the message-passing protocol generalizes the notion of a layer in a graph neural network—and the abstraction defined below correspondingly generalizes the standard definition of permutation equivariance [Xu et al., 2018]. We use the notation $\{\!\{\}\!\}$ to denote a multiset.

Definition 33 (Symmetric node message-passing protocol)**.** *A node message-passing protocol $P = (f_{t,v})_{t\in[T],v\in V}$ on graph $G = (V, E)$ is* symmetric *if there are functions $(f_t^{\mathsf{sym}})_{t\in[T]}$ so that for every $t \in [T]$ and $v \in V$, the function $f_{t,v}$ can be written as:*

$$f_{t,v}((c(v'))_{v'\in N_G(v)}, (I(e))_{e\in M_G(v)}) = f_t^{\mathsf{sym}}(c(v), \{\!\{(c(v'), I(\{v, v'\})) : v' \in N_G(v)\}\!\}).$$

Definition 34 (Symmetric edge message-passing protocol)**.** *An edge message-passing protocol*

$$P = ((f_{t,e})_{t\in[T],e\in E}, (\tilde{f}_v)_{v\in V})$$

on graph $G = (V, E)$ is symmetric *if there are functions $(f_t^{\mathsf{sym}})_{t\in[T]}$ and $\tilde{f}^{\mathsf{sym}}$ so that for every $t \in [T]$ and $e = \{u, v\} \in E$, the function $f_{t,e}$ can be written as:*

$$f_{t,e}((c(e'))_{e'\in M_G(e)}, I(e)) = f_t^{\mathsf{sym}}(I(e), c(e), \{\!\{\{\!\{c(\{u, v'\}) : v' \in N_G(u)\}\!\}, \{\!\{c(\{u', v\}) : u' \in N_G(v)\}\!\}\}\!\}),$$

and for every $v \in V$, $\tilde{f}_v$ can be written as $\tilde{f}_v((c(e))_{e\in M_G(v)}) = \tilde{f}^{\mathsf{sym}}(\{\!\{c(e) : e \in M_G(v)\}\!\})$.

9.5 Depth separation between edge and node message passing protocols under memory constraints

We will consider a common task in probabilistic inference on a *pairwise graphical model*: calculating the MAP (maximum a-posterior) configuration.

Definition 35 (Pairwise graphical model)**.** *For any graph $G = (V, E)$, the* pairwise graphical model *on G with potential functions $\phi_{\{a,b\}} : \{0, 1\}^2 \to \mathbb{R}$ is the distribution $p_\phi \in \Delta(\{0, 1\}^V)$ defined as*

$$p_\phi(x) \propto \exp\left(-\sum_{\{a,b\}\in E} \phi_{\{a,b\}}(x_a, x_b)\right).$$

Definition 36 (MAP evaluation)**.** *Let $\Phi \subseteq \{\phi : \{0, 1\}^2 \to \mathbb{R}\}$ be a finite set of potential functions. A* MAP (maximum a-posteriori) evaluator for *G (with potential function class Φ) is any function $g : \Phi^E \to \{0, 1\}^V$ that satisfies*

$$g(\phi) \in \arg\max_{x\in\{0,1\}^V} p_\phi(x)$$

for all $\phi \in \Phi^E$.

With this setup in mind, we will show that there exists a pairwise graphical model, and a local function class Φ, such that an edge message passing protocol can implement MAP evaluation with a constant

number of rounds and a constant amount of memory, while any node message protocol with T rounds and B bits of memory requires $TB = \Omega(\sqrt{|V|})$. Precisely, we show:

Theorem 11 (Main, separation between node and edge message-passing protocols)**.** *Fix $n \in \mathbb{N}$. There is a graph G with $O(n)$ vertices and $O(n)$ edges, and a function class Φ of size $O(1)$, so that:*

1. *Let g be any MAP evaluator for G with potential function class Φ. Any node message-passing protocol on G with T rounds and B bits of memory that computes g requires $TB \geq \sqrt{n} - 1$.*

2. *There is an edge message-passing protocol $(f_{t,e})_{t,e}$ on G with $O(1)$ rounds and $O(1)$ bits of memory that computes a MAP evaluator for G with potential function class Φ. Additionally, for all t, e, the update rule $f_{t,e}$ can be evaluated in $O(|M_G(e)|)$ time.*

We provide a proof sketch of the main techniques here, and relegate the full proofs to Appendix 16.1. The graph G that exhibits the claimed separation is a disjoint union of $\sqrt{n}$ path graphs, with an additional "hub vertex" that is connected to all other vertices in the graph (Figure 16.1). The intuition for the separation is that MAP estimation requires information to disseminate from one end of each path to the other, and the hub node is a bottleneck for node message-passing but not edge message-passing. We expand upon both aspects of this intuition below.

Lower bound for node message-passing protocols: Our main technical lemma for the first half of the theorem is Lemma 26. It gives a generic framework for lower bounding the complexity of any node message-passing protocol that computes some function g, by exhibiting a set of nodes $S \subset V$ where computing g requires large "information flow" from distant nodes. More precisely, for any fixed set of "bottleneck nodes" K, consider the radius-T neighborhood of S when K is removed from the graph. In any T-round protocol, input data from outside this neighborhood can only reach S by passing through K. But the total number of bits of information computed by K throughout the protocol is only $TB|K|$. This gives a bound on the number of values achievable by g on S. We formalize this argument below:

Lemma 26. *Let $G = (V, E)$ be a graph. Let P be a node message-passing protocol on G with T rounds and B bits of memory, which computes a function $g : \Phi^E \to \{0,1\}^V$. Pick any disjoint sets $K, S \subseteq V$. Define $H := G[\bar{K}], F := M_G(N_H^{T-1}(S))$. Then:*

$$TB \geq \frac{1}{|K|} \log \max_{I_F \in \Phi^F} \left| \left\{ g_S(I_F, I_{\overline{F}}) : I_{\overline{F}} \in \Phi^{\overline{F}} \right\} \right|.$$

Proof. First, we argue by induction that for each $t \in [T]$ and $v \in V \setminus K$, $P_t(v; I)$ is determined by $I_{M_G(N_H^{t-1}(v))}$ and $(P_\ell(k; I))_{\ell \in [t], k \in K}$. Indeed, by definition, $P_1(v; I)$ is determined by $I_{M_G^1(v)}$ for any $v \in V \setminus K$. For any $t > 1$ and $v \in V \setminus K$, $P_t(v; I)$ is determined by $(P_{t-1}(v'; I))_{v' \in N_G(v)}$ and $(I(e))_{e \in M_G(v)}$. Note that $N_G(v) \subseteq N_H(v) \cup K$. Thus, using the induction hypothesis for each $v' \in N_H(v)$, we get that $(P_{t-1}(v'; I))_{v' \in N_G(v)}$ is determined by $\bigcup_{v' \in N_H(v)} I_{M_G(N_H^{t-2}(v'))}$ and $(P_\ell(k; I))_{\ell \in [t], k \in K}$. So $P_t(v; I)$ is determined by $I_{M_G(N_H^{t-1}(v))}$ and $(P_\ell(k; I))_{\ell \in [t], k \in K}$, completing the induction.

Since P computes g and $S \subseteq V \setminus K$, we get that $g_S(I)$ is determined by $I_{M_G(N_H^{T-1}(S))} = I_F$ and $(P_\ell(k; I))_{\ell \in [T], k \in K}$. Thus, for any fixed $I_F \in \Phi_F$, we have

$$\left|\left\{g_S(I_F, I_{\overline{F}}) : I_{\overline{F}} \in \Phi^{\overline{F}}\right\}\right| \leq \left|\left\{(P_\ell(k; (I_F, I_{\overline{F}})))_{\ell \in [T], k \in K}) : I_{\overline{F}} \in \Phi^{\overline{F}}\right\}\right| \leq |\mathcal{X}_B|^{T|K|} = 2^{TB|K|}.$$

The lemma follows. □

Remark 8. *The proof technique is inspired by and related to classic techniques (specifically, Grigoriev's method) for proving time-space tradeoffs for restricted models of computation like branching programs ([Grigor'ev, 1976], see Chapter 10 in Savage [1998] for a survey). There, one defines the "flow" of a function, which quantifies the existence of subsets of coordinates, such that setting them to some value, and varying the remaining variables results in many possible outputs. In our case, the choice of subsets is inherently tied to the topology of the graph G. Our technique is also inspired by and closely related to the "light cone" technique for proving round lower bounds in the LOCAL computation model [Linial, 1992]. However, our technique takes advantage of bottlenecks in the graph to prove stronger lower bounds (which would be impossible in the LOCAL model where memory constraints are ignored).*

The proof of Part 1 of Theorem 11 now follows from an application of Lemma 26 with a particular choice of K and S. Specifically, we choose K to be the "hub" node (i.e. $K = \{0\}$) and S to be the set of left endpoints of each path. To show that any MAP evaluator has large information flow to S (in the quantitative sense of Lemma 26), it suffices to observe that in a pairwise graphical model on G where a different external field is applied to the right endpoint of each path, and all pairwise interactions along paths are positive, the MAP estimate on each vertex in S must match the external field on the corresponding right endpoint.

Upper bound for edge message-passing protocols: The key observation for constructing a constant-round edge message-passing protocol for MAP estimation on G is that all of the input data can be collected on the edges adjacent to the hub vertex. At this point, every such edge has access to all of the input data, and hence can evaluate the function. If G were an arbitrary graph, this final step would potentially be NP-hard. However, since the induced subgraph after removing the hub vertex is a disjoint union of paths, in fact there is a linear-time dynamic programming algorithm for MAP estimation on G (Lemma 48). This completes the proof overview for Theorem 11; we now provide the formal proof.

Proof of Theorem 11. Let G be the graph on vertex set $V := \{0\} \cup [\sqrt{n}] \times [\sqrt{n}]$ with edge set defined below (see also Figure 16.1):

$$E := \{\{0, (i, j)\} : i, j \in [\sqrt{n}]\} \cup \{\{(i, j), (i + 1, j)\} : 2 \leq i \leq \sqrt{n}, 1 \leq j \leq \sqrt{n}\}.$$

Define

$$\Phi := \{(x_a, x_b) \mapsto \mathbb{1}[x_a \neq x_b], (x_a, x_b) \mapsto \mathbb{1}[x_a \neq 1 \vee x_b \neq 1], (x_a, x_b) \mapsto \mathbb{1}[x_a \neq 0 \vee x_b \neq 0], (x_a, x_b) \mapsto 0\}.$$

First, let $g : \Phi^E \to \{0, 1\}^V$ be any MAP evaluator for G with potential function class Φ, and consider any node message-passing protocol on G with T rounds and B bits of memory that computes g. Let $K = \{0\}$ and $S = \{(1, j) : j \in [\sqrt{n}]\}$. Suppose that $T \leq \sqrt{n} - 2$. Let $F := M_G(N_H^{T-1}(S))$

and note that $\{(\sqrt{n}-1,j),(\sqrt{n},j)\} \notin F$ for all $j \in [\sqrt{n}]$. Let $I_F : F \to \Phi$ be the mapping that assigns the function $(x_a, x_b) \mapsto 0$ to each edge $\{0,(i,j)\} \in F$ and $(x_a, x_b) \mapsto \mathbb{1}[x_a \neq x_b]$ to each edge $\{(i,j),(i+1,j)\} \in F$. We claim that

$$\left|\left\{g_S(I_F, I_{\overline{F}}) : I_{\overline{F}} \in \Phi^{\overline{F}}\right\}\right| \geq 2^{\sqrt{n}}.$$

Indeed, for any string $y \in \{0,1\}^{\sqrt{n}}$, consider the mapping $I_{\overline{F}} : \overline{F} \to \Phi$ that assigns the function $(x_a, x_b) \mapsto \mathbb{1}[x_a \neq y_j \vee x_b \neq y_j]$ to each edge $\{(\sqrt{n}-1,j),(\sqrt{n},j)\} \in F$, assigns $(x_a, x_b) \mapsto 0$ to each edge $\{0,(i,j)\} \in E \setminus F$, and assigns $(x_a, x_b) \mapsto \mathbb{1}[x_a \neq x_b]$ to all remaining edges in $E \setminus F$. Then every minimizer of

$$\min_{x \in \{0,1\}^V} \sum_{\{a,b\} \in E} I_{\{a,b\}}(x_a, x_b)$$

satisfies $x_{(1,j)} = \cdots = x_{(\sqrt{n},j)} = y_j$ for all $j \in [\sqrt{n}]$. Hence, $g_S(I_F, I_{\overline{F}}) = y$. Since y was chosen arbitrarily, this proves the claim. But now Lemma 26 implies that $TB \geq \sqrt{n}$.

We now construct an edge message-passing protocol P on G with $T = 3$ and $B = 4$. We (arbitrarily) identify Φ with $\{0,1\}^2$. For all $i, j \in \sqrt{n}$, define

$$\begin{aligned} f_{1,\{(i,j),(i+1,j)\}}(x,y) &:= y && \text{if } i < \sqrt{n} \\ f_{2,\{0,(i,j)\}}(x,y) &:= (x_{\{(i,j),(i+1,j)\}}, x_{\{0,(i,j)\}}) && \text{if } i < \sqrt{n} \\ f_{3,\{0,(i,j)\}}(x,y) &:= (g_0(J(x)), g_{(i,j)}(J(x))) \end{aligned}$$

where the second line is well-defined since edge $\{0,(i,j)\}$ is adjacent to both itself and edge $\{(i,j),(i+1,j)\}$; and in the third line the function is computing g_0 and $g_{(i,j)}$ on the input $J(x) \in \Phi^E$ defined as

$$J(x)_e := \begin{cases} (x_{\{0,(k,\ell)\}})_{1:2} & \text{if } e = \{(k,\ell),(k+1,\ell)\} \\ (x_{\{0,(k,\ell)\}})_{3:4} & \text{if } e = \{0,(k,\ell)\} \end{cases},$$

where we use the notation $v_{a:b}$ for a vector v and indices $a, b \in \mathbb{N}$ to denote $(v_a, v_{a+1}, \ldots, v_b)$. Note that $J(x)$ is a well-defined function of x for every edge $\{0,(i,j)\}$, because $\{0,(i,j)\} \sim \{0,(k,\ell)\}$ for all $i, j, k, \ell \in [n]$. Finally, define all other functions $f_{t,e}$ to compute the all-zero function, and define

$$\tilde{f}_v(x) := \begin{cases} (x_{\{0,(1,1)\}})_{1:2} & \text{if } v = 0 \\ (x_{\{0,v\}})_{3:4} & \text{otherwise} \end{cases}.$$

This function is well-defined since $v = 0$ is adjacent to edge $\{0,(1,1)\}$ and any vertex $v \in V \setminus \{0\}$ is adjacent to edge $\{0,v\}$.

Fix any $I \in \Phi^E$. From the definition, it's clear that $P_2(\{0,(i,j)\}; I) = (I_{\{(i,j),(i+1,j)\}}, I_{\{0,(i,j)\}})$ for all I and $(i,j) \in [\sqrt{n}-1] \times [\sqrt{n}]$. Hence $J((P_2(e'; I))_{e' \in M_G(e)})_e = I$ for all edges e of the form $(0,\{i,j\})$, and so $P_3(\{0,(i,j)\}; I) = (g_0(I), g_{(i,j)}(I))$ for all $(i,j) \in [\sqrt{n}] \times [\sqrt{n}]$. This means that $\tilde{f}_v((P_3(e; I))_{e \in M_G(v)}) = g(I)_v$ for all $v \in V$, so the protocol indeed computes g.

It remains to argue about the computational complexity of the updates $f_{t,e}$. It's clear that for all $e \in E$ and $t \in \{1,2\}$, the function $f_{t,e}$ can be evaluated in input-linear time. The only case that requires proof is when $t = 3$ and $e = \{0,(i,j)\}$ for some $i, j \in \sqrt{n}$. In this case $|M_G(e)| = \Theta(n)$, so it suffices to

give an algorithm for evaluating the function $g : \Phi^E \to \{0,1\}^V$ on an explicit input J in $O(n)$ time. This can be accomplished via dynamic programming (Lemma 48). □

Remark 9. *A quantitatively stronger (and in fact tight) separation is possible if one considers general tasks rather than MAP estimation tasks – see Appendix 16.3.*

The separation discussed above crucially relies on the existence of a high-degree vertex in G. When the maximum degree of G is bounded by some parameter Δ, it turns out that any edge message-passing protocol can be simulated by a node message-passing protocol with roughly the same number of rounds and only a Δ factor more memory per processor. The idea is for each node to simulate the computation that would have been performed (in the edge message-passing protocol) on the adjacent edges. The following proposition formalizes this idea (proof in Appendix 16.1):

Proposition 12. *Let $T, B \geq 1$. Let $G = (V, E)$ be a graph with maximum degree Δ. Let P be an edge message-passing protocol on G with T rounds and B bits of memory. Then there is a node message-passing protocol P' on G that computes P with $T + 1$ rounds and $O(\Delta B)$ bits of memory.*

9.6 Depth separation under memory and symmetry constraints

One drawback of the separation in the previous section is that the constructed edge protocol was highly non-symmetric, whereas in practice GNN protocols are typically architecturally constrained to respect the symmetries of the underlying graph. In this section we prove that there is a separation between the memory/round trade-offs for node and edge message-passing protocols even under additional symmetry constraints.

Theorem 12. *Let $n \in \mathbb{N}$. There is a graph $G = (V, E)$ with $O(n)$ vertices and $O(n)$ edges, and a function $g : \{0,1\}^E \to \{0,1\}^V$, so that:*

1. *Any node message-passing protocol on G with T rounds and B bits of memory that computes g requires $TB \geq \Omega(\sqrt{n})$.*
2. *There is a symmetric edge message-passing protocol on G with $O(1)$ rounds and $O(\log n)$ bits of memory that computes g.*

For intuition, we start by sketching the proof of a relaxed version of the theorem, where the input alphabet is $[n]$ instead of $\{0,1\}$. We then discuss how to adapt the construction to binary alphabet.

Large-alphabet construction. Let $G = (V, E)$ be a star graph with root node 0 and leaves $\{1, \ldots, n\}$. We define a function $g : [n]^E \to \{0,1\}^V$ by $g(I)_v = 1$ if and only if there is some edge $e \neq \{0, v\}$ such that $I(e) = I(\{0, v\})$, i.e. the input on edge $\{0, v\}$ equals the input on some other edge. Since g is defined to be equivariant to relabelling the edges, and all edges are incident to each other, it is straightforward to see that there is a symmetric one-round edge message-passing protocol that computes

g with $O(\log n)$ memory (in contrast, the edge message-passing protocol constructed in Section 9.5 was not symmetric, as it required that the edges incident to the high-degree vertex were labelled by which path they belonged to). However, there is no low-memory, low-round *node* message-passing algorithm. Informally, this is because vertex 0 is an information bottleneck, and $\Omega(n)$ bits of information need to pass through it. Similar to in Section 9.5, this intuition can be made formal using Lemma 26.

Modifying for small alphabet. The large alphabet size seems crucial to the above construction: if we were to naively modify the above construction so that each edge takes input in $\{0,1\}$ (without changing the graph topology or the function g), then there *would* be a low-memory, low-round message-passing protocol, since the root node simply needs to compute the histogram of the leaves' inputs, which takes space $O(\log n)$. Each leaf node can use this information together with its own input value to compute its output. Essentially, there is no information bottleneck because there is a concise, sufficient "summary" of the input data.

However, the above construction can in fact be adapted to work with binary alphabet, by modifying the graph topology. At a high level, for each leaf node u in the above construction, we add n descendants and encode the input that was originally on u on the descendants of u, in unary. Of course, this new graph has n^2 nodes, so we must rescale parameters accordingly.

We now make this idea formal. For notational convenience, define $m = \lfloor \sqrt{n} \rfloor$. We define a graph $G = (V, E)$ that is a perfect n-ary tree of depth two. Formally, the graph G has vertex set $V = \{0\} \cup [m] \cup ([m] \times [m])$. Vertex 0 is adjacent to each $i \in [m]$, and each $i \in [m]$ is additionally adjacent to (i, j) for all $j \in [m]$. We define a function $g : \{0,1\}^E \to \{0,1\}^V$ as follows. On input $I \in \{0,1\}^E$, for each edge $e \in E$, define the *input summation* at e to be

$$C(I)_e := \sum_{e' \in M_G(e)} I(e').$$

Intuitively, one may think of $C(I)_e$ as simulating the input on e in the "large alphabet" construction described in Section 9.6. Next, define

$$\begin{aligned} g(I)_{(u,j)} &:= 0. \\ g(I)_u &:= \mathbb{1}[\#|e \in M_G(\{0,u\}) : C(I)_e = C(I)_{\{0,u\}}| > m+1]. \\ g(I)_0 &:= \mathbb{1}[\exists u \in [m] : g(I)_u = 1]. \end{aligned}$$

In words, $g(I)_u$ is the indicator for the event that, among the $2m+1$ edges adjacent to $\{0,u\}$ (which include $\{0,u\}$ itself), more than $m+1$ edges have the same input summation as $\{0,u\}$. At a high level, this definition of g was designed to satisfy three criteria. First, $g(I)_u$ depends on the input values on other branches of the tree: in particular, if $I_{\{0,v\}} = 0$ for all $v \in [n]$, then $C(I)_e = C(I)_{\{0,u\}}$ for all edges e in the subtree of u, so $g(I)_u$ exactly measures the event that there is *at least one* edge e outside the subtree of u for which $C(I)_e = C(I)_{\{0,u\}}$. Second, there is no concise "summary" of I such that $g(I)_u$ can be determined from this summary in conjunction with the inputs on the subtree of u. Third, $g(I)$ is equivariant to re-labelings of the tree.

The first two criteria, together with the fact that the root vertex 0 is an "information bottleneck" for G, can be used to show that any node message-passing algorithm that computes g on G requires either large memory or many rounds. The third criterion enables construction of a symmetric edge message-passing protocol for g. The arguments are formalized in the claims below.

Claim 1. *For graph G and function g as defined above, any node message-passing protocol on G that computes g with T rounds and B bits of memory requires $TB \geq \Omega(m)$.*

Proof. Consider any input $I \in \{0,1\}^E$ with $I(\{0,u\}) = 0$ for all $u \in [m]$. Then for any $u, j \in [m]$, we have

$$C(I)_{\{u,(u,j)\}} = C(I)_{\{0,u\}} = \sum_{i=1}^{m} I(\{u,(u,i)\}).$$

Thus $g(I)_u = 1$ if and only if there exists some $v \in [m] \setminus \{u\}$ with $C(I)_{\{0,u\}} = C(I)_{\{0,v\}}$, or equivalently $\sum_{i=1}^m I(\{u,(u,i)\}) = \sum_{i=1}^m I(\{v,(v,i)\})$.

Fix T, B and suppose that P is a node message-passing protocol on G that computes g with T rounds and B bits of memory. Define sets of vertices $K := \{0\}$ and $S := \{1, \ldots, m/2\}$. Let $H := G[\overline{K}]$ and $F := M_G(N_H^{T-1}(S))$. Then for any T, we have that

$$F = \{\{0,u\} : 1 \leq u \leq m/2\} \cup \{\{u,(u,j)\} : 1 \leq u \leq m/2, 1 \leq j \leq m\}.$$

Define a vector $I_F \in \Phi^F$ by

$$I_{\{0,u\}} = 0 \text{ for } 1 \leq u \leq m/2$$
$$I_{\{u,(u,j)\}} = \mathbb{1}[j \leq u] \text{ for } 1 \leq u \leq m/2, 1 \leq j \leq m.$$

Now fix any $x \in \{0,1\}^S$. We claim that there is some $I_{\overline{F}} \in \Phi^{\overline{F}}$ such that $g_S(I_F, I_{\overline{F}}) = x$. Indeed, let us define $I_{\overline{F}}$ by:

$$I_{\{0,v\}} = 0 \text{ for } m/2 < v \leq m$$
$$I_{\{v,(v,j)\}} = x_{v-m/2}\mathbb{1}[j \leq v - m/2] \text{ for } m/2 < v \leq m, 1 \leq j \leq m.$$

Then $C(I)_{\{0,u\}} = u$ for all $1 \leq u \leq m/2$, and $C(I)_{\{0,v\}} = (v - m/2)x_{v-m/2}$ for all $m/2 < v \leq m$. It follows that for any $1 \leq u \leq n/2$, $x_u = 1$ if and only if there exists some $v \in [m] \setminus u$ with $C(I)_{\{0,u\}} = C(I)_{\{0,v\}}$, and hence $x_u = g(I)_u$. We conclude that

$$\left|\left\{g_S(I_F, I_{\overline{F}}) : I_{\overline{F}} \in \Phi^{\overline{F}}\right\}\right| \geq 2^{m/2}.$$

Applying Lemma 26 we conclude that $TB \geq \Omega(m)$ as claimed. □

Claim 2. *For graph G and function g as defined above, there is a symmetric edge message-passing protocol on G that computes g with $O(1)$ rounds and $O(\log m)$ bits of memory.*

Proof. In the first round, each edge processor reads its input value. In the second round, each edge processor sums the values computed by all neighboring edges (including itself). In the third round, each edge processor computes the indicator for the event that strictly more than $m + 1$ neighboring edges (includ-

ing itself) have the same value as itself. In the final aggregation round, the output of a vertex is the indicator for the event that any neighbor has value 1.

By construction, the value computed by any edge e after the second round is exactly $C(I)_e$. Thus, after the third round, the value computed by any edge $\{0, u\}$ is exactly $g(I)_u$. Moreover, the value computed by any edge $\{u, (u, j)\}$ is 0 after the third round, since such edges only have $m + 1$ neighbors. It follows by construction of the final aggregation step that the protocol computes g. □

Proof of Theorem12. Immediate from Claim 1 and Claim 2. □

9.7 Symmetry alone provides no separation

In the previous sections we saw that examining *memory constraints* yields a separation between different GNN architectures (whether or not we take symmetry into consideration). In this section, we consider what happens if we solely consider *symmetry constraints* (that is, constraints imposed by requiring that the computation in a round of the protocol is invariant to permutations of the order of the neighbors). This viewpoint was initiated by Xu et al. [2018], who showed that when the initial node features are uninformative (that is, the same for each node), a standard GNN necessarily outputs the same answer for two graphs that are 1-Weisfeiler-Lehman equivalent (that is, graphs that cannot be distinguished by the Weisfeiler-Lehman test, even though they may not be isomorphic).

To be precise, we revisit the representational power of symmetric GNN architectures in the setting where the input features may be distinct and informative. We show that *if we remove the memory constraints* from Section 9.5, but *impose permutation invariance* for the computation in each round, any function that is computable by a T-layer edge message-passing protocol can be computed by a $(T + 1)$-layer node message-passing protocol. Note that this statement is incomparable to Proposition 12 because we impose constraints on symmetry, but remove constraints on memory.

Theorem 13 (No separation under symmetry constraints)**.** *Let $T \geq 1$. Let P be a symmetric edge message-passing protocol (Definition 34) on graph $G = (V, E)$ with T rounds. Then there is a $(T + 1)$-round symmetric node message-passing protocol (Definition 33) P' on G that computes the same function as P.*

Remark 10. *Theorem 13 and its proof are closely related to the fact that the 1-Weisfeiler-Lehman test is equivalent to the 2-Weisfeiler-Lehman test, which was reintroduced in the context of higher-order GNNs [Huang and Villar, 2021]. However, the k-Weisfeiler-Lehman test only characterizes the representational power of k-GNNs with uninformative input features (i.e. that are identical for all nodes). Theorem 13 shows that even with arbitrary input features on the edges, the computation of a GNN with edge embeddings and symmetric updates can be simulated by a GNN with only node embeddings, without losing symmetry.*

To prove Theorem 13, note that it suffices to simulate the protocol P for which the update rules f^{sym}, $\tilde{f}^{\mathsf{sym}}$ in Definition 34 are identity functions on the appropriate domains. In order to simulate P, we construct a symmetric node message-passing protocol P' for which the computation at time $t + 1$ and node v on input I is the multiset of features computed by P at time t at edges adjacent to v: $Q_t(v; I) :=$

$\{\!\{P_t(e;I) : e \in M_G(v)\}\!\}$. This is possible since the computation of P at time t and edge $e = (u, v)$ is $P_t(e;I) = (I(e), P_{t-1}(e;I), \{\!\{Q_{t-1}(u;I), Q_{t-1}(v;I)\}\!\})$. The node message-passing protocol is tracking $Q_{t-1}(\cdot;I)$; moreover, it can recursively compute $P_{t-1}(e;I)$ using the same formula. See Appendix 16.2 for the formal proof.

9.8 Empirical benefits of edge-based architectures

In this section we demonstrate that the representational advantages the theory suggests are borne out by experimental evaluations, both on real-life benchmarks and two natural synthetic tasks we provide. Note that all the experiments were done on a machine with 8 Nvidia A6000 GPUs.

9.8.1 Performance on common benchmarks

First we compare the performance of the most basic GNN architecture (Graph Convolutional Network, Kipf and Welling [2016]) with node versus edge embeddings. In the notation of Equation 9.1 and Equation 9.2, the AGGREGATE and COMBINE operations are integrated as a transformation that looks like Equation 9.3 or Equation 9.4:[2]

$$h_v^{(l+1)} = h_v^{(l)} + \sigma\big(W^{(l)}\text{MEAN}\big(h_w^{(l)} : w \in N_G(v) \setminus \{v\}\big)\big) \tag{9.3}$$

$$h_e^{(l+1)} = h_e^{(l)} + \sigma\big(W^{(l)}\text{MEAN}\big(h_f^{(l)} : f \in M_G(e) \setminus \{e\}\big)\big) \tag{9.4}$$

for trained matrices $W^{(l)}$ and a choice of non-linearity σ. The only difference between these architectures is that in the latter case, the message passing happens over the *line graph* of the original graph (i.e. the neighborhood of an edge is given by the other edges that share a vertex with it) — thus, this can be viewed as an ablation experiment in which the only salient difference is the type of embeddings being maintained. To also equalize the information in the input embeddings, we only use the node embeddings in the benchmarks we consider: for the edge-based architecture Equation 9.2, we initialize the edge embeddings by the concatenation of the node embeddings of the endpoints.

In Table 9.1, we show that *this single change* (without any other architectural modifications) uniformly results in the edge-based architecture at least matching the performance of the node-based architecture, sometimes improving upon it. *Note, the purpose of this table is not to advocate a new GNN architecture*[3] — but to confirm that the increased representational power of the edge-based architecture indicated by the theory also translates to improved performance when the model is trained. For each benchmark, we follow the best performing training configuration as delineated in [Dwivedi et al., 2023].

9.8.2 A synthetic task for topologies with node bottlenecks

The topologies of the graphs in Theorem 11 and Theorem 12 both involve a "hub" node, which is connected to all other nodes in the graph. Intuitively, in node-embedding architectures, such nodes have to mediate messages between many pairs of other nodes, which is difficult when the node is constrained by

[2] This is the "residual" parametrization, which we use in experiments unless otherwise stated.

[3] In particular, the edge-based architecture is often much more computationally costly to evaluate.

Model	ZINC	MNIST	CIFAR-10	Peptides-Func	Peptides-Struct
	MAE (↓)	ACCURACY (↑)	ACCURACY (↑)	AP (↑)	MAE (↓)
GCN	0.3430 ± 0.034	$\mathbf{95.29 \pm 0.163}$	55.71 ± 0.381	0.6816 ± 0.007	0.2453 ± 0.0001
Edge-GCN (Ours)	$\mathbf{0.3297 \pm 0.011}$	94.37 ± 0.065	$\mathbf{57.44 \pm 0.387}$	$\mathbf{0.6867 \pm 0.004}$	$\mathbf{0.2437 \pm 0.0005}$

Table 9.1: Comparison of node-based Equation 9.3 and edge-based Equation 9.4 GCN architectures across various graph benchmarks. The performance of the edge-based architecture robustly matches or improves the node-based architecture.

memory. To empirically stress test this intuition, we produce a synthetic dataset and train a GNN to solve a regression task on a graph with a fixed *star-graph* topology—a simpler topology than the constructions in Theorem 11 and Theorem 12—but capturing the core aspect of both. A star graph is a graph with a center node v_0, a set of n leaf nodes $\{v_i\}_{i\in[n]}$, and edge set $\{\{v_0, v_i\}_{i\in[n]}\}$. A training point in the dataset is a list $(x_i, y_i)_{i=1}^n$ where x_i is the *input feature* and y_i is the *label* for leaf node v_i.

The input features are in $\mathbb{R}^{10}$, and sampled from a standard Gaussian. The labels y_i are produced as outputs of a *planted* edge-based architecture. Namely, for a standard edge-based GCN as in Equation 9.4, we randomly choose values for the matrices $\{W_i\}_{i\in[k]}$ for some number of layers k, and set the labels to be the output of this edge-based GCN, when the initial edge features to the GCN are set as $h^{(0)}_{\{v_0,v_i\}} := x_i$, i.e. the input feature x_i at the corresponding leaf i. In Table 9.2, we show the performance of edge-based and node-based architectures on this dataset, varying the number of leaves n in the star graph and the depth k of the planted edge-based model. In each case, the numbers indicate RMSE of the best-performing edge-based and node-based architecture, sweeping over depths up to 10 ($2\times$ the planted model), widths $\in \{16, 32, 64\}$, and a range of learning rates.

Since the planted edge-based model satisfies both *invariance* constraints (by design of the GCN architecture) and *memory* constraints (since the planted model maintains 10-dimensional embeddings), we view these results as empirical corroboration of Theorem 12—and even for simpler topologies than the proof construction.

Number of Leaves	Depth of Planted Model (RMSE)					
	5		3		1	
	Edge	Node	Edge	Node	Edge	Node
64	0.004	0.3790	0.011	0.3596	0.008	0.3752
32	0.003	0.3664	0.005	0.3626	0.003	0.3614
16	0.007	0.3336	0.002	0.2100	0.002	0.2847

Table 9.2: Performance (in RMSE ↓) of edge-based and node-based architectures on a star-graph topology. The first number is the performance of the best edge-based model, and the second is the best node-based model, across a range of depths up to 10 ($2\times$ the planted model), widths $\in \{16, 32, 64\}$, and a range of learning rates.

9.8.3 A synthetic task for inference in Ising models

Finally, motivated by the probabilistic inference setting in Theorem 11, we consider a synthetic sandbox of using GNNs to predict the values of marginals in an Ising model [Ising, 1924, Onsager, 1944] – a natural type of pairwise graphical model where each node takes a value in $\{\pm 1\}$, and each edge potential is a weighted product of the edge endpoint values. Concretely, the probability distribution of an Ising model over graph $G = (V, E)$ has the form:

$$\forall x \in \{\pm 1\}^n : p_{J,h}(x) \propto \exp\Big(\sum_{\{i,j\} \in E} J_{\{i,j\}} x_i x_j + \sum_{i \in V} h_i x_i \Big).$$

Similar to in Section9.8.2, we construct a training set where the graph G and and edge potentials stay fixed (precisely, $J_{i,j} = 1$ for all $\{i, j\} \in E$). A training data-point consists of a vector of node potentials $\{h_i\}_{i \in [n]}$, and labels $\{\mathbb{E}[x_i]\}_{i \in [n]}$ consisting of the marginals from the resulting Ising model $p_{J,h}$. The node potentials are sampled from a standard Gaussian distribution.

There is a natural connection between GNNs and calculating marginals: a classical way to calculate $\{\mathbb{E}[x_i]\}$ when G is a *tree* is to iterate a message passing algorithm called *belief propagation* Equation 16.1, in which for each edge $\{i, j\}$ and direction $i \to j$, a message $\nu_{i \to j}^{(t+1)}$ is calculated that depends on messages $\{\nu_{k \to i}^{(t)}\}_{\{k,i\} \in E}$. The belief-propagation updates Equation 16.1 naturally fit the general edge-message passing paradigm from Equation 9.2. In fact, they fit even more closely a "directed" version of the paradigm, in which each edge $\{i, j\}$ maintains two embeddings $h_{i \to j}, h_{j \to i}$, such that the embedding for direction $h_{i \to j}$ depends on the embeddings $\{h_{k \to i}\}_{\{k,i\} \in E}$ — and it is possible to derive a similar "directed" node-based architecture (See Section 16.4.2). For both the undirected and directed version of the architecture, we see that maintaining edge embeddings gives robust benefits over maintaining node embeddings—for a variety of tree topologies including complete binary trees, path graphs, and uniformly randomly sampled trees of a fixed size. More details are included in Appendix 16.4.

9.9 Conclusions and future work

Graph neural networks are the best-performing machine learning method for many tasks over graphs. There is a wide variety of GNN architectures, which frequently make opaque design choices and whose causal influence on the final performance is difficult to understand and estimate. In this paper, we focused on understanding the impact of maintaining edge embeddings on the representational power, as well as the subtleties of considering constraints like memory and invariance. One significant downside of maintaining edge embeddings is the *computational* overhead on dense graphs. Hence, a fruitful direction for future research would be to explore more computationally efficient variants of edge-based architectures that preserve their representational power and performance.

Part V

Appendices

10 Appendix for Chapter 2

We begin by providing a brief overview of partial differential equations, some key results and useful lemmas. Hopefully this will also be useful for readers new to the field and want to learn and familiarize themselves with key definitions and basics.

10.1 Brief Overview of Partial Differential Equations

In this section, we introduce few key definitions and results from PDE literature. We note that the results in this section are standard and have been included in the Appendix for completeness. We refer the reader to classical texts on PDEs [Evans, 1998, Gilbarg and Trudinger, 2001] for more details.

We will use the following Poincaré inequality throughout our proofs.

Theorem 14 (Poincaré inequality)**.** *Given $\Omega \subset \mathbb{R}^d$, a bounded open subset, there exists a constant $C_p > 0$ such that for all $u \in H_0^1(\Omega)$*

$$\|u\|_{L^2(\Omega)} \leq C_p \|\nabla u\|_{L^2(\Omega)}.$$

Corollary 1. *For the bounded open subset $\Omega \subset \mathbb{R}^d$, for all $u \in H_0^1(\Omega)$, we define the norm in the Hilbert space $H_0^1(\Omega)$ as*

$$\|u\|_{H_0^1(\Omega)} = \|\nabla u\|_2. \tag{10.1}$$

Further, the norm in $H_0^1(\Omega)$ is equivalent to the norm $H^1(\Omega)$.

Proof. Note that for $u \in H_0^1(\Omega)$ we have,

$$\begin{aligned} \|u\|_{H^1(\Omega)} &= \|\nabla u\|_2 + \|u\|_2 \\ &\geq \|\nabla u\|_2 \\ \implies \|u\|_{H^1(\Omega)} &\geq \|u\|_{H_0^1(\Omega)}. \end{aligned}$$

Where we have used the definition of the norm in $H_0^1(\Omega)$ space.

Further, using the result in Theorem 14 we have

$$\|u\|_{H^1(\Omega)}^2 = \left(\|u\|_2^2 + \|\nabla u\|_{L^2(\Omega)}^2\right) \leq \left(C_p^2 + 1\right) \|\nabla u\|_{H^1(\Omega)}^2 \tag{10.2}$$

Therefore, combining the two inequalities we have

$$\|u\|_{H_0^1(\Omega)} \leq \|u\|_{H^1(\Omega)} \leq C_h \|u\|_{H_0^1(\Omega)} \tag{10.3}$$

where $C_h = (C_p^2 + 1)$. Hence we have that the norm in $H_0^1(\Omega)$ and $H^1(\Omega)$ spaces are equivalent. □

Proposition 13 (Equivalence between 2 and $H_0^1(\Omega)$ norms)**.** *If* $v \in \text{span}\{\varphi_1, \cdots, \varphi_k\}$ *then we have that* $\|v\|_2$ *is equivalent to* $\|v\|_{H_0^1(\Omega)}$.

Proof. We have from the Poincare inequality in Theorem 14 that for all $v \in H_0^1(\Omega)$, the norm in 2 is upper bounded by the norm in $H_0^1(\Omega)$, i.e.,

$$\|v\|_2^2 \leq \|v\|_{H_0^1(\Omega)}^2$$

Further, using results from Equation 10.5 and Equation 10.4 (where $b(u, v) := \langle Lu, v\rangle_2$), we know that for all $v \in H_0^1(\Omega)$ we have

$$m\|v\|_{H_0^1(\Omega)}^2 \leq \langle Lv, v\rangle_2 \leq \max\{M, C_p\|c\|_{L^\infty(\Omega)}\}\|v\|_{H_0^1(\Omega)}^2$$

This implies that $\langle Lu, v\rangle_2$ is equivalent to the inner product $\langle u, v\rangle_{H_0^1(\Omega)}$, i.e., for all $u, v \in H_0^1(\Omega)$,

$$m\langle u, v\rangle_{H_0^1(\Omega)} \leq \langle Lu, v\rangle_2 \leq \max\big\{M, C_p\|c\|_{L^\infty(\Omega)}\big\}\langle u, v\rangle_{H_0^1(\Omega)}$$

Further, since $v \in \text{span}\{\varphi_1, \cdots, \varphi_k\}$, we have from Lemma 2 that

$$\begin{aligned} \langle Lv, v\rangle_2 &\leq \lambda_k\|v\|_2^2 \\ \implies \|v\|_{H_0^1(\Omega)} &\leq \frac{\lambda_k}{c_1}\|v\|_2^2 \end{aligned}$$

Hence we have that for all $v \in \text{span}\{\varphi_1, \cdots, \varphi_k\}$ $\|v\|_2$ is equivalent to $\|v\|_{H_0^1(\Omega)}$ and by Corollary 1 is also equivalent to $\|v\|_{H^1(\Omega)}$. □

Now introduce a form for $\langle Lu, v\rangle_2$ that is more amenable for the existence and uniqueness results.

Lemma 27. *For all* $u, v \in H_0^1(\Omega)$*, we have the following,*

1. *The inner product* $\langle Lu, v\rangle_2$ *equals,*

$$\langle Lu, v\rangle_2 = \int_\Omega (A\nabla u \cdot \nabla v + cuv)\, dx$$

2. *The operator* L *is self-adjoint.*

Proof. 1. We will be using the following integration by parts formula,

$$\int_\Omega \frac{\partial u}{\partial x_i} dx = -\int_\Omega u\frac{\partial v}{\partial x_i} dx + \int_{\partial\Omega} uvn_i \partial\Gamma$$

Where n_i is a normal at the boundary and $\partial\Gamma$ is an infinitesimal element of the boundary.

Hence we have for all $u, v \in H_0^1(\Omega)$,

$$\begin{aligned}\langle Lu, v\rangle_2 &= \int_\Omega -\left(\sum_{i=1}^d (\partial_i (A\nabla u)_i)\right) v + cuv \; dx \\ &= \int_\Omega A\nabla u \cdot \nabla v dx - \int_{\partial\Omega} \left(\sum_{i=1}^d (A\nabla u)_i n_i\right) v d\Gamma + \int_\Omega cuv dx \\ &= \int_\Omega A\nabla u \cdot \nabla v dx + \int_\Omega cuv dx \qquad (\because v_{|\partial\Omega} = 0)\end{aligned}$$

2. To show that the operator $L : H_0^1(\Omega) \to H_0^1(\Omega)$ is self-adjoint, we show that for all $u, v \in H_0^1(\Omega)$ we have $\langle Lu, v\rangle = \langle u, Lv\rangle$.

   From Proposition 27, for functions $u, v \in H_0^1(\Omega)$ we have

$$\begin{aligned}\langle Lu, v\rangle_2 &= \int_\Omega A\nabla u \cdot \nabla v dx + \int_\Omega cuv dx \\ &= \int_\Omega A\nabla v \cdot \nabla u dx + \int_\Omega cvu dx \\ &= \langle u, Lv\rangle\end{aligned}$$

□

10.1.1 Proof of Proposition 1

We first show that if u is the unique solution then it minimizes the variational norm.

Let u denote the weak solution, further for all $w \in H_0^1(\Omega)$ let $v = u + w$. Using the fact that L is self-adjoint (as shown in Lemma 27) we have

$$\begin{aligned}J(v) = J(u+w) &= \frac{1}{2}\langle L(u+w), (u+w)\rangle_2 - \langle f, u+w\rangle_2 \\ &= \frac{1}{2}\langle Lu, u\rangle_2 + \frac{1}{2}\langle Lw, w\rangle_2 + \langle Lu, w\rangle_2 - \langle f, u\rangle_2 - \langle f, w\rangle_2 \\ &= J(u) + \frac{1}{2}\langle Lw, w\rangle_2 + \langle Lu, w\rangle_2 - \langle f, w\rangle_2 \\ &\geq J(u)\end{aligned}$$

where we use the fact that $\langle Lu, u\rangle_2 > 0$ and that u is a weak solution hence Equation 2.1 holds for all $w \in H_0^1(\Omega)$.

To show the other side, assume that u minimizes J, i.e., for all $\lambda > 0$ and $v \in H_0^1(\Omega)$ we have, $J(u + \lambda v) \geq J(u)$,

$$\begin{aligned} & J(u+\lambda v) \geq J(u) \\ & \frac{1}{2}\langle L(u+\lambda v),(u+\lambda v)\rangle_2 - \langle f,(u+\lambda v)\rangle_2 \geq \frac{1}{2}\langle Lu,u\rangle_2 - \langle f,u\rangle_2 \\ \implies & \frac{\lambda}{2}\langle Lv,v\rangle_2 + \langle Lu,v\rangle_2 - \langle f,v\rangle_2 \geq 0 \end{aligned}$$

Taking $\lambda \to 0$, we get

$$\langle Lu,v\rangle_2 - \langle f,v\rangle_2 \geq 0$$

and also taking v as $-v$, we have

$$\langle Lu,v\rangle_2 - \langle f,v\rangle_2 \leq 0$$

Together, this implies that if u is the solution to Equation 2.2, then u is also the weak solution, i.e, for all $v \in H_0^1(\Omega)$ we have

$$\langle Lu,v\rangle_2 = \langle f,v\rangle_2$$

Proof for Existence and Uniqueness of the Solution

In order to prove for the uniqueness of the solution, we first state the Lax-Milgram theorem.

Theorem 15 (Lax-Milgram, Lax and Milgram [1954])**.** *Let $\mathcal{H}$ be a Hilbert space with inner-product $(\cdot,\cdot) : \mathcal{H}\times\mathcal{H} \to \mathbb{R}$, and let $b : \mathcal{H}\times\mathcal{H} \to \mathbb{R}$ and $l : \mathcal{H} \to \mathbb{R}$ be the bilinear form and linear form, respectively. Assume that there exists constants $C_1, C_2, C_3 > 0$ such that for all $u, v \in \mathcal{H}$ we have,*

$$C_1\|u\|_{\mathcal{H}}^2 \leq b(u,u), \quad |b(u,v)| \leq C_2\|u\|_{\mathcal{H}}\|v\|_{\mathcal{H}}, \quad \text{and } |l(u)| \leq C_3\|u\|_{\mathcal{H}}.$$

Then there exists a unique $u \in \mathcal{H}$ such that,

$$b(u,v) = l(v) \quad \text{for all } v \in \mathcal{H}.$$

Having stated the Lax-Milgram Theorem, we make the following proposition,

Proposition 14. *Given the assumptions (i)-(iii), solution to the variational formulation in Equation 2.1 exists and is unique.*

Proof. Using the *variational formulation* defined in (2.1), we introduce the bilinear form $b(\cdot,\cdot) : H_0^1(\Omega)\times H_0^1(\Omega) \to \mathbb{R}$ where $b(u,v) := \langle Lu,v\rangle$. Hence, we prove the theorem by showing that the bilinear form $b(u,v)$ satisfies the conditions in Theorem 15.

We first show that for all $u, v \in H_0^1(\Omega)$ the following holds,

$$\begin{aligned}
|b(u,v)| &= \left| \int_\Omega (A\nabla u \cdot \nabla v + cuv) dx \right| \\
&\leq \int_\Omega |(A\nabla u \cdot \nabla v + cuv)| dx \\
&\leq \int_\Omega |A\nabla u \cdot \nabla v| dx + \int_\Omega |cuv| dx \\
&\leq \|A\|_{L^\infty(\Omega)} \|\nabla u\|_{L^2(\Omega)} \|\nabla v\|_{L^2(\Omega)} + \|c\|_{L^\infty(\Omega)} \|u\|_{L^2(\Omega)} \|v\|_{L^2(\Omega} \\
&\leq M \|\nabla u\|_{L^2(\Omega)} \|\nabla v\|_{L^2(\Omega)} + \|c\|_{L^\infty(\Omega)} \|u\|_{L^2(\Omega)} \|v\|_{L^2(\Omega)} \\
&\leq \max\{M, C_p \|c\|_{L^\infty(\Omega)}\} \|u\|_{H_0^1(\Omega)} \|v\|_{H_0^1(\Omega)} \qquad (10.4)
\end{aligned}$$

Now we show that the bilinear form $a(u,u)$ is lower bounded.

$$\begin{aligned}
b(v,v) &= \int_\Omega \left(A\nabla v \cdot \nabla v + cv^2 \right) dx \\
&\geq m \int_\Omega \|\nabla v\|^2 dx = m \|v\|_{H_0^1(\Omega)} \qquad (10.5)
\end{aligned}$$

Finally, for $v \in H_0^1(\Omega)$

$$|(f,v)| = \left| \int_\Omega fv dx \right| \leq \|f\|_{L^2(\Omega)} \|v\|_2 \leq C_p \|f\|_2 \|v\|_{H_0^1(\Omega)}$$

Hence, we satisfy the assumptions in required in Theorem 15 and therefore the variational problem defined in (2.1) has a unique solution. □

10.2 Perturbation Analysis

10.2.1 Proof of Lemma 3

Proof. Using the triangle inequality the error between $u^\star$ and $\tilde{u}^\star_{\text{span}}$, we have,

$$\|u^\star - \tilde{u}^\star_{\text{span}}\|_2 \leq \underbrace{\|u^\star - u^\star_{\text{span}}\|_2}_{(I)} + \underbrace{\|u^\star_{\text{span}} - \tilde{u}^\star_{\text{span}}\|_2}_{(II)} \qquad (10.6)$$

where $u^\star_{\text{span}}$ is the solution to the PDE $Lu = f_{\text{span}}$.

In order to bound Term (I), we use the inequality in Equation 2 to get,

$$\begin{aligned}\|u^\star - u^\star_{\text{span}}\|_2^2 &\leq \frac{1}{\lambda_1}\langle L(u^\star - u^\star_{\text{span}}), u^\star - u^\star_{\text{span}}\rangle_2 \\ &= \frac{1}{\lambda_1}\langle f - f_{\text{span}}, u^\star - u^\star_{\text{span}}\rangle_2 \\ &\leq \frac{1}{\lambda_1}\|f - f_{\text{span}}\|_2\|u^\star - u^\star_{\text{span}}\|_2 \\ \implies \|u^\star - u^\star_{\text{span}}\|_2 &\leq \frac{1}{\lambda_1}\|f - f_{\text{span}}\|_2 \leq \frac{\epsilon_{\text{span}}}{\lambda_1} \end{aligned} \tag{10.7}$$

We now bound Term (II).

First we introduce an intermediate PDE $Lu = \tilde{f}_{\text{span}}$, and denote the solution $\tilde{u}$. Therefore, by utilizing triangle inequality again Term (II) can be expanded as the following,

$$\|u^\star_{\text{span}} - \tilde{u}^\star_{\text{span}}\|_2 \leq \|u^\star_{\text{span}} - \tilde{u}\|_2 + \|\tilde{u} - \tilde{u}^\star_{\text{span}}\|_2 \tag{10.8}$$

We will tackle the second term in Equation 10.8 first.

Using $\tilde{u} = L^{-1}\tilde{f}_{\text{span}}$ and $\tilde{u}^\star_{\text{span}} = \tilde{L}^{-1}\tilde{f}_{\text{span}}$,

$$\begin{aligned}\|\tilde{u} - \tilde{u}^\star_{\text{span}}\|_2 &= \|(L^{-1} - \tilde{L}^{-1})\tilde{f}_{\text{span}}\|_2 \\ &= \|(L^{-1}\tilde{L} - I)\tilde{L}^{-1}\tilde{f}_{\text{span}}\|_2 \\ \implies \|\tilde{u} - \tilde{u}^\star_{\text{span}}\|_2 &= \|(L^{-1}\tilde{L} - I)\tilde{u}^\star_{\text{span}}\|_2 \end{aligned} \tag{10.9}$$

Therefore, using the inequality in Lemma 5 part (2.) we can upper bounded Equation 10.9 to get,

$$\|\tilde{u} - \tilde{u}^\star_{\text{span}}\|_2 \leq \delta\|\tilde{u}^\star_{\text{span}}\|_2 \tag{10.10}$$

where $\delta = \max\left\{\frac{\epsilon_A}{m}, \frac{\epsilon_c}{\zeta}\right\}$.

Proceeding to the first term in Equation 10.8, using Lemma 4, and the inequality in Equation 2, the term $\|u^\star_{\text{span}} - \tilde{u}\|_2$ can be upper bounded by,

$$\begin{aligned}\|u^\star_{\text{span}} - \tilde{u}\|_2^2 &\leq \frac{1}{\lambda_1}\langle L(u^\star_{\text{span}} - \tilde{u}), u^\star_{\text{span}} - \tilde{u}\rangle_2 \\ &\leq \frac{1}{\lambda_1}\langle f_{\text{span}} - \tilde{f}_{\text{span}}, u^\star_{\text{span}} - \tilde{u}\rangle_2 \\ &\leq \frac{1}{\lambda_1}\|f_{\text{span}} - \tilde{f}_{\text{span}}\|_2\|u^\star_{\text{span}} - \tilde{u}\|_2 \\ \implies \|u^\star_{\text{span}} - \tilde{u}\|_2 &\leq \frac{1}{\lambda_1}\|f_{\text{span}} - \tilde{f}_{\text{span}}\|_2 \leq \frac{\delta}{\lambda_1}\cdot\frac{\|f\|_2}{\gamma - \delta} \end{aligned} \tag{10.11}$$

Therefore Term (II), i.e., $\|u^\star_{\text{span}} - \tilde{u}^\star_{\text{span}}\|_2$ can be upper bounded by

$$\|u^\star_{\text{span}} - \tilde{u}^\star_{\text{span}}\|_2 \leq \|u^\star_{\text{span}} - \tilde{u}\|_2 + \|\tilde{u} - \tilde{u}^\star_{\text{span}}\|_2 \leq \frac{\hat{\epsilon}_f}{\lambda_1} + \delta\|\tilde{u}^\star_{\text{span}}\|_2 \tag{10.12}$$

Putting everything together, we can upper bound Equation 10.6 as

$$\begin{aligned}\|u^\star - \tilde{u}^\star_{\text{span}}\|_2 &\leq \|u^\star - u^\star_{\text{span}}\|_2 + \|u^\star_{\text{span}} - \tilde{u}^\star_{\text{span}}\|_2 \\ &\leq \frac{\epsilon_{\text{span}}}{\lambda_1} + \frac{\delta}{\lambda_1}\frac{\|f\|_2}{\gamma - \delta} + \delta\|\tilde{u}^\star_{\text{span}}\|_2\end{aligned}$$

where $\gamma = \frac{1}{\lambda_k} - \frac{1}{\lambda_{k+1}}$ and $\delta = \max\left\{\frac{\epsilon_A}{m}, \frac{\epsilon_c}{\zeta}\right\}$. □

10.2.2 Proof of Lemma 10

Proof. We define $r = \tilde{f}_{\text{span}} - f_{\text{nn}}$, therefore from Lemma 29 we have that for any multi-index α,

$$\|\tilde{L}^t r\|_2 \leq (t!)^2 \cdot C^t(\epsilon_{\text{nn}} + \epsilon_{\text{span}}) + 4\left(1 + \frac{\delta}{\gamma - \delta}\right)\lambda_k^t \|f_{\text{span}}\|_2.$$

For every $t \in \mathbb{N}$, we will write $u_t = \hat{u}_t + r_t$, s.t. $\hat{u}_t$ is a neural network and we (iteratively) bound $\|r_t\|_2$. Precisely, we define a sequence of neural networks $\{\hat{u}_t\}_{t=0}^\infty$, s.t.

$$\begin{cases}\hat{u}_0 = u_0, \\ \hat{u}_{t+1} = \hat{u}_t - \eta\left(\tilde{L}\hat{u}_t - f_{\text{nn}}\right)\end{cases}$$

Since $r_t = u_t - \hat{u}_t$, we can define a corresponding recurrence for r_t:

$$\begin{cases}r_0 = 0, \\ r_{t+1} = (I - \eta\tilde{L})r_t - r\end{cases}$$

Unfolding the recurrence, we get

$$r_{t+1} = \sum_{i=0}^{t}(I - \eta\tilde{L})^i r \tag{10.13}$$

Using the binomial expansion we can write:

$$
\begin{aligned}
(I-\eta\tilde{L})^t r &= \sum_{i=0}^{t}\binom{t}{i}(-1)^i(\eta\tilde{L})^i r \\
\implies \|(I-\eta\tilde{L})^{(t)}r\|_2 &= \left\|\sum_{i=0}^{t}\binom{t}{i}(-1)^i(\eta\tilde{L})^i r\right\|_2 \\
&\leq \sum_{i=0}^{t}\binom{t}{i}\eta^i\|\tilde{L}^i r\|_2 \\
&\leq \sum_{i=0}^{t}\left(\frac{te}{i}\right)^i\eta^i\|\tilde{L}^i r\|_2 \qquad \because \binom{t}{i}\leq\left(\frac{te}{i}\right)^i \\
&\overset{(1)}{\leq} \sum_{i=0}^{t}\left(\frac{te}{i}\eta\right)^i\left((i!)^2C^i(\epsilon_{\text{nn}}+\epsilon_{\text{span}})+4\left(1+\frac{\delta}{\gamma-\delta}\right)\lambda_k^i\|f_{\text{span}}\|_2\right) \\
&\leq \sum_{i=0}^{t}\left(\frac{te}{i}\eta\right)^i(i!)^2C^i\left((\epsilon_{\text{nn}}+\epsilon_{\text{span}})+4\left(1+\frac{\delta}{\gamma-\delta}\right)\frac{\lambda_k^i}{(i!)^2C^i}\|f_{\text{span}}\|_2\right) \\
&\overset{(2)}{\leq} \sum_{i=0}^{t}\left(\frac{te}{i}\eta i^2C\right)^i\left((\epsilon_{\text{nn}}+\epsilon_{\text{span}})+4\left(1+\frac{\delta}{\gamma-\delta}\right)\frac{\lambda_k^i}{(i!)^2C^i}\|f_{\text{span}}\|_2\right) \\
&\overset{(3)}{\leq} \sum_{i=0}^{t}\left(\frac{te}{i}\eta i^2C\right)^i\left((\epsilon_{\text{nn}}+\epsilon_{\text{span}})+4\left(1+\frac{\delta}{\gamma-\delta}\right)\lambda_k^i\|f_{\text{span}}\|_2\right) \\
&\leq \sum_{i=0}^{t}(tie\eta C)^i\left((\epsilon_{\text{nn}}+\epsilon_{\text{span}})+4\left(1+\frac{\delta}{\gamma-\delta}\right)\lambda_k^i\|f_{\text{span}}\|_2\right) \\
&\leq t\max\{1,(t^2e\eta C)^t\}\left((\epsilon_{\text{nn}}+\epsilon_{\text{span}})+4\left(1+\frac{\delta}{\gamma-\delta}\right)\lambda_k^t\|f_{\text{span}}\|_2\right)
\end{aligned}
$$

Here the inequality (1) follows by using the bound derived in Lemma 29. Further, we use that all $i\in\mathbb{N}$ we have $i!\leq i^i$ in (2) and the inequality (3) follows from the fact that $\frac{1}{(i!)^2C^i}\leq 1$.

Hence we have the the final upper bound:

$$
\|r_t\|_2 \leq t^2\max\{1,(t^2e\eta C)^t\}\left(\epsilon_{\text{nn}}+\epsilon_{\text{span}}+4\left(1+\frac{\delta}{\gamma-\delta}\right)\lambda_k^t\|f_{\text{span}}\|_2\right)
$$

□

10.3 Technical Lemmas: Perturbation Bounds

In this section we introduce some useful lemmas about perturbation bounds used in the preceding parts of the appendix.

First we show a lemma that's ostensibly an application of Davis-Kahan to the (bounded) operators L^{-1} and $\tilde{L}^{-1}$.

Lemma 28 (Subspace alignment)**.** *Consider linear elliptic operators L and $\tilde{L}$ with eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots$ and $\lambda_1 \leq \lambda_2 \leq \cdots$ respectively. Assume that $\gamma := \frac{1}{\lambda_k} - \frac{1}{\lambda_{k+1}} > 0$. For any function $g \in H_0^1(\Omega)$, we define $P_k g := \sum_{i=1}^k \langle g, \varphi_i \rangle_2 \varphi_i$ and $\tilde{P}_k g := \sum_{i=1}^k \langle g, \tilde{\varphi}_i \rangle_2 \tilde{\varphi}_i$ as the projection of g onto $\operatorname{span}\{\varphi_1, \cdots, \varphi_k\}$ and $\tilde{\Phi}_K$, respectively. Then we have:*

$$\|P_k g - \tilde{P}_k g\|_2 \leq \frac{\delta}{\gamma - \delta} \|g\|_2 \tag{10.14}$$

where $\delta = \max\left\{\frac{\epsilon_A}{m}, \frac{\epsilon_c}{\zeta}\right\}$.

Proof. We begin the proof by first showing that the inverse of the operators L and $\tilde{L}$ are close. Using the result from Lemma 5 with $\delta = \max\left\{\frac{\epsilon_A}{m}, \frac{\epsilon_c}{\zeta}\right\}$, we have:

$$\begin{aligned} & \langle (L^{-1}\tilde{L} - I)u, u \rangle_2 \leq \delta \|u\|_2^2 \\ \implies & \langle (L^{-1} - \tilde{L}^{-1})\tilde{L}u, u \rangle_2 \leq \delta \|u\|_2^2 \\ \implies & \langle (L^{-1} - \tilde{L}^{-1})v, u \rangle_2 \leq \delta \|u\|_2^2 \end{aligned}$$

Now, the operator norm $\|L^{-1} - \tilde{L}^{-1}\|$ can be written as,

$$\|L^{-1} - \tilde{L}^{-1}\| = \sup_{v \in H_0^1(\Omega)} \frac{\langle (L^{-1} - \tilde{L}^{-1})v, v \rangle_2}{\|v\|_2^2} \leq \delta \tag{10.15}$$

Further note that, $\{\frac{1}{\lambda_i}\}_{i=1}^\infty$ and $\{\frac{1}{\tilde{\lambda}_i}\}_{i=1}^\infty$ are the eigenvalues of the operators L^{-1} and $\tilde{L}^{-1}$, respectively. Therefore from *Weyl's Inequality* and Equation 10.15 we have:

$$\sup_i \left| \frac{1}{\lambda_i} - \frac{1}{\tilde{\lambda}_i} \right| \leq \|L^{-1} - \tilde{L}^{-1}\| \leq \delta \tag{10.16}$$

Therefore, for all $i \in \mathbb{N}$, we have that $\frac{1}{\tilde{\lambda}_i} \in [\frac{1}{\lambda_i} - \delta, \frac{1}{\lambda_i} + \delta]$, i.e., all the eigenvalues of $\tilde{L}^{-1}$ are within δ of the eigenvalue of L^{-1}. which therefore implies that the difference between k^{th} eigenvalues is,

$$\frac{1}{\tilde{\lambda}_k} - \frac{1}{\lambda_{k+1}} \geq \frac{1}{\lambda_k} - \frac{1}{\lambda_{k+1}} - \delta$$

Since the operators $L^{-1}, \tilde{L}^{-1}$ are bounded, the Davis-Kahan $\sin\Theta$ theorem [Davis and Kahan, 1970] can be used to conclude that:

$$\|\sin\Theta(\operatorname{span}\{\varphi_1, \cdots, \varphi_k\}, \tilde{\Phi}_K)\| = \|P_k - \tilde{P}_k\| \leq \frac{\|L^{-1} - \tilde{L}^{-1}\|}{\gamma - \delta} \leq \frac{\delta}{\gamma - \delta} \tag{10.17}$$

where $\|\cdot\|$ is understood to be the operator norm, and $\gamma = \frac{1}{\lambda_k} - \frac{1}{\lambda_{k+1}}$. Therefore for any function $g \in H_0^1(\Omega)$ we have

$$\begin{aligned}\|P_k g - \tilde{P}_k g\|_2 &\leq \|P_k - \tilde{P}_k\| \|g\|_2 \\ &\leq \frac{\|L^{-1} - \tilde{L}^{-1}\|}{\gamma - \delta} \|g\|_2\end{aligned}$$

By Equation 10.17, we then get $\|P_k g - \tilde{P}_k g\|_2 \leq \frac{\delta}{\gamma - \delta}\|g\|_2$, which finishes the proof. □

Finally, we show that repeated applications of $\tilde{L}$ to $f_{\text{nn}} - f$ have also bounded norms:

Lemma 29 (Bounding norms of applications of $\tilde{L}$)**.** *The functions f_{nn} and f satisfy:*

1. $\|\tilde{L}^n(f_{\text{nn}} - f_{\text{span}})\|_2 \leq (n!)^2 \cdot C^n(\epsilon_{\text{span}} + \epsilon_{\text{nn}})$
2. $\|\tilde{L}^n(f_{\text{nn}} - \tilde{f}_{\text{span}})\|_2 \leq (n!)^2 \cdot C^n(\epsilon_{\text{span}} + \epsilon_{\text{nn}}) + 4\left(1 + \frac{\delta}{\gamma - \delta}\right)\lambda_k^n \|f\|_2$

where $\delta = \max\left\{\frac{\epsilon_A}{m}, \frac{\epsilon_c}{\zeta}\right\}$.

Proof. For Part 1, by Lemma 33 we have that

$$\|\tilde{L}^n(f_{\text{nn}} - f_{\text{span}})\|_2 \leq (n!)^2 \cdot C^n \max_{\alpha:|\alpha|\leq n+2} \|\partial^\alpha(f_{\text{nn}} - f_{\text{span}})\|_2 \tag{10.18}$$

From Assumptions (i)-(iii), for any multi-index α we have:

$$\begin{aligned}\|\partial^\alpha f_{\text{nn}} - \partial^\alpha f_{\text{span}}\|_2 &\leq \|\partial^\alpha f_{\text{nn}} - \partial^\alpha f\|_2 + \|\partial^\alpha f - \partial^\alpha f_{\text{span}}\|_2 \\ &\leq \epsilon_{\text{nn}} + \epsilon_{\text{span}}\end{aligned} \tag{10.19}$$

Combining Equation 10.18 and Equation 10.19 we get the result for Part 1.

For Part 2 we have,

$$\|\tilde{L}^n(\tilde{f}_{\text{span}} - f_{\text{nn}})\|_2 = \|\tilde{L}^n(\tilde{f}_{\text{span}} - f_{\text{span}} + f_{\text{span}} - f_{\text{nn}})\|_2 \tag{10.20}$$

$$\leq \|\tilde{L}^n(\tilde{f}_{\text{span}} - f_{\text{span}})\|_2 + \|\tilde{L}^n(f_{\text{span}} - f_{\text{nn}})\|_2 \tag{10.21}$$

Note that from Lemma 5 part (2.) we have that $\|L^{-1}\tilde{L} - I\| \leq \delta$ (where $\|\cdot\|$ denotes the operator norm). This implies that there exists an operator Σ, such that $\|\Sigma\| \leq \delta$ and we can express $\tilde{L}$ as:

$$\tilde{L} = L(I + \Sigma)$$

We will show that there exists a $\tilde{\Sigma}$, s.t. $\|\tilde{\Sigma}\| \leq n2\delta$ and $\tilde{L}^n = (I + \tilde{\Sigma})L^n$. Towards that, we will denote $L^{-n} := \underbrace{L^{-1} \circ L^{-1} \circ \cdots L^{-1}}_{\text{n times}}$ and show that

$$\left\|L^{-n}\tilde{L}^n\right\| \leq 1 + n2\delta \tag{10.22}$$

We have:

$$
\begin{aligned}
\left\|L^{-n}\tilde{L}^n\right\| &= \left\|L^{-n}(L(I+\Sigma))^n\right\| \\
&= \left\|L^{-n}\left(L^n + \sum_{j=1}^{n} L^{j-1}\circ(L\circ\Sigma)\circ L^{n-j} + \cdots + (L\circ\Sigma)^n\right)\right\| \\
&= \left\|I + \sum_{j=1}^{n} L^{-n}\circ L^{j-1}\circ\Sigma\circ L^{n-j} + \cdots + L^{-n}\circ(L\circ\Sigma)^{(n)}\right\| \\
&\overset{(1)}{\leq} 1 + \left\|\sum_{j=1}^{n} L^{-n}\circ L^{j-1}\circ\Sigma\circ L^{n-j}\right\| + \cdots + \|L^{-n}\circ(L\circ\Sigma)^n\| \\
&\overset{(2)}{\leq} 1 + \sum_{i=1}^{n}\binom{n}{i}\delta^i \\
&= (1+\delta)^n \\
&\overset{(3)}{\leq} e^{n\delta} \\
&\leq 1 + 2n\delta
\end{aligned}
$$

where (1) follows from triangle inequality, (2) follows from Lemma 34, (3) follows from $1+x \leq e^x$, and the last part follows from $n\delta \leq 1/10$ and Taylor expanding e^x. Next, since L and $\tilde{L}$ are elliptic operators, we have $\|L^{-n}\tilde{L}^n\| = \|\tilde{L}^n L^{-n}\|$. From this, it immediately follows that there exists a $\tilde{\Sigma}$, s.t. $\tilde{L}^n = (I+\tilde{\Sigma})L^n$ with $\|\tilde{\Sigma}\| \leq n2\delta$.

Plugging this into the first term of Equation 10.21, we have

$$
\begin{aligned}
\|\tilde{L}^n(\tilde{f}_{\text{span}} - f_{\text{span}})\|_2 &= \|\tilde{L}^n\tilde{f}_{\text{span}} - \tilde{L}^n f_{\text{span}}\|_2 \\
&= \|\tilde{L}^n\tilde{f}_{\text{span}} - (I+\tilde{\Sigma})L^n f_{\text{span}}\|_2 \\
&\leq \|\tilde{L}^n\tilde{f}_{\text{span}} - L^n f_{\text{span}}\|_2 + \|\tilde{\Sigma}L^n f_{\text{span}}\|_2 \\
&\leq \|\tilde{L}^n\tilde{f}_{\text{span}} - L^n f_{\text{span}}\|_2 + \|\tilde{\Sigma}\|\|L^n f_{\text{span}}\|_2 \\
&\leq \|\tilde{L}^n\tilde{f}_{\text{span}} - L^n f_{\text{span}}\|_2 + n2\delta\lambda_k^n\|f_{\text{span}}\|_2 \qquad (10.23)
\end{aligned}
$$

The first term in first term in Equation 10.23 can be expanded as follows:

$$
\begin{aligned}
\|\tilde{L}^n\tilde{f}_{\text{span}} - L^n f_{\text{span}}\|_2 &= \|\tilde{L}^n\tilde{f}_{\text{span}} - L^n\tilde{f}_{\text{span}} + L^n\tilde{f}_{\text{span}} + L^n f_{\text{span}}\|_2 \\
&\leq \|\tilde{L}^n\tilde{f}_{\text{span}} - L^n\tilde{f}_{\text{span}}\| + \|L^n\tilde{f}_{\text{span}} - L^n f_{\text{span}}\|_2 \qquad (10.24)
\end{aligned}
$$

We'll consider the two terms in turn.

For the first term, the same proof as that of Equation 10.22 shows that there exists an operator $\hat{\Sigma}$, s.t. $\|\hat{\Sigma}\| \leq 2n\delta$ and $L^n = (I + \hat{\Sigma})\tilde{L}^n$. Hence, we have:

$$\begin{aligned}
\|\tilde{L}^n \tilde{f}_{\text{span}} - L^n \tilde{f}_{\text{span}}\| &= \|\tilde{L}^n \tilde{f}_{\text{span}} - (I + \hat{\Sigma})\tilde{L}^n \tilde{f}_{\text{span}}\| \\
&= \|\hat{\Sigma}\tilde{L}^n \tilde{f}_{\text{span}}\| \\
&\leq \tilde{\lambda}_k^n \|\hat{\Sigma}\| \|\tilde{f}_{\text{span}}\|_2 \\
&\leq 2n\delta\tilde{\lambda}_k^n \|f\|_2 \qquad (\because \|\tilde{f}_{\text{span}}\|_2 \leq \|f\|_2)
\end{aligned} \tag{10.25}$$

For the second term in Equation 10.23 we have:

$$\|L^n(\tilde{f}_{\text{span}} - f_{\text{span}})\|_2 \leq \sup_{v: v = v_1 - v_2, v_1 \in \Phi_k, v_2 \in \tilde{\Phi}_k} \frac{\|L^n v\|_2}{\|v\|_2} \|\tilde{f}_{\text{span}} - f_{\text{span}}\|_2 \tag{10.26}$$

To bound the first factor we have:

$$\begin{aligned}
\|L^n v\|_2 &= \|L^n(v_1 - v_2)\|_2 \\
&\leq \|L^n v_1\|_2 + \|L^n v_2\|_2 \\
&= \|L^n v_1\|_2 + \|(I + \hat{\Sigma})\tilde{L}^n v_2\|_2 \\
&\leq \lambda_k^n \|v_1\|_2 + \tilde{\lambda}_k^n \|I + \hat{\Sigma}\|_2 \|v_2\|_2 \\
&\leq (\lambda_k^n + \tilde{\lambda}_k^n(1 + 2n\delta))\|v\|_2
\end{aligned}$$

where we use the fact that $\|v_1\|_2, \|v_2\|_2 \leq \|v\|_2$ and $\|\hat{\Sigma}\| \leq 2n\delta$. Hence, we can bound

$$\sup_{v: v = v_1 - v_2, v_1 \in \Phi_k, v_2 \in \tilde{\Phi}_k} \frac{\|L^n v\|_2}{\|v\|_2} \leq (\lambda_k^n + \tilde{\lambda}_k^n(1 + 2n\delta)) \tag{10.27}$$

From Equation 10.27 and Lemma 4 we have:

$$\begin{aligned}
\|L^n(\tilde{f}_{\text{span}} - f_{\text{span}})\|_2 &\leq \sup_{v: v = v_1 - v_2, v_1 \in \Phi_k, v_2 \in \tilde{\Phi}_k} \frac{\|L^n v\|_2}{\|v\|_2} \|\tilde{f}_{\text{span}} - f_{\text{span}}\|_2 \\
&\leq (\lambda_k^n + \tilde{\lambda}_k^n(1 + 2n\delta)) \frac{\delta}{\gamma - \delta} \|f\|_2
\end{aligned} \tag{10.28}$$

Therefore from Equation 10.25 and Equation 10.28, we can upper bound $\|\tilde{L}^n(\tilde{f}_{\text{span}} - f_{\text{span}})\|_2$ using Equation 10.23 as follows:

$$\begin{aligned}
\|\tilde{L}^n(\tilde{f}_{\text{span}} - f_{\text{span}})\|_2 &\leq \|\tilde{L}^n \tilde{f}_{\text{span}} - L^n f_{\text{span}}\|_2 + 2n\delta\lambda_k^n \|f\|_2 \\
&\leq 2n\delta\tilde{\lambda}_k^n \|f\|_2 + (\lambda_k^n + \tilde{\lambda}_k^n(1 + 2n\delta)) \frac{\delta}{\gamma - \delta} \|f\|_2 + 2n\delta\lambda_k^n \|f\|_2 \\
&\overset{(i)}{\leq} (1 + 2n\delta\lambda_k)(1 + (1 + 2n\delta)) \frac{\delta}{\gamma - \delta} \lambda_k^n \|f\|_2 + 2n\delta\lambda_k^n \|f\|_2 \\
&\overset{(ii)}{\leq} 4\left(1 + \frac{\delta}{\gamma - \delta}\right) \lambda_k^n \|f\|_2
\end{aligned}$$

Here in (i) we use the result from Lemma 30 and write $\frac{\tilde{\lambda}_k^n}{\lambda_k^n} \leq 1 + 2n\delta\lambda_k$. In (iii), we use $n \leq T$ and the fact that $2T \min(1, \lambda_k)\delta \leq 1/10 \leq 1$

Therefore, finally we have:

$$\|\tilde{L}^n \tilde{f}_{\text{span}} - L^n f_{\text{span}}\|_2 \leq 4\left(\frac{\delta}{\gamma - \delta} + 1\right)\lambda_k^n \|f\|_2$$

Combining with the result for Part 1, Therefore we have the following:

$$\|\tilde{L}^n(\tilde{f}_{\text{span}} - f_{\text{nn}})\|_2 \leq (n!)^2 \cdot C^n(\epsilon_{\text{span}} + \epsilon_{\text{nn}}) + 4\left(1 + \frac{\delta}{\gamma - \delta}\right)\lambda_k^n \|f\|_2$$

□

10.4 Technical Lemmas: Manipulating Operators

Before we state the lemmas we introduce some common notation used throughout this section. We denote $L^n = \underbrace{L \circ L \circ \cdots \circ L}_{\text{n times}}$. Further we use L_k to denote the operator with $\partial_k a_{ij}$ for all $i, j \in [d]$ and $\partial_k c$ as coefficients, that is:

$$L_k u = \sum_{i,j=1}^{d} -(\partial_k a_{ij})\partial_{ij} u - \sum_{i,j=1}^{d} \partial_k(\partial_i a_i)\partial_j u + (\partial_k c)u$$

Similarly the operator L_{kl} is defined as:

$$L_{kl} u = \sum_{i,j=1}^{d} -(\partial_{kl} a_{ij})\partial_{ij} u - \sum_{i,j=1}^{d} \partial_{kl}(\partial_i a_i)\partial_j u + (\partial_{kl} c)u$$

Lemma 30. *Given φ_i and $\tilde{\varphi}_i$ for all $i \in [k]$ are top k eigenvalues of operators L and $\tilde{L}$ respectively, such that $\|L^{-1} - \tilde{L}^{-1}\|$ is bounded. Then for all $n \in \mathbb{N}$ we have that*

$$\tilde{\lambda}_i^n \leq (1 + \hat{e})\lambda_i^n$$

where $i \in [k]$ and $|\hat{e}| \leq 2n\delta\lambda_k$ and $\delta = \max\left\{\frac{\epsilon_A}{m}, \frac{\epsilon_c}{\zeta}\right\}$.

Proof. From Equation 10.15 and Weyl's inequality we have for all $i \in \mathbb{N}$

$$\sup_i \left|\frac{1}{\lambda_i} - \frac{1}{\tilde{\lambda}_i}\right| \leq \|L^{-1} - \tilde{L}^{-1}\| \leq \delta$$

From this, we can conclude that:

$$\begin{aligned}
&\left|\tilde{\lambda}_i - \lambda_i\right| \leq \delta\lambda_i\tilde{\lambda}_i \\
\implies &\tilde{\lambda}_i(1 - \delta\lambda_i) \leq \lambda_i \\
\implies &\tilde{\lambda}_i \leq \frac{\lambda_i}{(1 - \delta\lambda_i)} \\
\implies &\tilde{\lambda}_i \leq (1 + \delta\lambda_i)\lambda_i
\end{aligned}$$

Writing $\tilde{\lambda}_i = (1+\tilde{e}_i)\lambda_i$ (where $\tilde{e}_i = \delta\lambda_i$), we have

$$\begin{aligned}\left|\tilde{\lambda}_i^n - \lambda_i^n\right| &= |((1+\tilde{e}_i)\lambda_i)^n - \lambda_i^n| \\ &= |\lambda_i^n((1+\tilde{e}_i)^n - 1)| \\ &\overset{(1)}{\leq} \lambda_i^n |\tilde{e}|_i \left|\sum_{j=1}^{n}(1+\tilde{e}_i)^j\right| \\ &\overset{(2)}{\leq} \lambda_i^n n|\tilde{e}_i| e^{n|\tilde{e}_i|} \\ &\overset{(3)}{\leq} \lambda_i^n n|\tilde{e}_i|(1+|2n\tilde{e}_i|) \\ &\leq 2\lambda_i^n n|\tilde{e}_i|\end{aligned}$$

where (1) follows from the factorization $a^n - b^n = (a-b)(\sum_{i=0}^{n-1} a^i b^{n-i-i})$, (2) follows from $1+x \leq e^x$, and (3) follows from $n|\tilde{e}_i| \leq 1/20$ and Taylor expanding e^x. Hence, there exists a $\hat{e}_i$, s.t. $\tilde{\lambda}_i^n = (1+\hat{e}_i)\lambda_i^n$ and $|\hat{e}_i| \leq 2n|\tilde{e}_i|$ (i.e., $|\hat{e}_i| \leq 2n\delta\lambda_i$). Using the fact that $\lambda_i \leq \lambda_k$ for all $i \in [k]$ completes the proof. □

Lemma 31 (Operator Chain Rule)**.** *Given an elliptic operator L, for all $v \in C^\infty(\Omega)$ we have the following*

$$\nabla_k L^n u = \sum_{i=1}^{n} \big(L^{n-i} \circ L_k \circ L^{i-1}\big)(u) + L^n(\nabla_k u) \tag{10.29}$$

$$\begin{aligned}\nabla_{kl}(L^n u) = &\sum_{\substack{i,j \\ i<j}} \big(L^{n-i} \circ L_k \circ L^{j-i-1} \circ L_l \circ L^{j-1}\big)u \\ &+ \sum_{\substack{i,j \\ i>j}} \big(L^{n-j} \circ L_k \circ L^{i-j-1} \circ L_l \circ L^{i-1}\big)u \\ &+ \sum_{i} \big(L^{n-i} \circ L_{kl} \circ L^{i-1}\big)u + L^n(\nabla_{kl} u)\end{aligned} \tag{10.30}$$

where we assume that $L^{(0)} = I$.

Proof. We show the proof using induction on n. To handle the base case, for $n = 1$, we have

$$\begin{aligned}\nabla_k(Lu) &= \nabla_k(-\mathrm{div}_{\mathrm{x}}(A\nabla u) + cu) \\ &= \nabla_k\left(-\sum_{ij} a_{ij}\partial_{ij}u - \sum_{ij}\partial_i a_{ij}\partial_j u + cu\right) \\ &= \left(-\sum_{ij} a_{ij}\partial_{ij}(\partial_k u) - \sum_{ij}\partial_i a_{ij}\partial_j\partial_k u + c\partial_k u\right) \\ &+ \left(-\sum_{ij}\partial_k a_{ij}\partial_{ij}u - \sum_{ij}\partial_i\partial_k a_{ij}\partial_j u + \partial_k cu\right) \\ &= L(\nabla_k u) + L_k u\end{aligned} \tag{10.31}$$

Similarly $n = 1$ and $k, l \in [d]$,

$$\begin{aligned}
\nabla_{kl}(Lu) &= \nabla_{kl}(-\mathrm{div}_{\mathrm{x}}(A\nabla u) + cu) \\
&= \nabla_{kl}\left(-\sum_{ij} a_{ij}\partial_{ij}u - \sum_{ij}\partial_i a_{ij}\partial_j u + cu\right) \\
&= \left(-\sum_{ij} a_{ij}\partial_{ij}(\partial_{kl}u) - \sum_{ij}\partial_i a_{ij}\partial_j\partial_{kl}u + c\partial_{kl}u\right) \\
&+ \left(-\sum_{ij}\partial_k a_{ij}\partial_{ij}\partial_l u - \sum_{ij}\partial_i\partial_k a_{ij}\partial_j\partial_l u + \partial_k c\partial_l u\right) \\
&+ \left(-\sum_{ij}\partial_l a_{ij}\partial_{ij}\partial_k u - \sum_{ij}\partial_i\partial_l a_{ij}\partial_j\partial_k u + \partial_l c\partial_k u\right) \\
&+ \left(-\sum_{ij}\partial_{kl} a_{ij}\partial_{ij}u - \sum_{ij}\partial_i\partial_{kl} a_{ij}\partial_j u + \partial_{kl} cu\right) \\
&= L(\nabla_{kl}u) + L_k(\nabla_l u) + L_l(\nabla_k u) + L_{kl}u
\end{aligned} \tag{10.32}$$

For the inductive case, assume that for all $m < n$, Equation 10.29 and Equation 10.30 hold. Then, for any $k \in [d]$ we have:

$$\begin{aligned}
\nabla_k(L^n u) &= \nabla_k\big(L \circ L^{n-1}(u)\big) \\
&= L\big(\nabla_k(L^{n-1}u)\big) + L_k\big(L^{n-1}u\big) \\
&= L\left(\sum_{i=1}^{n-1}\big(L^{n-1-i}\circ L_k \circ L^{i-1}\big)u + L^{n-1}(\nabla_k u)\right) + L_k\big(L^{n-1}\big)u \\
&= \sum_{i=1}^{n}\big(L^{n-i}\circ L_k\circ L^{i-1}\big)(u) + L^n(\nabla_k u)
\end{aligned} \tag{10.33}$$

Similarly, for all $k, l \in [d]$ we have:

$$
\begin{aligned}
\nabla_{kl}(L^n u) &= \nabla_{kl}\big(L \circ L^{n-1}(u)\big) \\
&= L\big(\nabla_{kl}(L^{n-1}u)\big) + L_k\big(\nabla_l\big(L^{n-1}u\big)\big) + L_l\big(\nabla_k\big(L^{n-1}u\big)\big) + L_{kl}\big(L^{n-1}u\big) \\
&= L\Bigg(\sum_{\substack{i,j \\ i<j}}^{n-1}\big(L^{n-1-i}\circ L_k \circ L^{j-i-1}\circ L_l \circ L^{j-1}\big)u \\
&\qquad + \sum_{\substack{i,j \\ i>j}}^{n-1}\big(L^{n-1-j}\circ L_k \circ L^{i-j-1}\circ L_l \circ L^{i-1}\big)u \\
&\qquad + \sum_{i=1}^{n-1}\big(L^{n-1-i}\circ L_{kl}\circ L^{i-1}\big)u + L^{n-1}(\nabla_{kl}u)\Bigg) \\
&\quad + L_k\Bigg(\sum_{i=1}^{n-1}\big(L^{n-1-i}\circ L_l \circ L^{i-1}\big)(u) + L^{n-1}(\nabla_l u)\Bigg) \qquad \text{(from Equation 10.33)} \\
&\quad + L_l\Bigg(\sum_{i=1}^{n-1}\big(L^{n-1-i}\circ L_k \circ L^{i-1}\big)(u) + L^{n-1}(\nabla_k u)\Bigg) \qquad \text{(from Equation 10.33)} \\
&\quad + L_{kl}\big(L^{n-1}u\big) \\
&= \sum_{\substack{i,j \\ i<j}}^{n}\big(L^{n-i}\circ L_k \circ L^{j-i-1}\circ L_l \circ L^{j-1}\big)u \\
&\qquad + \sum_{\substack{i,j \\ i>j}}^{n}\big(L^{n-j}\circ L_k \circ L^{i-j-1}\circ L_l \circ L^{i-1}\big)u \\
&\qquad + \sum_{i}^{n}\big(L^{n-i}\circ L_{kl}\circ L^{i-1}\big)u + L^{n}(\nabla_{kl}u) \qquad (10.34)
\end{aligned}
$$

By induction, the claim follows. □

Lemma 32. *For all $u \in C^\infty(\Omega)$ then for all $k, l \in [d]$ the following upper bounds hold,*

$$\|Lu\|_2 \leq C \max_{\alpha:|\alpha|\leq 2} \|\partial^\alpha u\|_2 \qquad (10.35)$$

$$\|\nabla_k(Lu)\|_2 \leq 2 \cdot C \max_{\alpha:|\alpha|\leq 3} \|\partial^\alpha u\|_2 \qquad (10.36)$$

and

$$\|\nabla_{kl}(Lu)\|_2 \leq 4 \cdot C \max_{\alpha:|\alpha|\leq 4} \|\partial^\alpha u\|_2 \qquad (10.37)$$

where

$$C := (2d^2+1)\max\Bigg\{\max_{\alpha:|\alpha|\leq 3}\max_{i,j}\|\partial^\alpha a_{ij}\|_{L^\infty(\Omega)}, \max_{\alpha:|\alpha|\leq 2}\|\partial^\alpha c\|_{L^\infty(\Omega)}\Bigg\}.$$

Proof. We first show the upper bound on $\|Lu\|_2$:

$$\begin{aligned}
\|Lu\|_2 &\leq \left\| -\sum_{i,j=1}^{d} a_{ij}\partial_{ij}u - \sum_{i,j=1}^{d} \partial_i a_{ij}\partial_j u + cu \right\|_2 \\
&\leq^{(1)} \underbrace{(2d^2+1)\max\left\{ \max_{i,j} \|\partial_i a_{ij}\|_{L^\infty(\Omega)}, \max_{i,j} \|a_{ij}\|_{L^\infty(\Omega)}, \|c\|_{L^\infty(\Omega)} \right\}}_{C_1} \max_{\alpha:|\alpha|\leq 2} \|\partial^\alpha u\|_2 \\
&\leq C_1 \max_{\alpha:|\alpha|\leq 2} \|\partial^\alpha u\|_2 \qquad (10.38)
\end{aligned}$$

where (1) follows by Hölder.

Proceeding to $\|\nabla_k(Lu)\|_2$, from Lemma 33 we have

$$\begin{aligned}
\|\nabla_k(Lu)\|_2 &\leq \|L_k u\|_2 + \|L(\nabla_k u)\|_2 \\
&\leq \left\| -\sum_{i,j=1}^{d} \partial_k a_{ij}\partial_{ij}u - \sum_{i,j=1}^{d} \partial_{ik} a_{ij}\partial_j u + \partial_k cu \right\|_2 \\
&+ \left\| -\sum_{i,j=1}^{d} a_{ij}\partial_{ijk}u - \sum_{i,j=1}^{d} \partial_i a_{ij}\partial_{jk} u + c\partial_k u \right\|_2 \\
&\leq (2d^2+1)\max\left\{ \max_{\alpha:|\alpha|\leq 2} \max_{i,j} \|\partial^\alpha a_{ij}\|_{L^\infty(\Omega)}, \|\partial_k c\|_{L^\infty(\Omega)} \right\} \max_{\alpha:|\alpha|\leq 2} \|\partial^\alpha u\|_2 \\
&+ (2d^2+1)\max\left\{ \max_{\alpha:|\alpha|\leq 1} \max_{i,j} \|\partial^\alpha a_{ij}\|_{L^\infty(\Omega)}, \|c\|_{L^\infty(\Omega)} \right\} \max_{\alpha:|\alpha|\leq 3} \|\partial^\alpha u\|_2 \\
\implies \|\nabla_k(Lu)\|_2 &\leq 2 \cdot \underbrace{(2d^2+1)\max\left\{ \max_{\alpha:|\alpha|\leq 2} \max_{i,j} \|\partial^\alpha a_{ij}\|_{L^\infty(\Omega)}, \max_{\alpha:|\alpha|\leq 1} \|\partial^\alpha c\|_{L^\infty(\Omega)} \right\}}_{C_2} \max_{\alpha:|\alpha|\leq 3} \|\partial^\alpha u\|_2 \\
&\leq 2 \cdot C_2 \max_{\alpha:|\alpha|\leq 3} \|\partial^\alpha u\|_2 \qquad (10.39)
\end{aligned}$$

We use the result from Lemma 31 (equation Equation 10.32), to upper bound the quantity $\|\nabla_{kl}(Lu)\|_2$

$$\begin{aligned}
\|\nabla_{kl}(Lu)\|_2 &\leq \|L_{kl}u\|_2 + \|L_k(\nabla_l u)\|_2 + \|L_l(\nabla_k u)\|_2 + \|L(\nabla_{kl}u)\|_2 \\
&\leq \left\| -\sum_{i,j=1}^{d} \partial_{kl}a_{ij}\partial_{ij}u - \sum_{i,j=1}^{d} \partial_{ikl}a_{ij}\partial_j u + \partial_{kl}cu \right\|_2 \\
&+ \left\| -\sum_{i,j=1}^{d} \partial_k a_{ij}\partial_{ij}\partial_l u - \sum_{i,j=1}^{d} \partial_i\partial_k a_{ij}\partial_j\partial_l u + \partial_k c\partial_l u \right\|_2 \\
&+ \left\| -\sum_{i,j=1}^{d} \partial_l a_{ij}\partial_{ij}\partial_k u - \sum_{i,j=1}^{d} \partial_i\partial_l a_{ij}\partial_j\partial_k u + \partial_l c\partial_k u \right\|_2 \\
&+ \left\| -\sum_{i,j=1}^{d} a_{ij}\partial_{ijkl}u - \sum_{i,j=1}^{d} \partial_i a_{ij}\partial_{jkl}u + c\partial_{kl}u \right\|_2 \\
&\leq (2d^2+1)\max\left\{ \max_{\alpha:|\alpha|\leq 3} \max_{i,j} \|\partial^\alpha a_{ij}\|_{L^\infty(\Omega)}, \|\partial_{kl}c\|_{L^\infty(\Omega)} \right\} \max_{\alpha:|\alpha|\leq 2} \|\partial^\alpha u\|_2 \\
&+ 2(2d^2+1)\max\left\{ \max_{\alpha:|\alpha|\leq 2} \max_{i,j} \|\partial^\alpha a_{ij}\|_{L^\infty(\Omega)}, \|c\|_{L^\infty(\Omega)} \right\} \max_{\alpha:|\alpha|\leq 3} \|\partial^\alpha u\|_2 \\
&+ (2d^2+1)\max\left\{ \max_{\alpha:|\alpha|\leq 2} \max_{i,j} \|\partial^\alpha a_{ij}\|_{L^\infty(\Omega)}, \|c\|_{L^\infty(\Omega)} \right\} \max_{\alpha:|\alpha|\leq 4} \|\partial^\alpha u\|_2 \\
\implies \|\nabla_{kl}(Lu)\|_2 &\leq 4 \cdot \underbrace{(2d^2+1)\max\left\{ \max_{\alpha:|\alpha|\leq 3} \max_{i,j} \|\partial^\alpha a_{ij}\|_{L^\infty(\Omega)}, \max_{\alpha:|\alpha|\leq 2} \|\partial^\alpha c\|_{L^\infty(\Omega)} \right\}}_{C_3} \max_{\alpha:|\alpha|\leq 4} \|\partial^\alpha u\|_2 \\
&\leq 4 \cdot C_3 \max_{\alpha:|\alpha|\leq 4} \|\partial^\alpha u\|_2 \qquad (10.40)
\end{aligned}$$

Since $C_1 \leq C_2 \leq C_3$, we define $C := C_3$ and therefore from equations Equation 10.38, Equation 10.39 and Equation 10.40 the claim follows.

Further, we note that from Equation 10.39, we also have that

$$\|L_k(u)\|_2, \|L(\nabla_k u)\|_2 \leq C \max_{\alpha:|\alpha|\leq 3} \|\partial^\alpha u\|_2 \qquad (10.41)$$

and similarly from Equation 10.40 we have that,

$$\|L_{kl}(u)\|_2, \|L_k(\nabla_l u)\|_2, \|L_l(\nabla_k u)\|_2, \|L(\nabla_{kl}u)\|_2 \leq C \max_{\alpha:|\alpha|\leq 4} \|\partial^\alpha u\|_2 \qquad (10.42)$$

□

Lemma 33. *For all $u \in C^\infty(\Omega)$ and $k, l \in [d]$ then for all $n \in \mathbb{N}$ we have the following upper bounds,*

$$\|L^n u\|_2 \leq (n!)^2 \cdot C^n \max_{\alpha:|\alpha|\leq n+2} \|\partial^\alpha u\|_2 \qquad (10.43)$$

$$\|\nabla_k(L^n u)\|_2 \leq (n+1)\cdot(n!)^2 \cdot C^n \max_{\alpha:|\alpha|\leq n+2} \|\partial^\alpha u\|_2 \qquad (10.44)$$

$$\|\nabla_{kl}(L^n u)\|_2 \le ((n+1)!)^2 \cdot C^n \max_{\alpha:|\alpha|\le n+3} \|\partial^\alpha u\|_2 \tag{10.45}$$

where $C = (2d^2+1)\max\{\max_{\alpha:|\alpha|\le 3}\max_{i,j}\|\partial^\alpha a_{ij}\|_{L^\infty(\Omega)}, \max_{\alpha:|\alpha|\le 2}\|\partial^\alpha c\|_{L^\infty(\Omega)}\}$.

Proof. We prove the Lemma by induction on n. The base case $n = 1$ follows from Lemma 32, along with the fact that $\max_{\alpha:|\alpha|\le 2}\|\partial^\alpha u\|_2 \le \max_{\alpha:|\alpha|\le 3}\|\partial^\alpha u\|_2$.

To show the inductive case, assume that the claim holds for all $m \le (n-1)$. By Lemma 32, we have

$$\begin{aligned}
\|L^n u\|_2 &= \|L(L^{n-1}u)\|_2 \\
&\le \left\| -\sum_{i,j=1}^d a_{ij}\partial_{ij}(L^{n-1}u) - \sum_{i,j=1}^d \partial_i a_{ij}\partial_j(L^{n-1}u) + c(L^{n-1}u)\right\|_2 \\
&\le C\cdot\max\left\{\|L^{n-1}u\|_2, \max_i\|\nabla_i(L^{n-1}u)\|_2, \max_{i,j}\|\nabla_{ij}(L^{n-1}u)\|_2\right\} \\
&\le C\cdot(n!)^2\cdot C^{n-1}\max_{\alpha:|\alpha|\le(n-1)+3}\|\partial^\alpha u\|_2
\end{aligned}$$

Thus, we have

$$\|L^n u\|_2 \le (n!)^2\cdot C^n \max_{\alpha:|\alpha|\le n+2}\|\partial^\alpha u\|_2$$

as we need.

Similarly, for $k \in [d]$, we have:

$$\begin{aligned}
\|\nabla_k(L^n u)\|_2 &\le \sum_{i=1}^n \left\|\left(L^{n-i}\circ L_k\circ L^{i-1}\right)(u)\right\|_2 + \|L^n(\nabla_k u)\|_2 \\
&\le (n)\cdot(n!)^2\cdot C^n\max_{\alpha:|\alpha|\le n+2}\|\partial^\alpha u\|_2 + (n!)^2\cdot C^n\max_{\alpha:|\alpha|\le n+2}\|\partial^\alpha u\|_2 \\
&\le (n+1)\cdot(n!)^2\cdot C^n\max_{\alpha:|\alpha|\le n+2}\|\partial^\alpha u\|_2
\end{aligned} \tag{10.46}$$

Finally, for $k, l \in [d]$ we have

$$\begin{aligned}
\|\nabla_{kl}(L^n u)\|_2 &\leq \sum_{\substack{i,j \\ i<j}} \left\| \left(L^{n-i} \circ L_k \circ L^{j-i-1} \circ L_l \circ L^{j-1}\right) u \right\|_2 \\
&\quad + \sum_{\substack{i,j \\ i>j}} \left\| \left(L^{n-j} \circ L_k \circ L^{i-j-1} \circ L_l \circ L^{i-1}\right) u \right\|_2 \\
&\quad + \sum_{i} \left\| \left(L^{n-i} \circ L_{kl} \circ L^{i-1}\right) u \right\|_2 + \|L^n(\nabla_{kl} u)\|_2 \\
&\leq n(n+1) \cdot (n!)^2 \cdot C^n \max_{\alpha:|\alpha|\leq n+2} \|\partial^\alpha u\|_2 \\
&\quad + n \cdot (n!)^2 \cdot C^n \max_{\alpha:|\alpha|\leq n+2} \|\partial^\alpha u\|_2 + C^n \max_{\alpha:|\alpha|\leq n+3} \|\partial^\alpha u\|_2 \\
\implies \|\nabla_{kl}(L^n u)\|_2 &\leq ((n+1)!)^2 \cdot C^n \max_{\alpha:|\alpha|\leq n+3} \|\partial^\alpha u\|_2
\end{aligned} \tag{10.47}$$

Thus, the claim follows. □

Lemma 34. *Let A_n^i, $i \in [n]$ be defined as a composition of $(n-i)$ applications of L and i applications of $L \circ \Sigma$ (in any order), s.t. $\|\Sigma\| \leq \delta$. Then, we have:*

$$\|L^{-n} A_n^i\| \leq \delta^i \tag{10.48}$$

Proof. We prove the above claim by induction on n.

For $n = 1$ we have two cases. If $A^{(1)} = L \circ \Sigma$, we have:

$$\|L^{-1} \circ L \circ \Sigma\| \leq \delta$$

If $A^{(1)} = L$ we have:

$$\|L^{-1} L\| = 1$$

Towards the inductive hypothesis, assume that for $m \leq n-1$ and $i \in [n-1]$ it holds that,

$$\|L^{n-1} A_{n-1}^i\| \leq \delta^i$$

For n, we will have two cases. First, if $A_n^{i+1} = A_{n-1}^i \circ L \circ \Sigma$, by submultiplicativity of the operator norm, as well as the fact that similar operators have identical spectra (hence equal operator norm) we have:

$$\begin{aligned}
\|L^{-n} \circ A_n^{i+1}\| &= \|L^{-1} \circ L^{-(n-1)} \circ A_{n-1}^{(i)} \circ L \circ \Sigma\| \\
&= \|L^{-(n-1)} \circ A_{n-1}^i \circ L \circ \Sigma \circ L^{-1}\| \\
&\leq \delta \|L^{-(n-1)} A_{(n-1)}^{i-1}\| \|L \circ \Sigma \circ L^{-1}\| \\
&\leq \delta^i \delta = \delta^{i+1}
\end{aligned}$$

so the inductive claim is proved. In the second case, $A_n^i = A_{n-1}^i L$ and we have, by using the fact that the similar operators have identical spectra:

$$\begin{aligned} \|L^{-n} \circ A_n^i \circ L\| &= \|L^{-(n-1)} \circ A_{n-1}^i \circ L \circ L^{-1}\| \\ &= \|L^{-(n-1)} \circ A_{n-1}^i\| \leq \delta^i \end{aligned}$$

where the last inequality follows by the inductive hypothesis. □

11 Appendix for Chapter 4

11.1 Proofs from Section 4.6.1: Convergence Rate of Sequence

11.1.1 Proof of Lemma 17

Proof. In order to prove part 1, we will use the following integration by parts identity, for functions $r : \Omega \to \mathbb{R}$ such that and $s : \Omega \to \mathbb{R}$, and $r, s \in H_0^1(\Omega)$,

$$\int_\Omega \frac{\partial r}{\partial x_i} s dx = -\int_\Omega r \frac{\partial s}{\partial x_i} dx + \int_{\partial\Omega} r s n_i d\Gamma \tag{11.1}$$

where n_i is a normal at the boundary and $d\Gamma$ is an infinitesimal element of the boundary $\partial\Omega$.

Using the formula in Equation 11.1 for functions $u, v \in H_0^1(\Omega)$, we have

$$\begin{aligned}
\langle D\mathcal{E}(u), v\rangle_{L^2(\Omega)} &= \langle -\nabla_x \cdot \partial_{\nabla u} L(x, u, \nabla u) + \partial_u L(x, u, \nabla u), v\rangle_{L^2(\Omega)} \\
&= -\int_\Omega \nabla_x \cdot \partial_{\nabla u} L(x, u, \nabla u) v + \partial_u L(x, u, \nabla u) v \, dx \\
&= -\int_\Omega \sum_{i=1}^d \frac{\partial (\partial_{\nabla u} L(x, u, \nabla u))_i}{\partial x_i} v + \partial_u L(x, u, \nabla u) v \, dx \\
&= \int_\Omega \sum_{i=1}^d (\partial_{\nabla u} L(x, u, \nabla u))_i \frac{\partial v}{\partial x_i} dx + \int_\Omega \sum_{i=1}^d (\partial_{\nabla u} L(x, u, \nabla u))_i v n_i dx + \int_\Omega \partial_u L(x, u, \nabla u) v \, dx \\
&= \int \partial_{\nabla u} L(\nabla u) \cdot \nabla v + \partial_u L(x, u, \nabla u) v \, dx
\end{aligned}$$

where in the last equality we use the fact that the function $v \in H_0^1(\Omega)$, thus $v(x) = 0, \forall x \in \partial\Omega$.

To prove part 2. first note from Part 1. we know that $\langle D\mathcal{E}(u) - D\mathcal{E}(v), u - v\rangle_{L^2(\Omega)}$ takes the following form,

$$\begin{aligned}
&\langle D\mathcal{E}(u) - D\mathcal{E}(v), u - v\rangle_{L^2(\Omega)} \\
&= \langle \partial_{\nabla u} L(x, u, \nabla u) - \partial_{\nabla v} L(x, v, \nabla v), \nabla u - \nabla v\rangle_{L^2(\Omega)} + \langle \partial_u L(x, u, \nabla u) - \partial_v L(x, v, \nabla v), u - v\rangle_{L^2(\Omega)}
\end{aligned} \tag{11.2}$$

We know that for $x \in \Omega$, we have

$$\nabla^2_{(u,\nabla u)} L(x, u, \nabla u) \leq \mathrm{diag}([\Lambda, \Lambda \mathbf{1}_d])$$

Note that $\nabla_{(u,\nabla u)} L(x,u,\nabla u)$ is a vector, and we can write, $\partial_{(u,\nabla u)} L(x,u,\nabla u) = [\partial_u L(x,u,\nabla u), \partial_{\nabla u} L(x,u,\nabla u)]$ (here for two vectors a, b we define a new vector $c := [a,b]$ as their concatenation).

Using the smoothness of L can write,

$$\begin{aligned}
&[\partial_u L(x,u,\nabla u) - \partial_u L(x,v,\nabla v), \partial_{\nabla u} L(x,u,\nabla u) - \partial_{\nabla u} L(x,v,\nabla v)]^T ([u-v, \nabla u - \nabla v]) \\
&\leq [u-v, \nabla u - \nabla v]^T (\mathrm{diag}([\Lambda, \Lambda \mathbf{1}_d]))[u-v, \nabla u - \nabla v] \\
&\leq \Lambda [u-v, \nabla u - \nabla v]^T [u-v, \nabla u - \nabla v]
\end{aligned}$$

This implies that for $x \in \Omega$ we have

$$\begin{aligned}
&(\partial_{\nabla u} L(x,u(x),\nabla u(x)) - \partial_{\nabla u} L(x,v(x),\nabla v(x))^T (\nabla u(x) - \nabla v(x)) \\
&\quad + (\partial_u L(x,u(x),\nabla u(x)) - \partial_u L(x,v(x),\nabla v(x))^T (u(x) - v(x)) \\
&\leq \Lambda \|\nabla u(x) - \nabla v(x)\|_2^2 + \Lambda \|u(x) - v(x)\|_2^2
\end{aligned}$$

Integrating over Ω on both sides we get

$$\begin{aligned}
&\langle \partial_{\nabla u} L(x,u,\nabla u) - \partial_{\nabla v} L(x,v,\nabla v), \nabla u - \nabla v \rangle_{L^2(\Omega)} + \langle \partial_u L(x,u,\nabla u) - \partial_v L(x,v,\nabla v), u - v \rangle_{L^2(\Omega)} \\
&\leq \Lambda \|\nabla u - \nabla v\|_{L^2(\Omega)}^2 + \Lambda \|u - v\|_{L^2(\Omega)}^2 \\
&\leq \Lambda (1 + C_p^2) \cdot \|u - v\|_{H_0^1(\Omega)}^2.
\end{aligned}$$

the Poincare inequaltiy from Theorem 14 in the final equation. Hence plugging this result in Equation 11.2 we have,

$$\langle D\mathcal{E}(u) - D\mathcal{E}(v), u - v \rangle_{L^2(\Omega)} \leq (\Lambda + C_p^2 \Lambda) \|u - v\|_{H_0^1(\Omega)}^2$$

This proves the right hand side of the inequality in part 2.

To prove the left and side we use similar to the upper bound, using the convexity of the $L(x,\cdot,\cdot)\colon \mathbb{R} \times \mathbb{R}^d$, we can lower bound the following term,

$$\begin{aligned}
&[\partial_u L(x,u,\nabla u) - \partial_u L(x,v,\nabla v), \partial_{\nabla u} L(x,u,\nabla u) - \partial_{\nabla u} L(x,v,\nabla v)]^T ([u-v, \nabla u - \nabla v]) \\
&\geq [u-v, \nabla u - \nabla v]^T (\mathrm{diag}([0, \lambda \mathbf{1}_d]))[u-v, \nabla u - \nabla v] \\
&\geq \lambda (\nabla u - \nabla v)^T (\nabla u - \nabla v)
\end{aligned}$$

Therefore, for all $x \in \Omega$ we have

$$\begin{aligned}
&(\partial_{\nabla u} L(x,u(x),\nabla u(x)) - \partial_{\nabla u} L(x,v(x),\nabla v(x))^T (\nabla u(x) - \nabla v(x)) \\
&+ (\partial_u L(x,u(x),\nabla u(x)) - \partial_u L(x,v(x),\nabla v(x))^T (u(x) - v(x)) \\
&\geq \lambda \|\nabla u(x) - \nabla v(x)\|_2^2
\end{aligned}$$

Integrating over Ω on both sides we get

$$\begin{aligned}
&\langle \partial_{\nabla u} L(x,u,\nabla u) - \partial_{\nabla v} L(x,v,\nabla v), \nabla u - \nabla v \rangle_{L^2(\Omega)} \\
&+ \langle \partial_u L(x,u,\nabla u) - \partial_v L(x,v,\nabla v), u - v \rangle_{L^2(\Omega)} \\
&\geq \lambda \|\nabla u - \nabla v\|_{L^2(\Omega)}^2 \\
&= \lambda \|u - v\|_{H_0^1(\Omega)}^2.
\end{aligned}$$

Therefore we have,

$$\lambda\|u-v\|^2_{H^1_0(\Omega)} \leq \langle D\mathcal{E}(u) - D\mathcal{E}(v), u-v\rangle_{L^2(\Omega)} \leq (\Lambda + C_p^2\Lambda)\|u-v\|^2_{H^1_0(\Omega)}$$

as we wanted.

To show part 3, we will again use the fact that the function for a given $x \in \Omega$ the function $L(x,\cdot,\cdot)$ is strongly convex and smooth. Therefore using Taylor's Theorem $L(x, u+v, \nabla u+\nabla v)$ along $L(x,u,\nabla u)$ we can re-write the energy function as:

$$\begin{aligned}
&\mathcal{E}(u+v)\\
&= \int_\Omega L(x,u(x)+v(x),\nabla u(x)+\nabla v(x)) - f(x)(u(x)+v(x))dx\\
&= \int_\Omega L(x,u(x),\nabla u(x)) + \nabla_{(u,\nabla u)}L(x,u(x),\nabla u(x))^T[v(x),\nabla v(x)]\\
&\quad + \frac{1}{2}[v(x),\nabla v(x)]^T\nabla^2_{(u,\nabla u)}L(\tilde{x},u(\tilde{x}),\nabla\tilde{x})[u(x),\nabla u(x)] - \int f(x)(u(x)+v(x))dx\\
&= \int_\Omega L(x,u(x),\nabla u(x)) + [\partial_u L(u,u(x),\nabla u(x)), \partial_{\nabla u}L(x,u(x),\nabla u(x))]^T[v(x),\nabla v(x)]\\
&\quad + \frac{1}{2}[v(x),\nabla v(x)]^T\nabla^2_{(u,\nabla u)}L(\tilde{x},u(\tilde{x}),\nabla u(\tilde{x}))[v(x),\nabla v(x)] - \int f(x)(u(x)+v(x))dx
\end{aligned} \tag{11.3}$$

From Equation 4.2 of Definition 11 we know that for a given $x \in \Omega$ the function $L(x,\cdot,\cdot)$ is smooth and convex. In particular we know that,

$$\text{diag}([0,\lambda I_d]) \leq \nabla^2_{(u,\nabla u)} \leq \text{diag}[\Lambda, \Lambda I_d].$$

Using this to upper bound Equation 11.3 we get,

$$\begin{aligned}
\mathcal{E}(u+v) &\leq \int_\Omega L(x,u(x),\nabla u(x)) + [\partial_u L(u,u(x),\nabla u(x)), \partial_{\nabla u}L(x,u(x),\nabla u(x))]^T[v(x),\nabla v(x)]\\
&\quad + \frac{\Lambda}{2}[v(x),\nabla v(x)]^T[v(x),\nabla v(x)] - \int f(x)(u(x)+v(x))dx\\
&= \int_\Omega L(x,u(x),\nabla u(x)) + \partial_u L(u,u(x),\nabla u(x))v(x) + \partial_{\nabla u}L(x,u(x),\nabla u(x))\nabla v(x)\\
&\quad + \frac{\Lambda}{2}\left(v(x)^2 + \|\nabla v(x)\|_2^2\right) - \int f(x)(u(x)+v(x))dx\\
&= \mathcal{E}(u) + \langle D\mathcal{E}(u) - f, v\rangle_{L^2(\Omega)} + \frac{\Lambda}{2}\left(\|v\|_{L^2(\Omega)} + \|v\|_{H^1_0(\Omega)}\right)\\
\implies \mathcal{E}(u+v) &\leq \mathcal{E}(u) + \langle D\mathcal{E}(u) - f, v\rangle_{L^2(\Omega)} + \frac{\Lambda(1+C_p^2)}{2}\|v\|_{H^1_0(\Omega)}
\end{aligned} \tag{11.4}$$

We can similarly lower bound Equation 11.3 by using the convexity of $\nabla^2_{(u,\nabla u)} L$ as

$$\begin{aligned}
\mathcal{E}(u+v) &\geq \int_\Omega L(x,u(x),\nabla u(x)) + [\partial_u L(u,u(x),\nabla u(x)), \partial_{\nabla u} L(x,u(x),\nabla u(x))]^T [v(x),\nabla v(x)] \\
&\quad + \frac{\Lambda}{2}\nabla v(x)^T \nabla v(x) - \int f(x)(u(x)+v(x))dx \\
&= \int_\Omega L(x,u(x),\nabla u(x)) + \partial_u L(u,u(x),\nabla u(x))v(x) + \partial_{\nabla u} L(x,u(x),\nabla u(x))\nabla v(x) \\
&\quad + \frac{\lambda}{2}\|\nabla v(x)\|_2^2 - \int f(x)(u(x)+v(x))dx \\
\implies \mathcal{E}(u+v) &\geq \mathcal{E}(u) + \langle D\mathcal{E}(u) - f, v\rangle_{L^2(\Omega)} + \frac{\lambda}{2}\|v\|_{H_0^1(\Omega)} \qquad (11.5)
\end{aligned}$$

Combining Equation 11.4 and Equation 11.5 we get,

$$\frac{\lambda}{2}\|\nabla v\|^2_{L^2(\Omega)} + \langle D\mathcal{E}(u) - f, v\rangle_{L^2(\Omega)} \leq \mathcal{E}(u+v) - \mathcal{E}(u) \leq \langle D\mathcal{E}(u) - f, v\rangle_{L^2(\Omega)} + \frac{(1+C_p)^2\Lambda}{2}\|\nabla v\|^2_{L^2(\Omega)}$$

Finally, part 4 follows by plugging in $u = u^\star$ and $v = u - u^\star$ in part 3 and using the fact that $D\mathcal{E}(u^\star) = f$. □

11.1.2 Proof of Lemma 18

Proof. Let $\{\lambda_i, \phi_i\}_{i=1}^\infty$ denote the (eigenvalue, eigenfunction) pairs of the operator $-\Delta$ where $0 < \lambda_1 \leq \lambda_2 \leq \cdots$, which are real and countable. (Evans [2010], Theorem 1, Section 6.5)

Using the definition of eigenvalues and eigenfunctions, we have

$$\begin{aligned}
\lambda_1 &= \inf_{v \in H_0^1(\Omega)} \frac{\langle -\Delta v, v\rangle_{L^2(\Omega)}}{\|v\|^2_{L^2(\Omega)}} \\
&= \inf_{v \in H_0^1(\Omega)} \frac{\langle \nabla v, \nabla v\rangle_{L^2(\Omega)}}{\|v\|^2_{L^2(\Omega)}} \\
&= \frac{1}{C_p}.
\end{aligned}$$

where in the last equality we use Theorem 14.

Let us write the functions v, w in the eigenbasis as $v = \sum_i \mu_i \phi_i$. Notice that an eigenfunction of $-\Delta$ is also an eigenfunction for $(I - \Delta)^{-1}$, with correspondinding eigenvalue $\frac{1}{1+\lambda_i}$.

Thus, to show part 1, we have,

$$\begin{aligned}\left\|(I-\Delta)^{-1}\nabla_x\cdot\nabla v\right\|^2_{L^2(\Omega)} &= \left\|(I-\Delta)^{-1}\Delta v\right\|^2_{L^2(\Omega)}\\ &= \left\|\sum_{i=1}^{\infty}\frac{\lambda_i}{1+\lambda_i}\mu_i\phi_i\right\|^2_{L^2(\Omega)}\\ &\leq \left\|\sum_{i=1}^{\infty}\mu_i\phi_i\right\|^2_{L^2(\Omega)}\\ &= \sum_{i=1}^{\infty}\mu_i^2 = \|u\|^2_{L^2(\Omega)}\end{aligned}$$

where in the last equality we use the fact that ϕ_i are orthogonal.

Now, bounding $\langle(I-\Delta)^{-1}v,v\rangle_{L^2(\Omega)}$ for part 2. we use the fact that eigenvalues of the operator $(I-\Delta)^{-1}$ are of the form $\left\{\frac{1}{1+\lambda_i}\right\}_{i=1}^{\infty}$ we have,

$$\begin{aligned}\langle(I-\Delta)^{-1}v,v\rangle_{L^2(\Omega)} &= \left\langle\sum_{i=1}^{\infty}\frac{\mu_i}{1+\lambda_i}\phi_i,\sum_{i=1}^{\infty}\mu_i\phi_i\right\rangle_{L^2(\Omega)}\\ &\leq \left\langle\sum_{i=1}^{\infty}\mu_i\phi_i,\sum_{i=1}^{\infty}\mu_i\phi_i\right\rangle_{L^2(\Omega)}\\ &= \|u\|^2_{L^2(\Omega)}\end{aligned}\tag{11.6}$$

Before proving part 3., note that since $\lambda_1\leq\lambda_2\leq\cdots$ and $\frac{x}{1+x}$ is monotonically increasing, we have for all $i\in\mathbb{N}$

$$\frac{1}{1+\lambda_i}\geq\frac{1}{(1+C_p)\lambda_i}\tag{11.7}$$

and note that $\frac{1}{\lambda_i}$ are the eigenvalues for $(-\Delta)^{-1}$ for all $i\in\mathbb{N}$. Using the inequality in Equation 11.7 and the fact that ϕ_i's are orthogonal, we can further lower bound $\langle(I-\Delta)^{-1}v,v\rangle_{L^2(\Omega)}$ as follows,

$$\begin{aligned}\langle(I-\Delta)^{-1}v,v\rangle_{L^2(\Omega)} &= \sum_{i=1}^{\infty}\frac{\mu_i^2}{1+\lambda_i}\|\phi_i\|^2_{L^2(\Omega)}\\ &\geq \sum_{i=1}^{\infty}\frac{\mu_i^2}{(1+pc)\lambda_i}\|\phi_i\|^2_{L^2(\Omega)}\\ &= \frac{1}{1+C_p}\langle(-\Delta)^{-1}v,v\rangle_{L^2(\Omega)},\end{aligned}$$

where we use the following set of equalities in the last step,

$$\langle(-\Delta)^{-1}v,v\rangle_{L^2(\Omega)} = \left\langle\sum_{i=1}^{\infty}\frac{\mu_i}{\lambda_i}\phi_i,\sum_{i=1}^{\infty}\mu_i\phi_i\right\rangle_{L^2(\Omega)} = \sum_{i=1}^{\infty}\frac{\mu_i^2}{\lambda_i}\|\phi_i\|^2_{L^2(\Omega)}.$$ □

11.1.3 Proof of Lemma 19: Convergence of Preconditioned Gradient Descent

Proof. For the analysis we consider $\eta = \frac{\lambda^4}{4(1+C_p)^7\Lambda^4}$

Taylor expanding as in Equation 11.4, we have

$$\mathcal{E}(u_{t+1}) \leq \mathcal{E}(u_t) - \eta \underbrace{\left\langle D\mathcal{E}(\nabla u_t) - f, (I-\Delta_x)^{-1}(D\mathcal{E}(u_t)-f)\right\rangle_{L^2(\Omega)}}_{\text{Term 1}} + \underbrace{\frac{\eta^2(1+C_p)^2\Lambda}{2}\left\|\nabla_x(I-\Delta_x)^{-1}(D\mathcal{E}(u_t)-f)\right\|^2_{L^2(\Omega)}}_{\text{Term 2}}. \tag{11.8}$$

where we have in Equation 11.4 plugged in $u_{t+1} - u_t = -\eta(I-\Delta_x)^{-1}(D\mathcal{E}(u_t)-f)$.

First we lower bound *Term 1.* Since $u^\star$ is the solution to the PDE in Equation 4.4, we have $D\mathcal{E}(u^\star) = f$. Therefore we have

$$\left\langle D\mathcal{E}(u_t) - f, (I-\Delta_x)^{-1}(D\mathcal{E}(u_t)-f)\right\rangle_{L^2(\Omega)} = \left\langle D\mathcal{E}(u_t) - D\mathcal{E}(u^\star), (I-\Delta_x)^{-1}(D\mathcal{E}(u_t)-D\mathcal{E}(u^\star))\right\rangle_{L^2(\Omega)} \tag{11.9}$$

Using the result from Lemma 18 part 3., we have,

$$\begin{aligned}&\langle D\mathcal{E}(u_t) - D\mathcal{E}(u^\star), (I-\Delta_x)^{-1}D\mathcal{E}(u_t) - D\mathcal{E}(u^\star)\rangle_{L^2(\Omega)}\\ &\geq \frac{1}{1+C_p}\left(\langle D\mathcal{E}(u_t) - D\mathcal{E}(u^\star), (-\Delta_x)^{-1}D\mathcal{E}(u_t) - D\mathcal{E}(u^\star)\rangle_{L^2(\Omega)}\right)\end{aligned}$$

Using the Equation Equation 11.9 and the fact that $\langle D\mathcal{E}(u), v\rangle_{L^2(\Omega)} = \langle \partial_{\nabla u}L(x,u,\nabla u), \nabla v\rangle_{L^2(\Omega)} + \langle \partial_u L(x,u,\nabla u), v\rangle_{L^2(\Omega)}$ from Lemma 17 we get,

$$\begin{aligned}&\langle D\mathcal{E}(u_t) - D\mathcal{E}(u^\star), (I-\Delta_x)^{-1}D\mathcal{E}(u_t) - D\mathcal{E}(u^\star)\rangle_{L^2(\Omega)}\\ &\geq \frac{1}{1+C_p}\left(\langle D\mathcal{E}(u_t) - D\mathcal{E}(u^\star), (-\Delta_x)^{-1}D\mathcal{E}(u_t) - D\mathcal{E}(u^\star)\rangle_{L^2(\Omega)}\right)\\ &= \frac{1}{1+C_p}\left(\langle \partial_{\nabla u}L(x,u_t,\nabla u_t) - \partial_{\nabla u}L(x,u^\star,\nabla u^\star), \nabla_x(-\Delta_x)^{-1}(D\mathcal{E}(u_t)-D\mathcal{E}(u^\star))\rangle_{L^2(\Omega)}\right)\\ &\quad + \frac{1}{1+C_p}\left(\langle \partial_u L(x,u_t,\nabla u_t) - \partial_u L(x,u^\star,\nabla u^\star), (-\Delta_x)^{-1}(D\mathcal{E}(u_t)-D\mathcal{E}(u^\star))\rangle_{L^2(\Omega)}\right)\\ &= \frac{1}{1+C_p}\Big\langle \nabla_{(u,\nabla u)}L(x,u_t,\nabla u_t) - \nabla_{(u,\nabla u)}L(x,u^\star,\nabla u^\star),\\ &\qquad \left[(-\Delta_x)^{-1}(D\mathcal{E}(u_t)-D\mathcal{E}(u^\star)), \nabla_x(-\Delta_x)^{-1}(D\mathcal{E}(u_t)-D\mathcal{E}(u^\star))\right]\Big\rangle_{L^2(\Omega)}\end{aligned} \tag{11.10}$$

where we combine the terms $\nabla_x(-\Delta_x)^{-1}(D\mathcal{E}(u_t)-D\mathcal{E}(u^\star))$ and $\nabla_x(-\Delta_x)^{-1}(D\mathcal{E}(u_t)-D\mathcal{E}(u^\star))$ into a single vector in the last step.

Now, note that since for any $x \in \Omega$ the function $L(x,\cdot,\cdot)$ is strongly convex, we have

$$\nabla^2_{(u,\nabla u)}L(x,\nabla u,\nabla x) \geq \operatorname{diag}([0,\lambda \mathbf{1}_d])$$

Therefore for all x we can bound $\nabla_{(u,\nabla u)} L(x, u_t(x), \nabla u_t(x)) - \nabla_{(u,\nabla u)} L(x, u^\star(x), \nabla u^\star(x))$

$$\begin{aligned}
&\nabla_{(u,\nabla u)} L(x, u_t(x), \nabla u_t(x)) - \nabla_{(u,\nabla u)} L(x, u^\star(x), \nabla u^\star(x)) \\
&= [u_t(x) - u^\star(x), \nabla u_t(x) - \nabla u^\star(x)]^T \left(\nabla^2_{(u,\nabla u)} L(\tilde{x}, u(\tilde{x}), \nabla u(\tilde{x})\right)
\end{aligned} \tag{11.11}$$

where $\tilde{x} \in \Omega$ (and potentially different from x).

Using Equation 11.11 in Equation 11.10, we can lower bound the term as follows:

$$\begin{aligned}
&\langle D\mathcal{E}(u_t) - D\mathcal{E}(u^\star), (I - \Delta_x)^{-1} D\mathcal{E}(u_t) - D\mathcal{E}(u^\star)\rangle_{L^2(\Omega)} \\
&\geq \frac{1}{1+C_p} \Bigg\langle [u_t - u^\star, \nabla u_t - \nabla u^\star]^T \left(\nabla^2_{(u,\nabla u)} L(\tilde{x}, u(\tilde{x}), \nabla u(\tilde{x})\right), \\
&\qquad \left[(-\Delta_x)^{-1}(D\mathcal{E}(u_t) - D\mathcal{E}(u^\star)), \nabla_x (-\Delta_x)^{-1}(D\mathcal{E}(u_t) - D\mathcal{E}(u^\star))\right] \Bigg\rangle_{L^2(\Omega)} \\
&\geq \frac{1}{1+C_p} \left\langle [0, \lambda(\nabla u_t(x) - \nabla u^\star(x))], \left[(-\Delta_x)^{-1}(D\mathcal{E}(u_t) - D\mathcal{E}(u^\star)), \nabla_x (-\Delta_x)^{-1}(D\mathcal{E}(u_t) - D\mathcal{E}(u^\star))\right] \right\rangle_{L^2(\Omega)} \\
&= \frac{\lambda}{1+C_p} \left\langle \nabla u_t - \nabla u^\star, \nabla_x (-\Delta_x)^{-1}(D\mathcal{E}(u_t) - D\mathcal{E}(u^\star)) \right\rangle_{L^2(\Omega)} \\
&\overset{(i)}{=} \frac{\lambda}{1+C_p} \left\langle (-\Delta) u_t - (-\Delta) u^\star, (-\Delta_x)^{-1}(D\mathcal{E}(u_t) - D\mathcal{E}(u^\star)) \right\rangle_{L^2(\Omega)} \\
&\overset{(ii)}{=} \frac{\lambda}{1+C_p} \left\langle (-\Delta)^{-1}(-\Delta) u_t - (-\Delta)^{-1}(-\Delta) u^\star, (D\mathcal{E}(u_t) - D\mathcal{E}(u^\star)) \right\rangle_{L^2(\Omega)} \\
&\overset{(iii)}{=} \frac{\lambda}{1+C_p} \langle u_t - u^\star, (D\mathcal{E}(u_t) - D\mathcal{E}(u^\star)) \rangle_{L^2(\Omega)} \\
&\overset{(iv)}{\geq} \frac{\lambda^2}{1+C_p} \|u_t - u^\star\|^2_{H^1_0(\Omega)}
\end{aligned} \tag{11.12}$$

Here, we use the fact that for all $u, v \in H^1_0(\Omega)$ we have $\langle \nabla u, \nabla v \rangle_{L^2(\Omega)} = \langle -\Delta u, v \rangle_{L^2(\Omega)}$, i.e., Green's identity (along with the fact that we have a Dirichlet Boundary condition) to get step (i). We use the symmetry of the operator $(-\Delta)^{-1}$ in step (ii), and the fact that for a function $g \in H^1_0(\Omega)$ $(-\Delta)^{-1}(-\Delta) g = g$ in step (iii). We finally use Part 2 of Lemma 17 in the final step.

Hence finally Term 1 can be simplified as,

$$\begin{aligned}
&\langle D\mathcal{E}(u_t) - D\mathcal{E}(u^\star), (I - \Delta_x)^{-1} D\mathcal{E}(u_t) - D\mathcal{E}(u^\star)\rangle_{L^2(\Omega)} \\
&\geq \frac{\lambda^2}{1+C_p} \|u_t - u^\star\|^2_{H^1_0(\Omega)} \\
&\geq \frac{2\lambda^2}{(1+C_p)^3 \Lambda} (\mathcal{E}(u_t) - \mathcal{E}(u^\star))
\end{aligned}$$

where we use Part 4 from Lemma 17 in the final step.

We will proceed to upper bounding *Term 2*. Using the definition of $H_0^1(\Omega)$ norm, we can re-write *Term 2* as,

$$\left\|\nabla_x(1-\Delta_x)^{-1}(D\mathcal{E}(u_t)-f)\right\|^2_{L^2(\Omega)} = \left\|(1-\Delta_x)^{-1}(D\mathcal{E}(u_t)-f)\right\|^2_{H_0^1(\Omega)}$$

Writing the $H_0^1(\Omega)$ norm in its variational form (since $H_0^1(\Omega)$ norm is self-adjoint, Lemma 39) and upper bounding it,

$$\begin{aligned}
&\left\|(1-\Delta_x)^{-1}(D\mathcal{E}(u_t)-f)\right\|_{H_0^1(\Omega)} \\
&= \sup_{\substack{v\in H_0^1(\Omega)\\ \|v\|_{H_0^1(\Omega)}=1}} \left\langle \nabla_x(1-\Delta_x)^{-1}(D\mathcal{E}(u_t)-f), \nabla v\right\rangle_{L^2(\Omega)} \\
&= \sup_{\substack{v\in H_0^1(\Omega)\\ \|v\|_{H_0^1(\Omega)}=1}} \left\langle \nabla_x(1-\Delta_x)^{-1}(D\mathcal{E}(u_t)-D\mathcal{E}(u^\star)), \nabla v\right\rangle_{L^2(\Omega)} \\
&\overset{(i)}{=} \sup_{\substack{v\in H_0^1(\Omega)\\ \|v\|_{H_0^1(\Omega)}=1}} \left\langle (1-\Delta_x)^{-1}(D\mathcal{E}(u_t)-D\mathcal{E}(u^\star)), -\Delta v\right\rangle_{L^2(\Omega)} \\
&\overset{(ii)}{=} \sup_{\substack{v\in H_0^1(\Omega)\\ \|v\|_{H_0^1(\Omega)}=1}} \left\langle (-\Delta)(1-\Delta_x)^{-1}(D\mathcal{E}(u_t)-D\mathcal{E}(u^\star)), v\right\rangle_{L^2(\Omega)} \\
&\leq \sup_{\substack{v\in H_0^1(\Omega)\\ \|v\|_{H_0^1(\Omega)}=1}} \left\langle D\mathcal{E}(u_t)-D\mathcal{E}(u^\star), v\right\rangle_{L^2(\Omega)} \qquad (11.13)
\end{aligned}$$

here, step (i) follows from the equality that for all $u, v \in H_0^1(\Omega)$ we have $\langle \nabla u, \nabla v\rangle_{L^2(\Omega)} = \langle -\Delta u, v\rangle_{L^2(\Omega)}$ and the fact that $-\Delta$ is a symmetric operator in step (ii).

Finally we use Lemma 18 Part 1 for the final step. More precisely, we use Part 1 of Lemma 18 as follows, where for a $g \in H_0^1(\Omega)$ we can write,

$$\sup_{\substack{v\in L^2(\Omega)\\ \|v\|_{L^2(\Omega)}=1}} \langle(-\Delta)(I-\Delta)^{-1}g, v\rangle_{L^2(\Omega)} = \|-\Delta(I-\Delta)^{-1}g\|_{L^2(\Omega)} \leq \|g\|_{L^2(\Omega)} =: \sup_{\substack{v\in L^2(\Omega)\\ \|v\|_{L^2(\Omega)}=1}} \langle g, v\rangle_{L^2(\Omega)}$$

Note that, from Lemma 17 we know that for all u, v we can write the inner product $\langle D\mathcal{E}(u), v\rangle$ as follows

$$\begin{aligned}
\langle D\mathcal{E}(u), v\rangle_{L^2(\Omega)} &= \langle \partial_{\nabla u} L(x, u, \nabla u), v\rangle_{L^2(\Omega)} + \langle \partial_u L(x, u, \nabla u), v\rangle_{L^2(\Omega)} \\
&= \langle \nabla_{(u,\nabla u)} L(x, u, \nabla u), [v, \nabla v]\rangle_{L^2(\Omega)}
\end{aligned}$$

that is, we we combine $\partial_{\nabla u} L$ and $\partial_u L$ into a single vector $\nabla_{(u,\nabla u)} L := [\partial_u L(x, u, \nabla u), \partial_{\nabla u} L(x, u, \nabla u)] \in \mathbb{R}^{d+1}$ and combining u and ∇u as a vector $[u, \nabla u]$.

Using this form and re-writing Equation 11.13 and using the fact that for $x \in \Omega$ $L(x, \cdot, \cdot)$ is convex and smooth in step (i), we have

$$
\begin{aligned}
&\left\|(1-\Delta_x)^{-1}(D\mathcal{E}(u_t)-f)\right\|_{H_0^1(\Omega)} \\
&\leq \sup_{\substack{v\in H_0^1(\Omega)\\ \|v\|_{H_0^1(\Omega)}=1}} \left\langle \nabla_{(u,\nabla u)} L(x,u_t,\nabla u_t) - \nabla_{(u,\nabla u)} L(x,u^\star,\nabla u^\star), [v,\nabla v]\right\rangle_{L^2(\Omega)} \\
&\overset{(i)}{=} \sup_{\substack{v\in H_0^1(\Omega)\\ \|v\|_{H_0^1(\Omega)}=1}} \left\langle [u_t-u^\star, \nabla u_t - \nabla u^\star]^T \nabla^2_{(u,\nabla u)} L(\tilde{x}, u(\tilde{x}), \nabla u(\tilde{x})), [v,\nabla v]\right\rangle_{L^2(\Omega)} \\
&\leq \sup_{\substack{v\in H_0^1(\Omega)\\ \|v\|_{H_0^1(\Omega)}=1}} \Lambda \left\langle [u_t-u^\star, \nabla u_t - \nabla u^\star]^T, [v,\nabla v]\right\rangle_{L^2(\Omega)} \\
&= \sup_{\substack{v\in H_0^1(\Omega)\\ \|v\|_{H_0^1(\Omega)}=1}} \Lambda \langle u_t-u^\star, v\rangle_{L^2(\Omega)} + \Lambda \langle \nabla(u_t-u^\star), \nabla v\rangle_{L^2(\Omega)} \\
&= \sup_{\substack{v\in H_0^1(\Omega)\\ \|v\|_{H_0^1(\Omega)}=1}} \Lambda C_p^2 \|u_t-u^\star\|_{H_0^1(\Omega)}\|v\|_{H_0^1(\Omega)} + \Lambda \|u_t-u^\star\|_{H_0^1(\Omega)}\|v\|_{H_0^1(\Omega)} \\
&= \Lambda(1+C_p^2)\|u_t-u^\star\|_{H_0^1(\Omega)} \leq \Lambda(1+C_p)^2\|u_t-u^\star\|_{H_0^1(\Omega)}
\end{aligned} \tag{11.14}
$$

where we use the Poincare Inequality 14 in the final step.

Therefore, from the final result in Equation 11.14 we can upper bound Term 2 in Equation 11.8 to get,

$$
\begin{aligned}
\left\|\nabla_x (I-\Delta_x)^{-1} D\mathcal{E}(u_t)\right\|^2_{L^2(\Omega)} &\leq \Lambda^2(1+C_p)^2\|u_t-u^\star\|^2_{H_0^1(\Omega)} \\
&\leq \frac{\Lambda^2(1+C_p)^2}{\lambda}(\mathcal{E}(u_t)-\mathcal{E}(u^\star))
\end{aligned}
$$

where we use the result from part 4 from Lemma 17.

$$
\implies \mathcal{E}(u_{t+1})-\mathcal{E}(u^\star) \leq \mathcal{E}(u_t)-\mathcal{E}(u^\star) - \left(\frac{2\lambda^2}{(1+C_p)^3\Lambda} - \eta\frac{(1+C_p)^4\Lambda^3}{\lambda}\right)\eta(\mathcal{E}(u_t)-\mathcal{E}(u^\star))
$$

Since $\eta = \frac{\lambda^4}{4(1+C_p)^7\Lambda^4}$ we have

$$
\begin{aligned}
&\mathcal{E}(u_{t+1})-\mathcal{E}(u^\star) \leq \mathcal{E}(u_t)-\mathcal{E}(u^\star) - \frac{\lambda^2}{(1+C_p)^3\Lambda}\eta(\mathcal{E}(u_t)-\mathcal{E}(u^\star)) \\
&\implies \mathcal{E}(u_{t+1})-\mathcal{E}(u^\star) \leq \left(1-\frac{\lambda^6}{(1+C_p)^{10}\Lambda^5}\right)^t (\mathcal{E}(u_0)-\mathcal{E}(u^\star)).
\end{aligned}
$$

□

11.2 Error Analysis

11.2.1 Proof of Lemma 24

Proof. We define for all t $r_t = \tilde{u}_t - u_t$, and will iteratively bound $\|r_t\|_{L^2(\Omega)}$.

Starting with $u_0 = 0$ and $\tilde{u}_t = 0$, we define the iterative sequences as,

$$\begin{cases} u_0 = u_0 \\ u_{t+1} = u_t - \eta(I - \Delta_x)^{-1} D\mathcal{E}(u_t) \end{cases}$$

$$\begin{cases} \tilde{u}_t = u_0 \\ \tilde{u}_{t+1} = \tilde{u}_t - \eta(I - \Delta_x)^{-1} D\tilde{\mathcal{E}}(\tilde{u}_t) \end{cases}$$

where $\eta \in \left(0, \frac{\lambda^4}{4(1+C_p)^7\Lambda^4}\right]$. Subtracting the two we get,

$$\begin{aligned} \tilde{u}_{t+1} - u_{t+1} &= \tilde{u}_t - u_t - \eta(I - \Delta_x)^{-1}\Big(D\tilde{\mathcal{E}}(\tilde{u}_t) - D\mathcal{E}(u_t)\Big) \\ \implies r_{t+1} &= r_t - \eta(I - \Delta_x)^{-1}\Big(D\tilde{\mathcal{E}}(u_t + r_t) - D\mathcal{E}(u_t)\Big) \end{aligned} \tag{11.15}$$

Taking $H_0^1(\Omega)$ norm on both sides we get,

$$\|r_{t+1}\|_{H_0^1(\Omega)} \leq \|r_t\|_{H_0^1(\Omega)} + \eta\left\|(I - \Delta_x)^{-1}\Big(D\tilde{\mathcal{E}}(u_t + r_t) - D\mathcal{E}(u_t)\Big)\right\|_{H_0^1(\Omega)} \tag{11.16}$$

Towards bounding $\left\|(I-\Delta_x)^{-1}D\tilde{\mathcal{E}}(u_t+r_t)-D\mathcal{E}(u_t)\right\|_{H_0^1(\Omega)}$, from Lemma 38 we know that the dual norm of $\|w\|_{H_0^1(\Omega)}$ is $\|w\|_{H_0^1(\Omega)}$, thus,

$$
\begin{aligned}
&\left\|(I-\Delta_x)^{-1}D\tilde{\mathcal{E}}(u_t+r_t)-D\mathcal{E}(u_t)\right\|_{H_0^1(\Omega)}\\
&= \sup_{\substack{\varphi\in H_0^1(\Omega)\\ \|\varphi\|_{H_0^1(\Omega)}=1}} \left\langle \nabla(I-\Delta_x)^{-1}\Big(D\tilde{\mathcal{E}}(u_t+r_t)-D\mathcal{E}(u_t)\Big),\nabla\varphi\right\rangle_{L^2(\Omega)}\\
&= \sup_{\substack{\varphi\in H_0^1(\Omega)\\ \|\varphi\|_{H_0^1(\Omega)}=1}} \left\langle \nabla(I-\Delta_x)^{-1}\Big(D\tilde{\mathcal{E}}(u_t+r_t)-D\mathcal{E}(u_t+r_t)\Big),\nabla\varphi\right\rangle_{L^2(\Omega)}\\
&\quad + \sup_{\substack{\varphi\in H_0^1(\Omega)\\ \|\varphi\|_{H_0^1(\Omega)}=1}} \left\langle \nabla(I-\Delta_x)^{-1}(D\mathcal{E}(u_t+r_t)-D\mathcal{E}(u_t)),\nabla\varphi\right\rangle_{L^2(\Omega)}\\
&= \sup_{\substack{\varphi\in H_0^1(\Omega)\\ \|\varphi\|_{H_0^1(\Omega)}=1}} \left\langle (I-\Delta_x)^{-1}\Big(D\tilde{\mathcal{E}}(u_t+r_t)-D\mathcal{E}(u_t+r_t)\Big),\Delta\varphi\right\rangle_{L^2(\Omega)}\\
&\quad + \sup_{\substack{\varphi\in H_0^1(\Omega)\\ \|\varphi\|_{H_0^1(\Omega)}=1}} \left\langle (I-\Delta_x)^{-1}(D\mathcal{E}(u_t+r_t)-D\mathcal{E}(u_t)),\Delta\varphi\right\rangle_{L^2(\Omega)}\\
&= \sup_{\substack{\varphi\in H_0^1(\Omega)\\ \|\varphi\|_{H_0^1(\Omega)}=1}} \left\langle \Big(D\tilde{\mathcal{E}}(u_t+r_t)-D\mathcal{E}(u_t+r_t)\Big),(I-\Delta)^{-1}\Delta\varphi\right\rangle_{L^2(\Omega)}\\
&\quad + \sup_{\substack{\varphi\in H_0^1(\Omega)\\ \|\varphi\|_{H_0^1(\Omega)}=1}} \left\langle (D\mathcal{E}(u_t+r_t)-D\mathcal{E}(u_t)),(I-\Delta)^{-1}\Delta\varphi\right\rangle_{L^2(\Omega)}\\
&\leq \sup_{\substack{\varphi\in H_0^1(\Omega)\\ \|\varphi\|_{H_0^1(\Omega)}=1}} \left\langle \Big(D\tilde{\mathcal{E}}(u_t+r_t)-D\mathcal{E}(u_t+r_t)\Big),\varphi\right\rangle_{L^2(\Omega)}\\
&\quad + \sup_{\substack{\varphi\in H_0^1(\Omega)\\ \|\varphi\|_{H_0^1(\Omega)}=1}} \left\langle (D\mathcal{E}(u_t+r_t)-D\mathcal{E}(u_t)),\varphi\right\rangle_{L^2(\Omega)} \qquad (11.17)
\end{aligned}
$$

Now from Assumption 1, we know that for all $x\in\Omega$ and $u\in H_0^1(\Omega)$ we have the following bounds on the difference of partials of L and $\mathbf{L}$:

$$\sup\|\partial_u\mathbf{L}(x,u(x),\nabla u(x))-\partial_u L(x,u(x),\nabla u(x))\|_2\leq\epsilon_L\|u(x)\|_2, \qquad (11.18)$$

and

$$\sup\|\partial_{\nabla u}\mathbf{L}(x,u(x),\nabla u(x))-\partial_{\nabla u} L(x,u(x),\nabla u(x))\|_2\leq\epsilon_L\|u(x)\|_2, \qquad (11.19)$$

Therefore, note that we can bound the difference of $\nabla_{(u,\nabla u)}\mathbf{L}$ and $\nabla_{(u,\nabla u)}L$ for all $x\in\Omega$ and $u\in H_0^1(\Omega)$ as follows,

$$
\begin{aligned}
&\sup\left\|\nabla_{(u,\nabla u)}\mathbf{L}(x,u(x),\nabla u(x))-\nabla_{(u,\nabla u)}L(x,u(x),\nabla u(x))\right\|_2\\
&\leq \sup\|\partial_{\nabla u}\mathbf{L}(x,u(x),\nabla u(x))-\partial_{\nabla u}L(x,u(x),\nabla u(x))\|_2+\sup\|\partial_{\nabla u}\mathbf{L}(x,u(x),\nabla u(x))-\partial_{\nabla u}L(x,u(x),\nabla u(x)\\
&\leq 2\epsilon_L\|u(x)\|_2 \qquad (11.20)
\end{aligned}
$$

Note that, from Lemma 17 we know that for all u, v we can write the inner product $\langle D\mathcal{E}(u), v\rangle$ as follows

$$\begin{aligned}\langle D\mathcal{E}(u), v\rangle_{L^2(\Omega)} &= \langle \partial_{\nabla u} L(x, u, \nabla u), v\rangle_{L^2(\Omega)} + \langle \partial_u L(x, u, \nabla u), v\rangle_{L^2(\Omega)} \\ &= \langle \nabla_{(u,\nabla u)} L(x, u, \nabla u), [v, \nabla v]\rangle_{L^2(\Omega)} \end{aligned} \tag{11.21}$$

that is, we we combine $\partial_{\nabla u} L$ and $\partial_u L$ into a single vector $\nabla_{(u,\nabla u)} L := [\partial_u L(x, u, \nabla u), \partial_{\nabla u} L(x, u, \nabla u)] \in \mathbb{R}^{d+1}$ and combining u and ∇u as a vector $[u, \nabla u]$.

Using upper bound in Equation 11.20 we can upper bound $\sup_{\substack{\varphi\in H_0^1(\Omega)\\ \|\varphi\|_{H_0^1(\Omega)}=1}} \left\langle \left(D\tilde{\mathcal{E}}(u_t + r_t) - D\mathcal{E}(u_t + r_t)\right), \varphi\right\rangle_{L^2(\Omega)}$ (by expanding it as in Equation 11.21) as follows,

$$\begin{aligned}
&\sup_{\substack{\varphi\in H_0^1(\Omega)\\ \|\varphi\|_{H_0^1(\Omega)}=1}} \left\langle \left(D\tilde{\mathcal{E}}(u_t + r_t) - D\mathcal{E}(u_t + r_t)\right), \varphi\right\rangle_{L^2(\Omega)} \\
&= \sup_{\substack{\varphi\in H_0^1(\Omega)\\ \|\varphi\|_{H_0^1(\Omega)}=1}} \left\langle \nabla_{(u,\nabla u)} \mathbf{L}(x, u_t + r_t, \nabla u_t + \nabla r_t) - \nabla_{(u,\nabla u)} L(x, u_t + r_t, \nabla u_t + \nabla r_t), [\varphi, \nabla\varphi]\right\rangle_{L^2(\Omega)} \\
&= \sup_{\substack{\varphi\in H_0^1(\Omega)\\ \|\varphi\|_{H_0^1(\Omega)}=1}} \langle \partial_{\nabla u} \mathbf{L}(x, u_t + r_t, \nabla u_t + \nabla r_t) - \partial_{\nabla u} L(x, u_t + r_t, \nabla u_t + \nabla r_t), \nabla\varphi\rangle_{L^2(\Omega)} \\
&= \sup_{\substack{\varphi\in H_0^1(\Omega)\\ \|\varphi\|_{H_0^1(\Omega)}=1}} \langle \partial_{\nabla u} \mathbf{L}(x, u_t + r_t, \nabla u_t + \nabla r_t) - \partial_{\nabla u} L(x, u_t + r_t, \nabla u_t + \nabla r_t), \varphi\rangle_{L^2(\Omega)} \\
&\quad + \sup_{\substack{\varphi\in H_0^1(\Omega)\\ \|\varphi\|_{H_0^1(\Omega)}=1}} \langle \partial_u \mathbf{L}(x, u_t + r_t, \nabla u_t + \nabla r_t) - \partial_u L(x, u_t + r_t, \nabla u_t + \nabla r_t), \varphi\rangle_{L^2(\Omega)} \\
&\leq \sup_{\substack{\varphi\in H_0^1(\Omega)\\ \|\varphi\|_{H_0^1(\Omega)}=1}} \epsilon_L \|u_t + r_t\|_{L^2(\Omega)} (1 + C_p) \|\varphi\|_{L^2(\Omega)} \\
&\leq \epsilon_L (1 + C_p) \|u_t + r_t\|_{L^2(\Omega)} \\
&\leq \epsilon_L (1 + C_p)^2 \|u_t + r_t\|_{H_0^1(\Omega)}
\end{aligned} \tag{11.22}$$

We can similarly bound $\sup_{\substack{\varphi\in H_0^1(\Omega)\\ \|\varphi\|_{H_0^1(\Omega)}=1}} \langle (D\mathcal{E}(u_t + r_t) - D\mathcal{E}(u_t)), \varphi\rangle_{L^2(\Omega)}$ where will use the convexity of the function $L(x, \cdot, \cdot)$ for all $u \in H_0^1(\Omega)$ to bound the gradient $\nabla_{(u,\nabla u)} L(x, u_t + r_t, \nabla u_t + \nabla r_t)$ using Taylor's theorem in the following way,

$$\begin{aligned}
&\nabla_{(u,\nabla u)} L(x, u_t + r_t, \nabla u_t + \nabla r_t) = \nabla_{(u,\nabla u)} L(x, u_t, \nabla u_t) + [r_t, \nabla r_t]^T \nabla^2_{(u,\nabla u)} L(\tilde{x}, u_t(\tilde{x}), \nabla u(\tilde{x})) \\
\implies &\nabla_{(u,\nabla u)} L(x, u_t + r_t, \nabla u_t + \nabla r_t) - \nabla_{(u,\nabla u)} L(x, u_t, \nabla u_t) = [r_t, \nabla r_t]^T \nabla^2_{(u,\nabla u)} L(\tilde{x}, u_t(\tilde{x}), \nabla u(\tilde{x}))
\end{aligned}$$

here $\tilde{x} \in \Omega$. Therefore, bounding $\sup_{\substack{\varphi \in H_0^1(\Omega) \\ \|\varphi\|_{H_0^1(\Omega)}=1}} \langle (D\mathcal{E}(u_t + r_t) - D\mathcal{E}(u_t)), \varphi \rangle_{L^2(\Omega)}$ we get,

$$
\begin{aligned}
&\sup_{\substack{\varphi \in H_0^1(\Omega) \\ \|\varphi\|_{H_0^1(\Omega)}=1}} \langle (D\mathcal{E}(u_t + r_t) - D\mathcal{E}(u_t)), \varphi \rangle_{L^2(\Omega)} \\
&= \sup_{\substack{\varphi \in H_0^1(\Omega) \\ \|\varphi\|_{H_0^1(\Omega)}=1}} \left\langle \nabla_{(u,\nabla u)} L(x, u_t + r_t, \nabla u_t + \nabla r_t) - \nabla_{(u,\nabla u)} L(x, u_t, \nabla u_t), [\varphi, \nabla \varphi] \right\rangle_{L^2(\Omega)} \\
&= \sup_{\substack{\varphi \in H_0^1(\Omega) \\ \|\varphi\|_{H_0^1(\Omega)}=1}} \left\langle [r_t, \nabla r_t]^T \nabla^2_{(u,\nabla u)} L(\tilde{x}, u(\tilde{x}), \nabla u(\tilde{x})), [\varphi, \nabla \varphi] \right\rangle_{L^2(\Omega)} \\
&\leq \sup_{\substack{\varphi \in H_0^1(\Omega) \\ \|\varphi\|_{H_0^1(\Omega)}=1}} \Lambda \left\langle [r_t, \nabla r_t]^T, [\varphi, \nabla \varphi] \right\rangle_{L^2(\Omega)} \\
&\leq \sup_{\substack{\varphi \in H_0^1(\Omega) \\ \|\varphi\|_{H_0^1(\Omega)}=1}} \Lambda \left(\|r_t\|_{L^2(\Omega)} \|\varphi\|_{L^2(\Omega)} + \|\nabla r_t\|_{L^2(\Omega)} \|\nabla \varphi\|_{L^2(\Omega)} \right) \\
&\leq \Lambda (1 + C_p)^2 \|r\|_{H_0^1(\Omega)}
\end{aligned} \tag{11.23}
$$

Plugging in Equations Equation 11.22 and Equation 11.23 in Equation 11.17 we get,

$$
\begin{aligned}
\left\| (I - \Delta_x)^{-1} D\tilde{\mathcal{E}}(u_t + r_t) - D\mathcal{E}(u_t) \right\|_{H_0^1(\Omega)} &\leq \epsilon_L (1 + C_p)^2 \|u_t + r_t\|_{H_0^1(\Omega)} + \Lambda (1 + C_p)^2 \|r\|_{H_0^1(\Omega)} \\
&= (1 + C_p)^2 (\epsilon_L + \Lambda) \|r_t\|_{H_0^1(\Omega)} + \epsilon (1 + C_p)^2 \|u_t\|
\end{aligned} \tag{11.24}
$$

Furthermore, from Lemma 19 we have for all $t \in \mathbb{N}$,

$$
\begin{aligned}
\mathcal{E}(u_{t+1}) - \mathcal{E}(u^\star) &\leq \left(1 - \frac{\lambda^6}{(1 + C_p)^8 \Lambda^5} \right)^t \mathcal{E}(u_0) \\
&\leq \mathcal{E}(u_0)
\end{aligned}
$$

and

$$
\begin{aligned}
\|u_t - u^\star\|_{H_0^1(\Omega)} &\leq \frac{2}{\lambda} (\mathcal{E}(u_t) - \mathcal{E}(u_0)) \\
&\leq \frac{2}{\lambda} \mathcal{E}(u_0)
\end{aligned}
$$

Hence we have that for all $t \in \mathbb{N}$,

$$
\|u_t\|_{H_0^1(\Omega)} \leq \|u^\star\|_{H_0^1(\Omega)} + \frac{2}{\lambda} \mathcal{E}(u_0) =: R.
$$

Putting this all together, we have

$$
\left\| (I - \Delta_x)^{-1} D\tilde{\mathcal{E}}(u_t + r_t) - D\mathcal{E}(u_t) \right\|_{H_0^1(\Omega)} \leq (1 + C_p)^2 (\epsilon_L + \Lambda) \|r_t\|_{H_0^1(\Omega)} + \epsilon_L (1 + C_p)^2 R \tag{11.25}
$$

Hence using the result from Equation 11.25 in Equation 11.16 and unfolding the recursion, we get,

$$\begin{aligned}
&\|r_{t+1}\|_{H_0^1(\Omega)} \leq \big(1+\eta(1+C_p)^2(\epsilon_L+\Lambda)\big)\|r_t\|_{H_0^1(\Omega)} + (1+C_p)^2\epsilon_L\eta R \\
\implies &\|r_{t+1}\|_{H_0^1(\Omega)} \leq \frac{(1+C_p)^2\epsilon_L\eta R}{\eta(1+C_p)^2(\epsilon_L+\Lambda)}\Big(\big(1+\eta(1+C_p)^2(\epsilon_L+\Lambda)\big)\Big)^t - 1\Big) \\
\implies &\|r_{t+1}\|_{H_0^1(\Omega)} \leq \frac{\epsilon_L R}{\epsilon_L+\Lambda}\Big(\big(1+\eta(1+C_p)^2(\epsilon_L+\Lambda)\big)\Big)^t - 1\Big) \qquad (11.26)
\end{aligned}$$

as we needed. □

11.3 Proofs for Section 4.6.2: Bounding the Barron Norm

11.3.1 Proof of Lemma 20: Barron Norm Increase after One Update

Proof. Note that the update equation looks like,

$$\begin{aligned}
\tilde{u}_{t+1} &= \tilde{u}_t - \eta(I-\Delta_x)^{-1}D\mathcal{E}(u_t) \\
&= \tilde{u}_t - \eta(I-\Delta_x)^{-1}(-\nabla\cdot\partial_{\nabla u}L(x,\tilde{u}_t,\nabla\tilde{u}_t) + \partial_u L(x,\tilde{u}_t,\nabla\tilde{u}_t) - f) \\
&= \tilde{u}_t - \eta(I-\Delta_x)^{-1}\left(-\sum_{i=1}^d \partial_i\partial_{\nabla u}L(x,\tilde{u}_t,\nabla\tilde{u}_t) + \partial_u L(x,\tilde{u}_t,\nabla\tilde{u}_t) - f\right) \qquad (11.27)
\end{aligned}$$

From Lemma 21 we have

$$\|\nabla\tilde{u}_t\|_{\mathcal{B}(\Omega)} = \max_{i\in[d]}\|\partial_i\tilde{u}_t\|_{\mathcal{B}(\Omega)} \leq 2\pi W_t\|\tilde{u}_t\|_{\mathcal{B}(\Omega)} \qquad (11.28)$$

This also implies that

$$\max\{\|\tilde{u}_t\|_{\mathcal{B}(\Omega)}, \|\nabla\tilde{u}_t\|_{\mathcal{B}(\Omega)}\} \leq 2\pi W_t\|\tilde{u}_t\|_{\mathcal{B}(\Omega)}.$$

Note that since $\tilde{u}_t \in \Gamma_{W_t}$ we have $\nabla\tilde{u}_t \in \Gamma_{2\pi W_t}$ and $L(x,\tilde{u}_t,\nabla\tilde{u}_t) \in \Gamma_{2\pi k_{\mathsf{L}} W_t}$ (from Assumption 1).

Therefore, we can bound the Barron norm as,

$$\begin{aligned}
&\left\|(I-\Delta_x)^{-1}\left(-\sum_{i=1}^d \partial_i\partial_{\nabla u}L(x,\tilde{u}_t,\nabla\tilde{u}_t) + \partial_u L(x,\tilde{u}_t,\nabla\tilde{u}_t) - f\right)\right\|_{\mathcal{B}(\Omega)} \\
&\overset{(i)}{\leq} \left\|-\sum_{i=1}^d \partial_i\partial_{\nabla u}L(x,\tilde{u}_t,\nabla\tilde{u}_t)\right\|_{\mathcal{B}(\Omega)} + \|\partial_u L(x,\tilde{u}_t,\nabla\tilde{u}_t)\|_{\mathcal{B}(\Omega)} + \|f\|_{\mathcal{B}(\Omega)} \\
&\overset{(ii)}{\leq} d\|\partial_i\partial_{\nabla u}L(x,\tilde{u}_t,\nabla\tilde{u}_t)\|_{\mathcal{B}(\Omega)} + \|\partial_u L(x,\tilde{u}_t,\nabla\tilde{u}_t)\|_{\mathcal{B}(\Omega)} + \|f\|_{\mathcal{B}(\Omega)} \\
&\leq dB_{\mathsf{L}}2\pi k_{\mathsf{L}}(2\pi W_t)^{p_{\mathsf{L}}}\|u\|^{p_{\mathsf{L}}}_{\mathcal{B}(\Omega)} + B_{\mathsf{L}}(2\pi W_t)^{p_{\mathsf{L}}}\|u\|^{p_{\mathsf{L}}}_{\mathcal{B}(\Omega)} + \|f\|_{\mathcal{B}(\Omega)} \\
&\leq (2\pi k_{\mathsf{L}} d+1)B_{\mathsf{L}}(2\pi W_t)^{p_{\mathsf{L}}}\|u\|^{p_{\mathsf{L}}}_{\mathcal{B}(\Omega)} + \|f\|_{\mathcal{B}(\Omega)}
\end{aligned}$$

where we use the fact that for a function h, we have $\|(I-\Delta_x)^{-1}h\|_{\mathcal{B}(\Omega)} \leq \|h\|_{\mathcal{B}(\Omega)}$ from Lemma 21 in (i) and the bound from Equation 11.28 in (ii).

Using the result of *Addition* from Lemma 21 we have

$$\begin{aligned}\|\tilde{u}_{t+1}\|_{\mathcal{B}(\Omega)} &\leq \|\tilde{u}_t\|_{\mathcal{B}(\Omega)} + \eta\Big(2\pi k_{\mathbf{L}} d + 1)B_{\mathbf{L}}(2\pi W_t)^{p_{\mathbf{L}}}\|u\|^{p_{\mathbf{L}}}_{\mathcal{B}(\Omega)} + \|f\|_{\mathcal{B}(\Omega)}\Big) \\ &\leq (1 + \eta(2\pi k_{\mathbf{L}} d + 1)B_{\mathbf{L}}(2\pi W_t)^{p_{\mathbf{L}}})\|\tilde{u}\|^{p_{\mathbf{L}}}_{\mathcal{B}(\Omega)} + \eta\|f\|_{\mathcal{B}(\Omega)}\end{aligned}$$

□

11.3.2 Proof of Lemma 22: Final Barron Norm Bound

Proof. From Lemma 20 we have

$$\begin{aligned}\|\tilde{u}_{t+1}\|_{\mathcal{B}(\Omega)} &\leq \|\tilde{u}_t\|_{\mathcal{B}(\Omega)} + \eta\Big((2\pi k_{\mathbf{L}} d + 1)B(2\pi W_t)^{p}\|u\|^{p}_{\mathcal{B}(\Omega)} + \|f\|_{\mathcal{B}(\Omega)}\Big) \\ &\leq (1 + \eta(2\pi k_{\mathbf{L}} d + 1)B(2\pi W_t)^{p})\|u\|^{p}_{\mathcal{B}(\Omega)} + \eta\|f\|_{\mathcal{B}(\Omega)}\end{aligned}$$

Denoting the constant $A = (1 + \eta(2\pi k_{\mathbf{L}} d + 1)B(2\pi W_t)^{p})$ we have

$$\begin{aligned}\|\tilde{u}_{t+1}\|_{\mathcal{B}(\Omega)} &= A\|\tilde{u}_t\|^{p}_{\mathcal{B}(\Omega)} + \eta\|f\|_{\mathcal{B}(\Omega)} \\ \log\big(\|\tilde{u}_{t+1}\|_{\mathcal{B}(\Omega)}\big) &= \log\Big(A\|\tilde{u}_t\|^{p}_{\mathcal{B}(\Omega)} + \eta\|f\|_{\mathcal{B}(\Omega)}\Big) \\ &= \log\left(A\|\tilde{u}_t\|^{p}_{\mathcal{B}(\Omega)}\left(1 + \frac{\eta\|f\|_{\mathcal{B}(\Omega)}}{A\|\tilde{u}_t\|^{p}_{\mathcal{B}(\Omega)}}\right)\right) \\ &\leq \log\left(A\|\tilde{u}_t\|^{p}_{\mathcal{B}(\Omega)}\left(1 + \frac{\eta\|f\|_{\mathcal{B}(\Omega)}}{\max\{1, A\|\tilde{u}_t\|^{p}_{\mathcal{B}(\Omega)}\}}\right)\right) \\ &= \log\Big(A\|\tilde{u}_t\|^{p}_{\mathcal{B}(\Omega)}\big(1 + \eta\|f\|_{\mathcal{B}(\Omega)}\big)\Big) \\ &= \log\Big(\|\tilde{u}_t\|^{p}_{\mathcal{B}(\Omega)}\Big) + \log\big(A\big(1 + \eta\|f\|_{\mathcal{B}(\Omega)}\big)\big) \\ &= r\log(\|\tilde{u}_t\|_{\mathcal{B}(\Omega)}) + \log\big(A\big(1 + \eta\|f\|_{\mathcal{B}(\Omega)}\big)\big) \qquad (11.29)\end{aligned}$$

The above equation is a recursion of the form

$$x_{t+1} \leq rx_t + c$$

which implies

$$x_{t+1} \leq c\frac{p^t - 1}{p - 1} + p^t x_0.$$

Therefore the final bound in Equation 11.29 is,

$$
\begin{aligned}
&\log\big(\|\tilde{u}_{t+1}\|_{\mathcal{B}(\Omega)}\big) \leq r\log(\|\tilde{u}_t\|_{\mathcal{B}(\Omega)}) + \log\big(A\big(1+\eta\|f\|_{\mathcal{B}(\Omega)}\big)\big)\\
\implies &\log\big(\|\tilde{u}_{t+1}\|_{\mathcal{B}(\Omega)}\big) \leq \frac{r^n-1}{r-1}\log\big(A\big(1+\eta\|f\|_{\mathcal{B}(\Omega)}\big)\big) + p^t\log(\|\tilde{u}_0\|_{\mathcal{B}(\Omega)})\\
\implies &\|\tilde{u}_{t+1}\|_{\mathcal{B}(\Omega)} \leq \big(A\big(1+\eta\|f\|_{\mathcal{B}(\Omega)}\big)\big)^{\frac{p^t-1}{p-1}}\|\tilde{u}_0\|^{p^t}_{\mathcal{B}(\Omega)}\\
\implies &\|\tilde{u}_{t+1}\|_{\mathcal{B}(\Omega)} \leq \big((1+\eta(2\pi k_{\mathbf{L}}d+1)B_{\tilde{L}}(2\pi W_t)^p)\big(1+\eta\|f\|_{\mathcal{B}(\Omega)}\big)\big)^{\frac{p^t-1}{p-1}}\|\tilde{u}_0\|^{p^t}_{\mathcal{B}(\Omega)}\\
\overset{(i)}{\implies} &\|\tilde{u}_{t+1}\|_{\mathcal{B}(\Omega)} \leq \big(\big(1+\eta(2\pi k_{\mathbf{L}}d+1)B_{\tilde{L}}(2\pi k_{\mathbf{L}}^t W_0)^p\big)\big(1+\eta\|f\|_{\mathcal{B}(\Omega)}\big)\big)^{\frac{p^t-1}{p-1}}\|\tilde{u}_0\|^{p^t}_{\mathcal{B}(\Omega)}\\
\overset{(ii)}{\implies} &\|\tilde{u}_{t+1}\|_{\mathcal{B}(\Omega)} \leq \big((1+\eta(2\pi k_{\mathbf{L}}d+1)B_{\tilde{L}}(2\pi k_{\mathbf{L}}W_0)\big)\big(1+\eta\|f\|_{\mathcal{B}(\Omega)}\big)\big)^{pt+\frac{p^t-1}{p-1}}\|\tilde{u}_0\|^{p^t}_{\mathcal{B}(\Omega)}\\
\implies &\|\tilde{u}_{t+1}\|_{\mathcal{B}(\Omega)} \leq \big((1+\eta 2\pi k_{\mathbf{L}}W_0(2\pi kd+1)B_{\tilde{L}}\big)\big(1+\eta\|f\|_{\mathcal{B}(\Omega)}\big)\big)^{pt+\frac{p^t-1}{p-1}}\Big(\max\{1,\|\tilde{u}_0\|^{p^t}_{\mathcal{B}(\Omega)}\}\Big)
\end{aligned}
$$

where we use the fact that $W_t = k_{\mathbf{L}}^T W_0$ since $\tilde{u}_t \in \Gamma_{k_{\mathbf{L}}^T W_0}$ in step (i) and use the property that $(1+x^p) \leq (1+x)^p$ since $x > 0$ in step (ii). □

11.3.3 Proof of Lemma 23

Lemma 35 (Lemma 23 restated)**.** *Let*

$$f(x) = \sum_{\alpha,|\alpha|\leq P}\left(A_\alpha \prod_{i=1}^{d} x_i^{\alpha_i}\right)$$

where α is a multi-index and $x \in \mathbb{R}^d$ and $A_\alpha \in \mathbb{R}$ is a scalar. If $g : \mathbb{R}^d \to \mathbb{R}^d$ is a function such that $g \in \Gamma_W$, then we have $f \circ g \in \Gamma_{PW}$ and the Barron norm can be bounded as,

$$\|f\circ g\|_{\mathcal{B}(\Omega)} \leq d^{P/2}\left(\sum_{\alpha,|\alpha|=1}^{P}|A_\alpha|^2\right)^{1/2}\|g\|^P_{\mathcal{B}(\Omega)}$$

Proof. Recall from Definition 16 we know that for a vector valued function $g : \mathbb{R}^d \to \mathbb{R}^d$, we have

$$\|g\|_{\mathcal{B}(\Omega)} = \max_{i\in[d]}\|g_i\|_{\mathcal{B}(\Omega)}.$$

Then, using Lemma 21, we have

$$
\begin{aligned}
\|f(g)\|_{\mathcal{B}(\Omega)} &= \left\| \sum_{\alpha,|\alpha|=0}^{P} A_\alpha \prod_{i=1}^{d} g_i^{\alpha_i} \right\|_{\mathcal{B}(\Omega)} \\
&\leq \sum_{\alpha,|\alpha|=0}^{P} \left\| A_\alpha \prod_{i=1}^{d} g_i^{\alpha_i} \right\|_{\mathcal{B}(\Omega)} \\
&\leq \sum_{\alpha,|\alpha|=0}^{P} |A_\alpha| \left\| \prod_{i=1}^{d} g_i^{\alpha_i} \right\|_{\mathcal{B}(\Omega)} \\
&\leq \sum_{\alpha,|\alpha|=0}^{P} |A_\alpha| \left\| \prod_{i=1}^{d} g_i^{\alpha_i} \right\|_{\mathcal{B}(\Omega)} \\
&\leq \sum_{\alpha,|\alpha|=0}^{P} |A_\alpha| \left(\prod_{i=1}^{d} \|g_i^{\alpha_i}\|_{\mathcal{B}(\Omega)} \right) \\
&\leq \sum_{\alpha,|\alpha|=0}^{P} |A_\alpha| \left(\prod_{i=1}^{d} \|g_i\|_{\mathcal{B}(\Omega)}^{\alpha_i} \right) \\
&= \sum_{\alpha,|\alpha|=0}^{P} |A_\alpha| \left(\prod_{i=1}^{d} \|g_i\|_{\mathcal{B}(\Omega)}^{\alpha_i} \right) \\
&\leq \left(\sum_{\alpha,|\alpha|=0}^{P} |A_\alpha|^2 \right)^{1/2} \left(\sum_{\alpha,|\alpha|=1}^{P} \left(\prod_{i=1}^{d} \|g_i\|_{\mathcal{B}(\Omega)}^{\alpha_i} \right)^2 \right)^{1/2} \qquad (11.30)
\end{aligned}
$$

where we have repeatedly used Lemma 21 and Cauchy-Schwartz in the last line. Using the fact that for a multivariate function $g : \mathbb{R}^d \to \mathbb{R}^d$ we have for all $i \in [d]$

$$\|g\|_{\mathcal{B}(\Omega)} \geq \|g_i\|_{\mathcal{B}(\Omega)}.$$

Therefore, from Equation 11.30 we get,

$$
\begin{aligned}
\|f(g)\|_{\mathcal{B}(\Omega)} &\leq \left(\sum_{\alpha,|\alpha|=0}^{P} |A_\alpha|^2 \right)^{1/2} \left(\sum_{\alpha,|\alpha|=1}^{P} \left(\|g\|_{\mathcal{B}(\Omega)}^{\sum_{i=1}^{d} \alpha_i} \right)^2 \right)^{1/2} \\
&\leq \left(\sum_{\alpha,|\alpha|=0}^{P} |A_\alpha|^2 \right)^{1/2} \left(\sum_{\alpha,|\alpha|=1}^{P} \left(\|g\|_{\mathcal{B}(\Omega)}^{\alpha} \right)^2 \right)^{1/2} \\
&\leq d^{P/2} \left(\sum_{\alpha,|\alpha|=0}^{P} |A_\alpha|^2 \right)^{1/2} \|g\|_{\mathcal{B}(\Omega)}^{P}
\end{aligned}
$$

Since the maximum power of the polynomial can take is P from Corollary 2 we will have $f \circ g \in \Gamma_{PW}$. □

11.3.4 Proof of Lemma 21: Barron Norm Algebra

The proof of Lemma 21 is fairly similar to the proof of Lemma 3.3 in Chen et al. [2021]—the change stemming from the difference of the Barron norm being considered

Proof. We first show the result for *Addition* and bound $\|h_1 + h_2\|_{\mathcal{B}(\Omega)}$,

$$\begin{aligned}
\|g_1 + g_2\|_{\mathcal{B}(\Omega)} &= \sum_{\omega \in \mathbb{N}^d} (1 + \|\omega\|_2) |\widehat{g_1 + g_2}(\omega)| \\
&= \sum_{\omega \in \mathbb{N}^d} (1 + \|\omega\|_2) |\hat{g}_1(\omega) + \hat{g}_2(\omega)| \\
&\leq \sum_{\omega \in \mathbb{N}^d} (1 + \|\omega\|_2) |\hat{g}_1(\omega)| + \sum_{\omega \in \mathbb{N}^d} (1 + \|\omega\|_2) |\hat{g}_2(\omega)| \\
\implies \|h_1 + h_2\|_{\mathcal{B}(\Omega)} &\leq \|h_1\|_{\mathcal{B}(\Omega)} + \|h_2\|_{\mathcal{B}(\Omega)}.
\end{aligned}$$

For *Multiplication*, first note that multiplication of functions is equal to convolution of the functions in the frequency domain, i.e., for functions $g_1 : \mathbb{R}^d \to d$ and $g_2 : \mathbb{R}^d \to d$, we have,

$$\widehat{g_1 \cdot g_2} = \hat{g}_1 * \hat{g}_2 \tag{11.31}$$

Now, to bound the Barron norm for the multiplication of two functions,

$$\begin{aligned}
\|g_1 \cdot g_2\|_{\mathcal{B}(\Omega)} &= \sum_{\omega \in \mathbb{N}^d} (1 + \|\omega\|_2) |\widehat{g_1 \cdot g_2}(\omega)| \\
&= \sum_{\omega \in \mathbb{N}^d} (1 + \|\omega\|_2) |\hat{g}_1 * \hat{g}_2(\omega)| \\
&= \sum_{\omega \in \mathbb{N}^d} \sum_{z \in \mathbb{N}^d} (1 + \|\omega\|_2) |\hat{g}_1(z) \hat{g}_2(\omega - z)| \\
&\leq \sum_{\omega \in \mathbb{N}^d} \sum_{z \in \mathbb{N}^d} (1 + \|\omega - z\|_2 + \|z\|_2 + \|z\|_2 \|\omega - z\|_2) |\hat{g}_1(z) \hat{g}_2(\omega - z)|
\end{aligned}$$

Where we use $\|\omega\|_2 \leq \|\omega - z\|_2 + \|z\|_2$ and the fact that

$$\sum_{\omega} \sum_{z} \|z\|_2 \|\omega - z\|_2 |\hat{g}_1(z) \hat{g}_2(\omega - z)| > 0.$$

Collecting the relevant terms together we get,

$$\begin{aligned}
\|g_1 \cdot g_2\|_{\mathcal{B}(\Omega)} &\leq \sum_{\omega \in \mathbb{N}^d} \sum_{z \in \mathbb{N}^d} (1 + \|\omega - z\|_2) \cdot (1 + \|z\|_2) |\hat{g}_1(z)| |\hat{g}_2(\omega - z)| \\
&= ((1 + \|\omega\|_2) \hat{g}_1(\omega)) * ((1 + \|\omega\|_2) \hat{g}_2(\omega))
\end{aligned}$$

Hence using Young's convolution identity from Lemma 36 we have

$$\|g_1 \cdot g_2\|_{\mathcal{B}(\Omega)} \leq \left(\sum_{\omega \in \mathbb{R}^d} (1 + \|w\|_2)\hat{g}_1(\omega)d\omega\right)\left(\sum_{\omega \in \mathbb{R}^d} (1 + \|w\|_2)\hat{g}_2(\omega)d\omega\right)$$
$$\implies \|g_1 \cdot g_2\|_{\mathcal{B}(\Omega)} \leq \|h_1\|_{\mathcal{B}(\Omega)}\|h_2\|_{\mathcal{B}(\Omega)}.$$

In order to show the bound for *Derivative*, since $h \in \Gamma_W$, there exists a function $g : \mathbb{R}^d \to \mathbb{R}$ such that,

$$g(x) = \sum_{\|\omega\|_\infty \leq W} e^{2\pi i \omega^T x}\hat{g}(\omega)d\omega$$

Now taking derivative on both sides we get,

$$\partial_j g(x) = \sum_{\|\omega\|_\infty \leq W} ie^{i\omega^T x} 2\pi\omega_j \hat{g}(\omega) \tag{11.32}$$

This implies that we can upper bound $|\widehat{\partial_i g}(\omega)|$ as

$$\widehat{\partial_j g}(\omega) = i2\pi\omega_j \hat{g}(\omega)$$
$$\implies |\widehat{\partial_j g}(\omega)| \leq 2\pi W|\hat{g}(\omega)| \tag{11.33}$$

Hence we can bound the Barron norm of $\partial_j h$ as follows:

$$\begin{aligned}
\|\partial_j g\|_{\mathcal{B}(\Omega)} &= \sum_{\|\omega\|_\infty \leq W} (1 + \|\omega\|_\infty)|\widehat{\partial_j g}(\omega)|d\omega \\
&\leq \sum_{\|\omega\|_\infty \leq W} (1 + \|\omega\|_\infty)|2\pi W\hat{g}(\omega)|d\omega \\
&\leq 2\pi W \sum_{\|\omega\|_\infty \leq W} (1 + \|\omega\|_\infty)|\hat{g}(\omega)|d\omega \\
&\leq 2\pi W\|h\|_{\mathcal{B}(\Omega)}
\end{aligned}$$

In order to show the preconditioning, note that for functions $g, f : \Omega^d \to \mathbb{R}$, if $f = (I - \Delta)^{-1}g$ then we have then we have $(I - \Delta)f = g$. Furthermore, by Lemma 37 we have

$$(1 + \|\omega\|_2^2)\hat{f}(\omega) = \hat{g}(\omega) \implies \hat{f}(\omega) = \frac{\hat{g}(\omega)}{1 + \|\omega\|_2^2}. \tag{11.34}$$

Bounding $\|(I - \Delta)^{-1}f\|_{\mathcal{B}(\Omega)}$,

$$\begin{aligned}
\|(I - \Delta)^{-1}g\|_{\mathcal{B}(\Omega)} &= \sum_{\omega \in \mathbb{N}^d} \frac{1 + \|\omega\|_2}{(1 + \|\omega\|_2^2)}\hat{g}(\omega)d\omega \\
&\leq \sum_{\omega \in \mathbb{N}^d} (1 + \|\omega\|_2)\hat{g}(\omega)d\omega \\
\implies \|(I - \Delta)^{-1}g\|_{\mathcal{B}(\Omega)} &\leq \|g\|_{\mathcal{B}(\Omega)}.
\end{aligned}$$

□

Corollary 2. *Let $g : \mathbb{R}^d \to \mathbb{R}$ then for any $k \in \mathbb{N}$ we have $\|g^k\|_{\mathcal{B}(\Omega)} \leq \|g\|^k_{\mathcal{B}(\Omega)}$. Furthermore, if the function $g \in \Gamma_W$ then the function $g^k \in \Gamma_{kW}$.*

Proof. The result from $\|g^k\|_{\mathcal{B}(\Omega)}$ follows from the multiplication result in Lemma 21 and we can show this by induction. For $n = 2$, we have from Lemma 21 we have,

$$\|g^2\|_{\mathcal{B}(\Omega)} \leq \|g\|^2_{\mathcal{B}(\Omega)} \tag{11.35}$$

Assuming that we have for all n till $k-1$ we have

$$\|g^n\|_{\mathcal{B}(\Omega)} \leq \|g\|^n_{\mathcal{B}(\Omega)} \tag{11.36}$$

for $n = k$ we get,

$$\|g^k\|_{\mathcal{B}(\Omega)} = \|gg^{k-1}\|_{\mathcal{B}(\Omega)} \leq \|g\|_{\mathcal{B}(\Omega)}\|g^{k-1}\|_{\mathcal{B}(\Omega)} \leq \|g\|^k_{\mathcal{B}(\Omega)}. \tag{11.37}$$

To show that for any k the function $g^k \in \Gamma_{kW}$, we write g^k in the Fourier basis. We have:

$$\begin{aligned} g^k(x) &= \prod_{j=1}^{k} \left(\sum_{\|\omega_j\|_\infty \leq W} \hat{g}(\omega_j) e^{2i\pi\omega_j^T x} d\omega_j \right) \\ &= \sum_{\|\omega\|_\infty \leq kW} \left(\sum_{\sum_{l=1}^k \omega_l = \omega} \prod_{j=1}^{k} \hat{g}(\omega_j) d\omega_1 \dots d\omega_k \right) e^{i2\pi\omega^T k} d\omega \end{aligned}$$

In particular, the coefficients with $\|\omega\|_\infty > kW$ vanish, as we needed. □

Lemma 36 (Young's convolution identity)**.** *For functions $g \in L^p(\mathbb{R}^d)$ and $h \in L^q(\mathbb{R}^d)$ and*

$$\frac{1}{p} + \frac{1}{q} = \frac{1}{r} + 1$$

where $1 \leq p, q, r \leq \infty$ we have

$$\|f * g\|_r \leq \|g\|_p \|h\|_q.$$

Here $*$ denotes the convolution operator.

Lemma 37. *For a differentiable function $f : [0,1]^d \to \mathbb{R}$, such that $f \in L^1(\mathbb{R}^d)$ we have*

$$\widehat{\nabla f}(\omega) = i2\pi\omega \hat{f}(\omega)$$

11.4 Existence Uniqueness and Definition of the Solution

11.4.1 Proof of Existence and Uniqueness of Minima

Proof. The proof follows a similar sketch of that provided in Fernández-Real and Ros-Oton [2020] Chapter 3, Theorem 3.3.

We first show that the minimizer $u^\star$ of the energy functional $\mathcal{E}(u)$ exists.

Note that from Definition 11 we have for a fixed $x \in \Omega$ the function $L(x,\cdot,\cdot)$ is convex and smooth it has a unique minimum, i.e., there exists a $(y_L, z_L) \in \mathbb{R} \times \mathbb{R}^d$ such that for all $(y,z) \in \mathbb{R} \times \mathbb{R}^d$ we have $L(x,y,z) \geq L(x,y_L,z_L)$ and that $\nabla L(x,y_L,z_L) = 0$. Furthermore, using Equation 4.2 from Definition 11 this also implies the following,

$$\lambda\|z - z_L\|_2^2 \leq L(x,y,z) - L(x,y_L,z_L) \leq \Lambda\big(\|y-y_L\|_2^2 + \|z-z_L\|_2^2\big).$$

Note we can (w.l.o.g) assume that for a fixed $x \in \Omega$ we have, $L(x,0,0) = 0$, and $\nabla_{y,z} L(x,0,0) = 0$ (we can redefine L as $\widetilde{L}(x,y,z) = L(x,y+y_L,z+z_L) - L(x,y_L,z_L)$ if necessary), hence the above equation can be simplified to,

$$\lambda\|z\|_2^2 \leq L(x,y,z) \leq \Lambda\big(\|y\|_2^2 + \|z\|_2^2\big), \quad \forall p \in \Omega \times \mathbb{R} \times \mathbb{R}^d. \tag{11.38}$$

Now, we define,

$$\mathcal{E}_\circ = \inf\left\{ \int_\Omega L(x,v,\nabla v) - fv \, dx : x \in \Omega, v \in H_0^1(\Omega) \right\}$$

. Let us first show that $\mathcal{E}_\circ$ is finite. Indeed, using Equation 11.38 for any $v \in H_0^1(\Omega)$ and $x \in \Omega$, we have

$$\begin{aligned}
\mathcal{E}(v) &= \int_\Omega L(x,v,\nabla v) - fv \, dx \\
&\leq \int_\Omega \Lambda\big(\|v(x)\|_2^2 + \|\nabla v(x)\|_2^2\big) + \|f(x)v(x)\|_2 dx \\
&\leq \Lambda\Big(\|v\|_{L^2(\Omega)}^2 + \|\nabla v\|_{L^2(\Omega)}^2\Big) + \|f\|_{L^2(\Omega)}\|v\|_{L^2(\Omega)}
\end{aligned}$$

and is thus finite.

Moreover, using Equation 11.38 for all $v \in H_0^1(\Omega)$ and $x \in \Omega$, $\mathcal{E}(v)$ can be lower bounded as

$$\begin{aligned}
\mathcal{E}(v) &= \int_\Omega L(x,v,\nabla v) - fv \, dx \\
&\geq \int_\Omega \lambda\|\nabla v(x)\|_2 - \|f(x)v(x)\|_2 \, dx \\
&\geq \lambda\|\nabla v\|_{L^2(\Omega)}^2 - \|f\|_{L^2(\Omega)}\|v\|_{L^2(\Omega)} \\
&\geq \frac{\lambda}{2}\|\nabla v\|_{L^2(\Omega)}^2 + \left(\frac{\lambda}{2C_p} - \frac{1}{C}\right)\|v\|_{L^2(\Omega)}^2 - C\|f\|_{L^2(\Omega)}^2
\end{aligned} \tag{11.39}$$

for some large constant C so that $\lambda/2C_p - 1/C > 0.$, where we have used the Poincare inequality (Theorem 14) and Cauchy-Schwarz inequality to get the last inequality.

Let $\{u_k\}$ where $u_k \in H_0^1(\Omega)\ \forall k$ define a minimizing sequence of function, that is, we have $\mathcal{E}(u_k) \to \mathcal{E}_\circ = \inf_v \mathcal{E}(v)$ as $k \to 0$. From 11.39 we have for all k

$$\frac{\lambda}{2}\|\nabla u_k\|_{L^2(\Omega)}^2 + \left(\frac{\lambda}{2C_p} - \frac{1}{C}\right)\|u_k\|_{L^2(\Omega)}^2 - C\|f\|_{L^2(\Omega)}^2 \leq \mathcal{E}(u_k).$$

Therefore since $\mathcal{E}(u_k)$ is bounded, we have that $\|u_k\|_{H_0^1(\Omega)}$ is uniformly bounded, and thus we can extract a weakly convergent subsequence. With some abuse of notations, let us without loss of generality assume that $u_k \rightharpoonup u$.

We will now show that if $u_k \rightharpoonup u$,

$$\mathcal{E}(u) \leq \liminf_{k\to\infty} \mathcal{E}(u_k) = \mathcal{E}_\circ$$

and therefore conclude that the limit u is a minimizer. This property is also referred to as weak-lower semi-continuity of $\mathcal{E}$.

In order to show the weak-lower semicontinuity of $\mathcal{E}$ we define the following set,

$$\mathcal{A}(t) := \{v \in H_0^1(\Omega) : \mathcal{E}(v) \leq t\}.$$

Furthermore, note that the functional $\mathcal{E}(v)$ is convex in v (since the function L is convex and the term $f(x)v(x)$ is linear), and this also implies that the set $\mathcal{A}(t)$ is convex.

Further, for any sequence of functions $\{w_k\}$ where $w_k \in \mathcal{A}(t)$ such that $w_k \to w$ from Fatou's Lemma,

$$\mathcal{E}(w) = \int_\Omega L(x, w(x), \nabla w(x)) - f(x)w(x)dx \leq \liminf_{k\to\infty} \int_\Omega L(x, w_k(x), \nabla w_k(x)) - f(x)w_k(x)dx \leq t$$

hence we also have that the function $w \in \mathcal{A}(t)$. Therefore the set $\mathcal{A}(t)$ is closed (w.r.t $H_0^1(\Omega)$ norm), and it is convex. Since the set $A(t)$ is closed and convex (it is also weakly closed) therefore if $w_k \to w$ it also implies that $w_k \rightharpoonup w$ in $H_0^1(\Omega)$.

Hence, consider a weakly converging sequence in $H_0^1(\Omega)$, i.e., $w_k \rightharpoonup w$ and define

$$t^* := \liminf_{k\to\infty} \mathcal{E}(w_k)$$

Now, for any $\varepsilon > 0$, there exists a subsequence $w_{k_{j,\varepsilon}} \rightharpoonup w$ in $H_0^1(\Omega)$ and $\mathcal{E}_{w_{k_{j,\varepsilon}}} \leq t^\star + \varepsilon$, that is, $w_{k_{j,\varepsilon}} \in \mathcal{A}(t^* + \varepsilon)$. This this is true for all $\epsilon > 0$ this implies that $\mathcal{E}(w) \leq t^* = \liminf_{k\to 0} \mathcal{E}$. Hence the function $\mathcal{E}$ is lower-semi-continuous, and hence the minimizer exists!

Now to show that the minimum is unique. Note the function $\mathcal{E}$ is convex in u. We will prove that the minima is unique by contradiction.

Let $u, v \in H_0^1(\Omega)$ be two (distinct) minima of $\mathcal{E}$, i.e., we have, $\mathcal{E}(u) = \mathcal{E}_\circ$ and $\mathcal{E}(v) = \mathcal{E}_\circ$.

Now using the fact that the function $L : \Omega \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ is convex, and the minimality of $\mathcal{E}_\circ$, we have for all $x \in \Omega$ we have

$$\begin{aligned}
\mathcal{E}_\circ \leq \mathcal{E}\left(\frac{u+v}{2}\right) &= \int_\Omega L\left(x, \frac{u(x)+v(x)}{2}, \frac{\nabla u(x)+\nabla v(x)}{2}\right) + f(x)\frac{u(x)+v(x)}{2} \\
&= \int_\Omega L\left(\frac{x+x}{2}, \frac{u(x)+v(x)}{2}, \frac{\nabla u(x)+\nabla v(x)}{2}\right) + f(x)\frac{u(x)+v(x)}{2} \\
&\leq \int_\Omega \frac{1}{2}(L(x,u(x),\nabla u(x)) + u(x)) + \int_\Omega \frac{1}{2}(L(x,v(x),\nabla v(x)) + v(x)) \\
&\leq \frac{1}{2}\mathcal{E}(u) + \frac{1}{2}\mathcal{E}(v) \\
\implies \mathcal{E}_\circ \leq \mathcal{E}\left(\frac{u+v}{2}\right) &\leq \frac{1}{2}\mathcal{E}(u) + \frac{1}{2}\mathcal{E}(v) = \mathcal{E}_\circ.
\end{aligned}$$

The last inequality is a contradiction and therefore the minima is unique. □

11.4.2 Proof of Lemma 15: Nonlinear Elliptic Variational PDEs

Proof of Lemma 15. If the function $u^\star$ minimizes the energy functional in Definition 11 then we have for all $\epsilon \in \mathbb{R}$

$$\mathcal{E}(u) \leq \mathcal{E}(u + \epsilon\varphi)$$

where $\varphi \in C_c^\infty(\Omega)$. That is, we have a minima at $\epsilon = 0$ and taking a derivative w.r.t ϵ and using Taylor expansion we get,

$$\begin{aligned}
d\mathcal{E}[u](\varphi) &= \lim_{\epsilon\to 0} \frac{\mathcal{E}(u+\epsilon\varphi) - \mathcal{E}(u)}{\epsilon} = 0 \\
&= \lim_{\epsilon\to 0} \frac{\int_\Omega L(x, u+\epsilon\varphi, \nabla u + \epsilon\nabla\varphi) - f(x)(u(x)+\epsilon\varphi(x)) - L(x,u,\nabla u) + f(x)u(x)\,dx}{\epsilon} \\
&= \lim_{\epsilon\to 0} \frac{\int_\Omega L(x, u+\epsilon\varphi, \nabla u) + \partial_{\nabla u}L(x, u+\epsilon\varphi, \nabla u) + r_1(x) - \epsilon f(x)\epsilon\varphi(x) - L(x,u,\nabla u)\,dx}{\epsilon} \\
&= \lim_{\epsilon\to 0} \frac{\int_\Omega L(x,u,\nabla u) + \epsilon\partial_u L(x,u,\nabla u)\varphi + r_2(x)}{\epsilon} \\
&\quad + \lim_{\epsilon\to 0} \frac{\epsilon\partial_{\nabla u}L(x,u,\nabla u)\nabla\varphi + \epsilon^2\partial_u\partial_{\nabla u}L(x,u,\nabla u)\nabla\varphi\cdot\varphi + r_1(x) - \epsilon f(x)\epsilon\varphi(x) - L(x,u,\nabla u)\,dx}{\epsilon} \\
&= \lim_{\epsilon\to 0} \frac{\int_\Omega \epsilon\partial_{\nabla u}L(x,u,\nabla u)\nabla\varphi + \epsilon\partial_u L(x,u,\nabla u)u + r_1(x) + r_2(x) - \epsilon f(x)\varphi(x)\,dx}{\epsilon}
\end{aligned} \tag{11.40}$$

where for all $x \in \Omega$ we have,

$$\begin{aligned}
|r_1(x)| &\leq \frac{\epsilon^2}{2}\sup_{y\in\Omega}\left|\left((\nabla u(x))^T\partial^2_{\nabla u}L(y, u+\epsilon\varphi, \nabla u)\nabla u(x)\right)\right| \\
&\leq \frac{\Lambda\epsilon^2}{2}\|\nabla u(x)\|_2^2
\end{aligned} \tag{11.41}$$

Similarity we have,

$$|r_2(x)| \leq \frac{\epsilon^2}{2} \sup_{y \in \Omega} \left| \partial_u L(y, u, \nabla u) u(x)^2 \right|$$
$$\leq \frac{\Lambda \epsilon^2}{2} u(x)^2 \tag{11.42}$$

Using results from Equation 11.40 and Equation 11.42 in Equation Equation 11.41 and taking $\epsilon \to 0$, the derivative in the direction of φ is,

$$d\mathcal{E}[u](\varphi) = \lim_{\epsilon \to 0} \frac{\int_\Omega \partial_{\nabla u} L(x, u, \nabla u) \nabla \varphi + \partial_u L(x, u, \nabla u) u - f(x)\varphi(x)\, dx}{\epsilon}$$

Since $\epsilon \to 0$ the final derivative is of the form,

$$d\mathcal{E}[u](\varphi) = \int_\Omega \Big(\partial_{\nabla u} L(x, u, \nabla u) \nabla \varphi + \partial_u L(x, u, \nabla v) \varphi - f\varphi \Big) dx = 0. \tag{11.43}$$

We will now use the following integration by parts identity, for functions $r : \Omega \to \mathbb{R}$ such that and $s : \Omega \to \mathbb{R}$, and $r, s \in H_0^1(\Omega)$,

$$\int_\Omega \frac{\partial r}{\partial x_i} s dx = -\int_\Omega r \frac{\partial s}{\partial x_i} dx + \int_{\partial \Omega} r s n d\Gamma \tag{11.44}$$

where n_i is a normal at the boundary and $d\Gamma$ is an infinitesimal element of the boundary $\partial\Omega$.

Using the identity in Equation 11.44 in Equation 11.43 we get,

$$\begin{aligned}
d\mathcal{E}[u](\varphi) &= \int_\Omega \Big(\partial_{\nabla u} L(x, u, \nabla u) \nabla \varphi + + \partial_u L(x, u, \nabla v) \varphi - f\varphi \Big) dx \\
&= \int_\Omega \Big(\sum_{i=1}^d (\partial_{\nabla u} L(x, u, \nabla u))_i \partial_i \varphi + \partial_u L(x, u, \nabla v) \varphi - f\varphi \Big) dx \\
&= \int_\Omega \Big(\sum_{i=1}^d -\partial_i (\partial_{\nabla u} L(x, u, \nabla u))_i \varphi + \partial_u L(x, u, \nabla v) \varphi - f\varphi \Big) dx \\
&= \int_\Omega \Big(-\nabla_x \cdot (\partial_{\nabla u} L(x, u, \nabla u)) \varphi + \partial_u L(x, u, \nabla u) \varphi - f\varphi \Big) dx = 0 \\
\implies d\mathcal{E}[u](\varphi) &= \int_\Omega \Big(-\operatorname{div}_x (\partial_{\nabla u} L(x, u, \nabla u)) \varphi + \partial_u L(x, u, \nabla u) \varphi - f\varphi \Big) dx = 0
\end{aligned}$$

That is the minima for the energy functional is reached at a u which solves the following PDE,

$$d\mathcal{E}(u) := -\operatorname{div}_x (\partial_{\nabla u} L(x, u, \nabla u)) + \partial_u L(x, u, \nabla u) = f.$$

where we define $d\mathcal{E}(\cdot)$ as the operator $-\operatorname{div}_x (\partial_{\nabla u} L(x, \cdot, \nabla \cdot)) + \partial_u L(x, \cdot, \nabla \cdot)$.

□

11.4.3 Proof of Lemma 16: Poincare constant of Unit Hypercube

Proof of Lemma 16. We use the fact that the Poincare constant is the smallest eigenvalue of Δ, i.e.,

$$\frac{1}{C_p} := \inf_{u \in L^2(\Omega)} \frac{\|\Delta u\|_{L^2(\Omega)}}{\|u\|_{L^2(\Omega)}}.$$

Note that the eigenfunctions of Δ for the domain $\Omega := [0,1]^d$ are defined as

$$\phi_\omega(x) = \prod_{i=1}^{d} \sin(\pi i \omega_i x_i), \quad \forall \omega \in \mathbb{N}^d \;\; \& \;\; x \in \Omega.$$

Furthermore, this also implies that for all $\omega \in \mathbb{N}^d$ we have,

$$\Delta \phi_\omega = \pi^2 \|\omega\|_2^2 \phi_\omega.$$

We can expand any function $u \in H_0^1(\Omega)$ in terms of ϕ_ω as $u(x) = \sum_{\omega \in \mathbb{N}^d} d_\omega \phi_\omega(x)$ where $d_\omega = \langle u, \phi_\omega \rangle_{L^2(\Omega)}$.

Note that for all $x \in \Omega$, we have,

$$\Delta u(x) = \sum_{\omega \in \mathbb{N}^d} \pi^2 \|\omega\|_2^2 d_\omega \phi_\omega(x).$$

Taking square $L^2(\Omega)$ norm on both sides, we get,

$$\begin{aligned}
\|\Delta u\|_{L^2(\Omega)}^2 &= \pi^4 \left\| \sum_{\omega \in \mathbb{N}^d} \|\omega\|_2^2 d_\omega \phi_\omega \right\|_{L^2(\Omega)}^2 \\
&\overset{(i)}{\geq} \pi^4 d^2 \left\| \sum_{\omega \in \mathbb{N}^d} d_\omega \phi_\omega \right\|_{L^2(\Omega)}^2 \\
&\overset{(ii)}{=} \pi^4 d^2 \|u\|_{L^2(\Omega)}^2 \\
\implies \frac{\|\Delta u\|_{L^2(\Omega)}}{\|u\|_{L^2(\Omega)}} &\geq \pi^2 d
\end{aligned}$$

where we use the fact that $\|\omega\|_2 \geq \sqrt{d}$ (since $\forall i \in [d]$ we have $\omega_i \in \mathbb{N}$) in step (i), and use the orthogonality of $\{\phi_\omega\}_{\omega \in \mathbb{N}^d}$ in (ii). Moreover, it's easy to see that equality can be achieved by taking $u = \phi_{(1,1,\ldots,1)}$.

Hence the Poincare constant can be calculated as,

$$\begin{aligned}
\frac{1}{C_p} := \inf_{u \in L^2(\Omega)} \frac{\|\Delta u\|_{L^2(\Omega)}}{\|u\|_{L^2(\Omega)}} &= \pi^2 d \\
\implies C_p &= \frac{1}{\pi^2 d}.
\end{aligned}$$

□

11.5 Important Helper Lemmas

Lemma 38. *The dual norm of* $\|\cdot\|_{H_0^1(\Omega)}$ *is* $\|\cdot\|_{H_0^1(\Omega)}$.

Proof. If $\|u\|_*$ denotes the dual norm of $\|u\|_{H_0^1(\Omega)}$, by definition we have,

$$\begin{aligned}\|u\|_* &= \sup_{\substack{v\in H_0^1(\Omega)\\ \|v\|_{H_0^1(\Omega)}=1}} \langle u, v\rangle_{H_0^1(\Omega)}\\ &= \sup_{\substack{v\in H_0^1(\Omega)\\ \|v\|_{H_0^1(\Omega)}=1}} \langle \nabla u, \nabla v\rangle_{L^2(\Omega)}\\ &\leq \sup_{\substack{v\in H_0^1(\Omega)\\ \|v\|_{H_0^1(\Omega)}=1}} \|\nabla u\|_{L^2(\Omega)}\|\nabla v\|_{L^2(\Omega)}\\ &= \|\nabla u\|_{L^2(\Omega)}\end{aligned}$$

where the inequality follows by Cauchy- Schwarz. On the other hand, equality can be achieved by taking $v = \frac{u}{\|\nabla u\|_2}$. Thus, $\|u\|_* = \|\nabla u\|_{L^2(\Omega)} = \|u\|_{H_0^1(\Omega)}$ as we wanted. □

11.5.1 Useful properties of Laplacian and Laplacian Inverse

Lemma 39. *The operator* $(-\Delta)^{-1}$ *is self-adjoint.*

Proof. Note that since the operator $(-\Delta)^{-1}$ is bounded, to show that it is self-adjoint, we only need to show that the operator is also symmetric, i.e., for all $u, v \in H_0^1(\Omega)$ we have

$$\langle(-\Delta)^{-1}u, v\rangle_{L^2(\Omega)} = \langle u, (-\Delta)^{-1}v\rangle_{L^2(\Omega)}.$$

To show this, we first show that the operator Δ is symmetric. i.e, we have

$$\langle -\Delta u, v\rangle_{L^2(\Omega)} = \langle u, -\Delta v\rangle_{L^2(\Omega)} \tag{11.45}$$

This is a direct consequence of the Green's Identity where for functions $u, v \in C_0^\infty$ the following holds,

$$\begin{aligned}\int_\Omega -(\Delta u)v dx &= \int_\Omega \nabla u\cdot\nabla v dx + \int_{\partial\Omega}\frac{\partial u}{\partial n}v d\Gamma\\ &= \int_\Omega \nabla u\cdot\nabla v dx\\ &= -\int_\Omega u\Delta v dx + \int_{\partial\Omega}\frac{\partial v}{\partial n}u d\Gamma\end{aligned}$$

where we use the fact that since $u, v \in H_0^1(\Omega)$ we have $u(x) = 0$ and $v(x) = 0$ for all $x \in \partial\Omega$.

Now, taking $\tilde{u} = -\Delta u$ and $\tilde{v} = (-\Delta)^{-1} v$ from Equation Equation 11.45 we get,

$$\langle -\Delta u, v\rangle_{L^2(\Omega)} = \langle u, \Delta v\rangle_{L^2(\Omega)}$$
$$\langle \tilde{u}, (-\Delta)^{-1}\tilde{v}\rangle_{L^2(\Omega)} = \langle (-\Delta)^{-1}\tilde{u}, \tilde{v}\rangle_{L^2(\Omega)}.$$

Hence we have that the operator $(-\Delta)^{-1}$ is symmetric and bounded and therefore is self-adjoint. □

Lemma 40. *Given a vector valued function $f : \mathbb{R}^d \to \mathbb{R}^d$, such that $f \in C^2$ the following identity holds,*

$$\nabla \mathrm{div}_{\mathrm{x}}(f) = \mathrm{div}_{\mathrm{x}}(\nabla f). \tag{11.46}$$

Proof. We first simplify the right hand side of Equation Equation 11.46. Note that since $\nabla f : \mathbb{R}^d \to \mathbb{R}^{d\times d}$ is a is a matrix valued function the divergence of ∇f is going to be vector valued. More precisely for all $x \in \Omega$, $-\mathrm{div}_{\mathrm{x}}(\nabla f)$ is defined as

$$\begin{aligned}\mathrm{div}_{\mathrm{x}}(\nabla f(x)) &= \left[\sum_{j=1}^{d} \partial_j [\nabla f(x)]_i\right]_{i=1}^{d} \\ &= \left[\sum_{j=1}^{d} \partial_j \partial_i f(x)\right]_{i=1}^{d}\end{aligned} \tag{11.47}$$

where for a vector valued function the notation $[g(x)]_i$ denotes its i^{th} coordinate, and the notation $[g(x)]_{i=1}^{d} := (g(x)_1, g(x)_2, \cdots, g(x)_d)$ denotes a d dimensional vector.

Now, simplifying the left hand side, for all $x \in \Omega$ we get,

$$\begin{aligned}\nabla \mathrm{div}_{\mathrm{x}}(f(x)) &= \nabla\left(\sum_{j=1}^{d} \partial_j f(x)\right) \\ &= \left[\partial_i\left(\sum_{j=1}^{d} \partial_j f(x)\right)\right]_{i=1}^{d} \\ &= \left[\left(\sum_{j=1}^{d} \partial_i \partial_j f(x)\right)\right]_{i=1}^{d}\end{aligned} \tag{11.48}$$

Since the term in Equation 11.47 is equal to Equation 11.48 we have $\nabla \mathrm{div}_{\mathrm{x}}(f) = \mathrm{div}_{\mathrm{x}}(\nabla f)$. □

Lemma 41. *For a function $g : \mathbb{R}^d \to \mathbb{R}$ such that $g \in C^3$ the following identity holds,*

$$\Delta \nabla g = \nabla \Delta g$$

Proof. The term $\Delta\nabla g$ can be simplified as follows,

$$\begin{aligned}\Delta\nabla g &= \Delta\left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_d}\right) \\ &= \Delta\left[\frac{\partial f}{\partial x_i}\right]_{i=1}^{d} \\ &= \left[\Delta\frac{\partial f}{\partial x_i}\right]_{i=1}^{d} \\ &= \left[\sum_{j=1}^{d}\frac{\partial}{\partial x_j^2}\frac{\partial f}{\partial x_i}\right]_{i=1}^{d} \\ &= \left[\sum_{j=1}^{d}\frac{\partial^2 f}{\partial x_j^2 \partial x_i}\right]_{i=1}^{d} \end{aligned} \tag{11.49}$$

Further, $\nabla\Delta g$ can be simplified as follows,

$$\begin{aligned}\nabla\Delta g &= \nabla\left(\sum_{j=1}^{d}\frac{\partial g}{\partial x_j^2}\right) \\ &= \left[\sum_{j=1}^{d}\frac{\partial}{\partial x_1}\frac{\partial g}{\partial x_j^2}, \sum_{j=1}^{d}\frac{\partial}{\partial x_2}\frac{\partial g}{\partial x_j^2}, \cdots, \sum_{j=1}^{d}\frac{\partial}{\partial x_d}\frac{\partial g}{\partial x_j^2},\right] \\ &= \left[\sum_{j=1}^{d}\frac{\partial^2 g}{\partial x_i \partial x_j^2},\right]_{i=1}^{d} \end{aligned} \tag{11.50}$$

Since Equation 11.49 is equal to Equation 11.50 it implies that

$$\Delta\nabla g = \nabla\Delta g.$$

□

Corollary 3. *For all vector valued function $f : \mathbb{R}^d \to \mathbb{R}^d$ functions the following holds,*

$$\nabla(-\Delta)^{-1}\mathrm{div}_{\mathrm{x}}(f) = (-\Delta)^{-1}\mathrm{div}_{\mathrm{x}}(\nabla f). \tag{11.51}$$

Proof. We know from Lemma 40 that for a vector valued function $f : \mathbb{R}^d \to \mathbb{R}^d$ that we have

$$\nabla\mathrm{div}_{\mathrm{x}}(f) = \mathrm{div}_{\mathrm{x}}(\nabla f).$$

Now, using for a fact that any function g can be written as, $g = (-\Delta)(-\Delta)^{-1}g$ we get,

$$\begin{aligned} &\nabla\mathrm{div}_{\mathrm{x}}(f) = \mathrm{div}_{\mathrm{x}}(\nabla f) \\ \implies &\nabla(-\Delta)(-\Delta)^{-1}\mathrm{div}_{\mathrm{x}}(f) = \mathrm{div}_{\mathrm{x}}(\nabla f) \\ \overset{(i)}{\implies} &(-\Delta)\nabla(-\Delta)^{-1}\mathrm{div}_{\mathrm{x}}(f) = \mathrm{div}_{\mathrm{x}}(\nabla f) \\ \implies &\nabla(-\Delta)^{-1}\mathrm{div}_{\mathrm{x}}(f) = (-\Delta)^{-1}\mathrm{div}_{\mathrm{x}}(\nabla f) \end{aligned}$$

where (i) follows from Lemma 41, i.e., for any function $g \in C^3$, we have, $\nabla\Delta g = \Delta\nabla g$. □

11.5.2 Some properties of Sub-Matrices

Lemma 42. *Given matrices $A \in \mathbb{R}^{d\times d}$ and $B \in \mathbb{R}^{d\times d}$ if we have $A \preceq B$ then for any set of indices $U \subseteq \{1, 2, \cdots d\}$ where $|U| = n \leq d$ then for all $y \in \mathbb{R}^n$ we have $y^T A_U y \leq y^T B_U y$. where $A_U = A_{i,j}$ for all $i, j \in U$. Similarly if if we have $A \succeq B$ for all $y \in \mathbb{R}^n$ we have, $y^T A_U y \geq y^T B_U y$.*

Proof. We will show that $A \preceq B \implies A_U \preceq B_U$. The proof for $A \succeq B \implies A_U \succeq B_U$ will follow similarly.

Without loss of generality we can assume that $U = \{1, 2, \cdots n\}$ and a set $V = \{n, \cdots d\}$, where $n \leq d$. Since $A \preceq B$ we know that there exists $x \in \mathbb{R}^d$ we have $x^T Ax \leq x^T Bx$.

For all $y \in \mathbb{R}^d$ define $x := (y, \mathbf{0}_{d-n})$, and let $A_{U,V} = A_{i,j}$ be $i \in U$ and $j \in V$

$$\begin{aligned}
&\begin{bmatrix} y & \mathbf{0} \end{bmatrix}^T \begin{bmatrix} A_U & A_{U,V} \\ A_{V,U} & A_V \end{bmatrix} \begin{bmatrix} y & \mathbf{0} \end{bmatrix}^T \leq \begin{bmatrix} y & \mathbf{0} \end{bmatrix}^T \begin{bmatrix} B_U & B_{U,V} \\ B_{V,U} & B_V \end{bmatrix} \begin{bmatrix} y & \mathbf{0} \end{bmatrix}^T \\
\implies &\begin{bmatrix} y & \mathbf{0} \end{bmatrix}^T \begin{bmatrix} B_U - A_U & B_{U,V} - A_{U,V} \\ B_{V,U} - A_{V,U} & B_V - A_V \end{bmatrix} \begin{bmatrix} y & \mathbf{0} \end{bmatrix}^T \geq 0 \\
\implies &y^T (B_U - A_U) y \geq 0
\end{aligned}$$

Since we have for all $y \in \mathbb{R}^n$ we have $y^T(B_U - A_U)y \geq 0$, therefore this implies that $A_U \preceq B_U$. □

12 Appendix for Chapter 5

12.1 Implementation Details

Training details. We train all the networks for 500 epochs with Adam optimizer. The learning rate is set to 0.001 for Darcy flow and 0.005 for Navier-Stokes. We use learning rate weight decay of 1e-4 for both Navier-Stokes and Darcy flow. The batch size is set to 32. In case of Darcy flow, we also use cosine annealing for learning rate scheduling. We run all our experiments on a combination of NVIDIA RTX A6000, NVIDIA GeForce RTX 2080 Ti and 3080 Ti. All networks can easily fit on a single NVIDIA RTX A6000, but training time varies between the networks.

For FNO-DEQ, we use Anderson solver [Anderson, 1965] to solve for the fixed point in the forward pass. The maximum number of Anderson solver steps is kept fixed at 32 for Dary Flow, and 16 for Navier Stokes. For the backward pass, we use phantom gradients [Geng et al., 2021] which are computed as:

$$u^\star = \tau G_\theta(u^\star, a) + (1-\tau)u^\star \tag{12.1}$$

where τ is a tunable damping factor and $u^\star$ is the fixed point computed using Anderson solver in the forward pass. This step can be repeated S times. We use $\tau = 0.5$ and $S = 1$ for Darcy Flow, and $\tau = 0.8$ and $S = 3$ for Navier-Stokes.

For the S-FNO-DEQ used in Table 12.5, we use Broyden's method [Broyden, 1965] to solve for the fixed point in the forward pass and use exact implicit gradients, computed through implicit function theorem as shown in Equation 5.6, for the backward pass through DEQ. The maximum number of solver steps is fixed at 32.

For weight-tied networks, we repeatedly apply the FNO block to the input 12 times for Darcy flow, and 6 times for Navier-Stokes.

Network architecture details. The width of an FNO layer set to 32 across all the networks. Additionally, we retain only 12 Fourier modes in FNO layer, and truncate higher Fourier modes. We use the code provided by Li et al. [2020a] to replicate the results for FNO, and construct rest of the networks on top of this as described in Section 9.8.

12.2 Datasets

12.2.1 Darcy Flow

As mentioned in Section 9.8 we use the dataset provided by Li et al. [2020a] for our experiments with steady-state Darcy-Flow.

All the models are trained on 1024 data samples and tested on 500 samples. The resolution of original images is 421×421 which we downsample to 85×85 for our experiments. For experiments with noisy inputs/observations, the variance of Gaussian noise that we add to PDEs are [0, 1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3].

12.2.2 Steady-State Incompressible Fluid Navier-Stoke

$$\begin{aligned} u \cdot \nabla \omega &= \nu \Delta \omega + f, \qquad && x \in \Omega \\ \nabla \cdot u &= 0 && x \in \Omega \end{aligned}$$

To generate the dataset for steady-state Navier-Stokes, instead of solving the steady state PDE using steady-state solvers like the SIMPLE algorithm Patankar and Spalding [1983], we first choose the solution $\omega^\star := \nabla \times u^\star$ of the PDE and then generate the corresponding equation, i.e. calculate the corresponding force term $f = u^\star \cdot \nabla \omega^\star - \nu \Delta \omega^\star$.

To generate the solutions $\omega^\star$, we forward propagate a relatively simple initial distribution of ω_0 (sampled from a Gaussian random field) through a time-dependent Navier-Stokes equation in the vorticity form for a short period of time. This ensures our dataset contains solutions ω^* that are rich and complex. Precisely, recall the Navier-Stokes equations in their vorticity form:

$$\begin{aligned} \partial_t \omega(x,t) + u(x,t) \cdot \nabla \omega(x,t) &= \nu \Delta \omega(x,t) + g(x) \qquad x \in (0, 2\pi)^2, t \in [0, T] \\ \nabla \cdot u(x,t) &= 0 \qquad x \in (0, 2\pi)^2, t \in [0, T] \\ \omega(x, 0) &= \omega_0(x) \qquad x \in (0, 2\pi)^2 \end{aligned} \tag{12.2}$$

where $g(x) = \nabla \times \tilde{g}(x)$ and $\tilde{g}(x) = \sin(5x_1)\hat{x_2}$ is a divergence free forcing term and $x = (x_1, x_2)$ are the two coordinates of the input vector. We forward propagate the equations Equation 12.2 using a pseudo-spectral method using the functions provided in JAX-CFD [Kochkov et al., 2021, Dresdner et al., 2022] package. The initial vorticity ω_0 is sampled from a Gaussian random field $\mathcal{N}(0, (5^{3/2}(I + 25\Delta)^{-2.5}))$, which is then made divergence free. We forward propagate the Navier-Stokes equation in Equation 12.2 for time $T = 0.5$ with $dt = 0.002$ to get $\omega(1, x)$, which we choose as the solution to the steady-state PDE in Equation 5.10, i.e, $\omega^\star$ for Equation 5.10.

Subsequently, we use the stream function Ψ [Batchelor and Batchelor, 1967] to calculate $u = (\partial\Psi/\partial x_1, \partial\Psi/\partial x_2)$ by solving the Poisson equation $\Delta\Psi = \omega$ in the Fourier domain. Furthermore, since $f = u^\star \cdot \nabla\omega^\star - \nu\Delta\omega^\star$ we use the stream function to calculate (f_1, f_2), i.e., the different components of the force term.

We use 4500 training samples and 500 testing samples. The input to the network is the vector field $\tilde{f} = (f_1, f_2)$ and we learn a map that outputs the vorticity $\omega^\star$. The resolution of grid used to generate the dataset is 256×256 which we downsample to 128×128 while training the models. For experiments

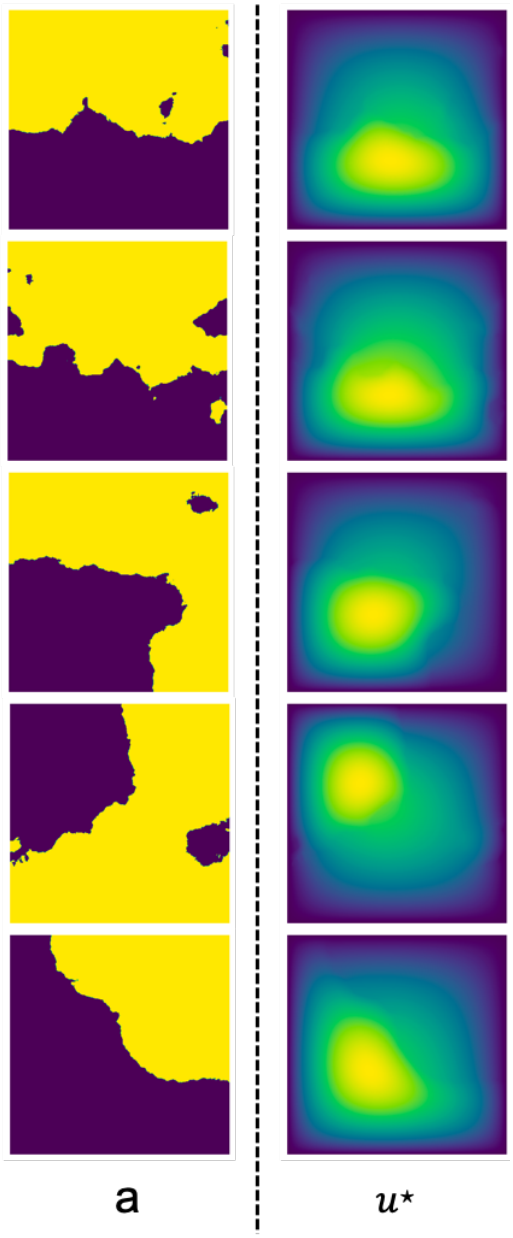


Figure 12.1: Samples from Darcy Flow

with noisy inputs/observations, we consider two values of maximum variance of Gaussian noise: 1e-3 and 4e-3. The variances of the Gaussian noise that we add to the PDEs for the latter case are [0, 1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 2e-3, 4e-3]. However, when conducting experiments with a variance of 1e-3, we exclude the last two values of variance from this list.

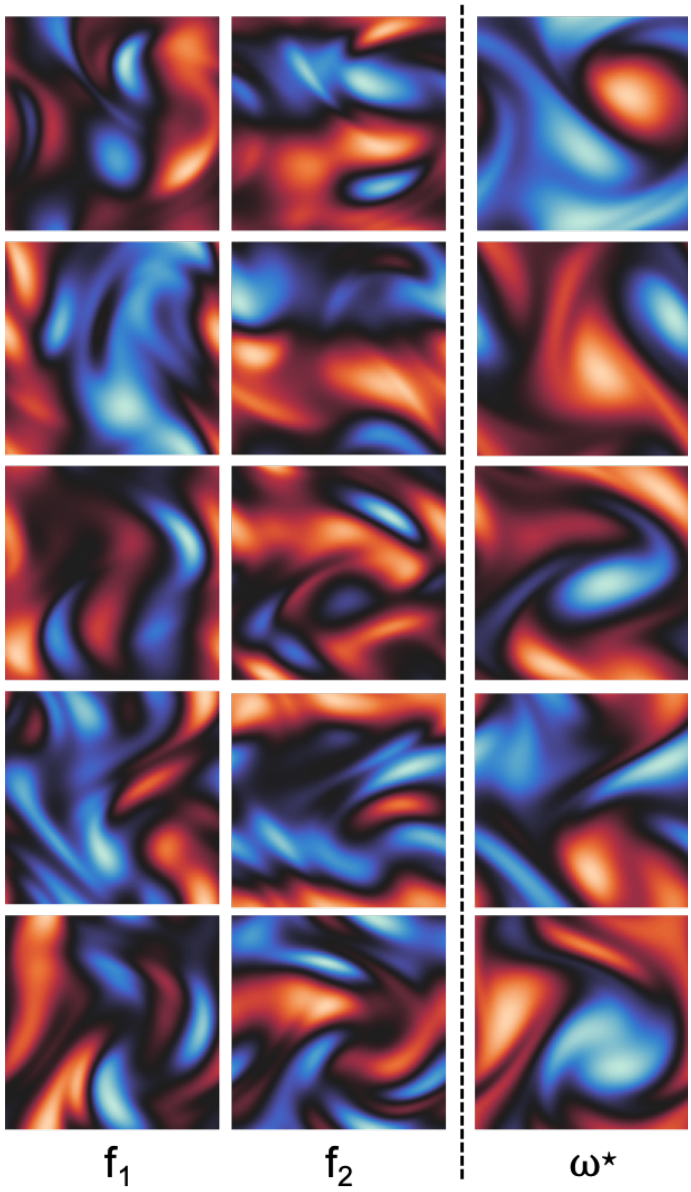


Figure 12.2: Samples from Steady-state Navier-Stokes dataset with viscosity 0.001. Each triplet visualizes the inputs f_1, f_2 and the ground truth output i.e. $\omega^\star$.

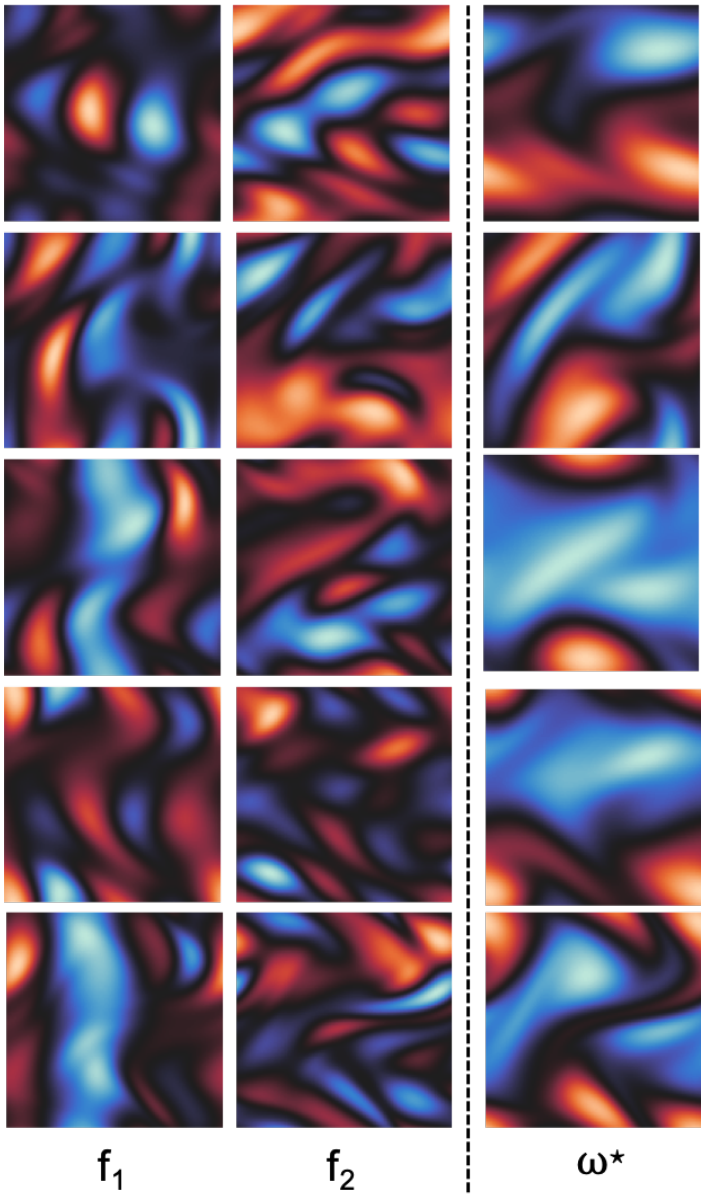


Figure 12.3: Samples from Steady-state Navier-Stokes dataset with viscosity 0.01. Each triplet visualizes the inputs f_1, f_2 and the ground truth output i.e. $\omega^\star$.

12.3 Proof of Universal Approximation

The proof of the universal approximation essentially follows from the result on the universal approximation capabilities of FNO layers in Kovachki et al. [2021a], applied to $\mathcal{G}(v, f) = v - (Lv - f)$. For the sake of completeness, we reitarate the key steps.

For simplicity, we will assume that $d_u = d_v = d_f = 1$. (The results straightforwardly generalize.) We will first establish some key technical lemmas and introduce some notation and definitions useful for the proof for Theorem 8.

Definition 37. *An operator $T : L^2(\Omega; \mathbb{R}) \to L^2(\Omega; \mathbb{R})$ is continuous at $u \in L^2(\Omega; \mathbb{R})$ if for every $\epsilon > 0$, there exists a $\delta > 0$, such that for all $v \in L^2(\Omega)$ with $\|u - v\|_{L^2(\Omega)} \leq \delta$, we have $\|L(u) - L(v)\|_{L^2(\Omega)} \leq \epsilon$.*

First, we approximate the infinite dimensional operator $\mathcal{G} : L^2(\Omega) \times L^2(\Omega) \to L^2(\Omega)$ by projecting the functions in $L^2(\Omega)$ to a finite-dimensional approximation $L^2_N(\Omega)$, and considering the action of the operator on this subspace. The linear projection we use is the one introduced in Equation 5.11. More precisely we show the following result,

Lemma 43. *Given a continuous operator $L : L^2(\Omega) \to L^2(\Omega)$ as defined in Equation 5.1, let us define an operator $\mathcal{G} : L^2(\Omega) \times L^2(\Omega) \to L^2(\Omega)$ as $\mathcal{G}(v, f) := v - (L(v) - f)$. Then, for every $\epsilon > 0$ there exists an $N \in \mathbb{N}$ such that for all v, f in any compact set $K \subset L^2(\Omega)$, the operator $\mathcal{G}_N = \Pi_N \mathcal{G}(\Pi_N v, \Pi_N f)$ is an ϵ-approximation of $\mathcal{G}(v, f)$, i.e., we have,*

$$\sup_{v,f \in K} \|\mathcal{G}(v, f) - \mathcal{G}_N(v, f)\|_{L^2(\Omega)} \leq \epsilon.$$

Proof. Note that for an $\epsilon > 0$ there exists an $N = N(\epsilon, d)$ such that for all $v \in K$ we have

$$\sup_{v \in K} \|v - \Pi_N v\|_{L^2(\Omega)} \leq \epsilon.$$

Therefore, using the definition of $\mathcal{G}_N$ we can bound the $L^2(\Omega)$ norm of the difference between $\mathcal{G}$ and $\mathcal{G}_N$ as follows,

$$\begin{aligned}
&\|\mathcal{G}(v, f) - \Pi_N \mathcal{G}(v_n, f_n)\|_{L^2(\Omega)} \\
&\leq \|\mathcal{G}(v, f) - \Pi_N \mathcal{G}(v, f)\|_{L^2(\Omega)} + \|\Pi_N \mathcal{G}(v, f) - \Pi_N \mathcal{G}(\Pi_N v, \Pi_N f)\|_{L^2(\Omega)} \\
&\leq \underbrace{\|\mathcal{G}(v, f) - \Pi_N \mathcal{G}(v, f)\|_{L^2(\Omega)}}_{I} + \underbrace{\|\mathcal{G}(v, f) - \mathcal{G}(\Pi_N v, \Pi_N f)\|_{L^2(\Omega)}}_{II}
\end{aligned}$$

We first bound the term I as follows:

$$\begin{aligned}
&\|\mathcal{G}(v, f) - \Pi_N \mathcal{G}(v, f)\|_{L^2(\Omega)} \\
&= \|v - (L(v) - f) - \Pi_N (v - (L(v) - f))\|_{L^2(\Omega)} \\
&= \|v - \Pi_N v\|_{L^2(\Omega)} + \|f - \Pi_N f\|_{L^2(\Omega)} + \|L(v) - \Pi_N L(v)\|_{L^2(\Omega)} \\
&= \epsilon + \epsilon + \|L(v) - \Pi_N L(v)\|_{L^2(\Omega)}
\end{aligned} \tag{12.3}$$

Since L is continuous, for all compact sets $K \subset L^2(\Omega)$, $L(K)$ is compact as well. This is because: (1) for any $u \in K$, $\|L(u)\|_{L^2(\Omega)}$ is finite; (2) for any $v \in K$, $\|L(v)\|_{L^2(\Omega)} \leq \|L(u)\|_{L^2(\Omega)} + C\|u - v\|_{L^2(\Omega)}$. Therefore, for every $\epsilon > 0$, there exists an $N \in \mathbb{N}$ such that

$$\sup_{v \in K} \|L(v) - \Pi_N L(v)\|_{L^2(\Omega)} \leq \epsilon.$$

Substituting the above result in Equation 12.3, we have

$$\|\mathcal{G}(v, f) - \Pi_N \mathcal{G}(v, f)\|_{L^2(\Omega)} \leq 3\epsilon. \tag{12.4}$$

Similarly, for all $v \in K$ where K is compact, we can bound Term II as following,

$$\begin{aligned}
&\|\mathcal{G}(v, f) - \mathcal{G}(\Pi_N v, \Pi_N f)\|_{L^2(\Omega)} \\
&\leq \|v - (L(v) - f) - \Pi_N v - (L(\Pi_N v) - \Pi_N f)\|_{L^2(\Omega)} \\
&\leq \|v - \Pi_N v\|_{L^2(\Omega)} + \|f - \Pi_N f\|_{L^2(\Omega)} + \|L(v) - L(\Pi_N v)\|_{L^2(\Omega)} \\
&\leq \epsilon + \epsilon + \|L(v) - L(\Pi_N v)\|_{L^2(\Omega)}.
\end{aligned} \tag{12.5}$$

Now, since $v \in K$ and $L : L^2(\Omega) \to L^2(\Omega)$ is a continuous operator, there exists a modulus of continuity (an increasing real valued function) $\alpha \in [0, \infty)$, such that for all $v \in K$, we have

$$\|L(v) - L(\Pi_N v)\|_{L^2(\Omega)} \leq \alpha\big(\|v - \Pi_N v\|_{L^2(\Omega)}\big)$$

Hence for every $\epsilon > 0$ there exists an $N \in \mathbb{N}$ such that,

$$\alpha(\|v - \Pi_N v\|_{L^2(\Omega)}) \leq \epsilon.$$

Plugging these bounds in Equation 12.5, we get,

$$\|\mathcal{G}(v, f) - \mathcal{G}(\Pi_N v, \Pi_N f)\|_{L^2(\Omega)} \leq 3\epsilon. \tag{12.6}$$

Therefore, combining Equation 12.4 and Equation 12.6 then for $\epsilon > 0$, there exists an $N \in \mathbb{N}$, such that for all $v, f \in K$ we have

$$\sup_{v, f \in K} \|\mathcal{G}(v, f) - \Pi_N \mathcal{G}(v_n, f_n)\|_{L^2(\Omega)} \leq 6\epsilon. \tag{12.7}$$

Taking $\epsilon' = 6\epsilon$ proves the claim. □

Proof of Theorem 8. For Lemma 43 we know that there exists a finite dimensional projection for the operator $\mathcal{G}$, defined as $\mathcal{G}_N(v, f)$ such that for all $v, f \in L^2(\Omega)$ we have

$$\|\mathcal{G}(v, f) - \mathcal{G}_N(v, f)\|_{L^2(\Omega)} \leq \epsilon.$$

Now using the definition of $\mathcal{G}_N(v, f)$ we have

$$\begin{aligned}
\mathcal{G}_N(v, f) &= \Pi_N \mathcal{G}(\Pi_N v, \Pi_N f) \\
&= \Pi_N v - (\Pi_N L(\Pi_N v) - \Pi_N f)
\end{aligned}$$

From Kovachki et al. [2021a], Theorem 2.4 we know that there exists an FNO network G_{θ^L} of the form defined in Equation 5.2 such that for all $v \in K$, where K is a compact set, there exists an ϵ^L we have

$$\sup_{v \in K} \|\Pi_N L(\Pi_N v) - G_{\theta^L}\|_{L^2(\Omega)} \leq \epsilon^L \tag{12.8}$$

Finally, note that from Lemma D.1 in Kovachki et al. [2021a], we have that for any $v \in K$, there exists an FNO layers $G_{\theta^f} \in L^2(\Omega)$ and $G_{\theta^v} \in L^2(\Omega)$ defined in Equation 5.3 such that

$$\sup_{v \in K} \|\Pi_N v - G_{\theta^v}\|_{L^2(\Omega)} \leq \epsilon^v \tag{12.9}$$

and

$$\sup_{f \in K} \|\Pi_N f - G_{\theta^f}\|_{L^2(\Omega)} \leq \epsilon^f \tag{12.10}$$

for $\epsilon^v > 0$ and $\epsilon^f > 0$.

Therefore there exists an $\tilde{\epsilon} >$ such that there is an FNO network $G_\theta : L^2(\Omega) \times L^2(\Omega) \to L^2(\Omega)$ where $\theta := \{\theta^L, \theta^v, \theta^f\}$ such that

$$\sup_{v \in K, f \in L^2(\Omega)} \|\mathcal{G}_N(v, f) - G_\theta(v, f)\|_{L^2(\Omega)} \leq \tilde{\epsilon} \tag{12.11}$$

Now, since we know that $u^\star$ is the fixed point of the operator $\mathcal{G}$ we have from Lemma 43 and Equation 12.11,

$$\begin{aligned} \|\mathcal{G}(u^\star, f) - G_\theta(u^\star, f)\|_{L^2(\Omega)} &\leq \|u^\star - \mathcal{G}_N(u^\star, f)\|_{L^2(\Omega)} + \|\mathcal{G}_N(u^\star, f) - G_\theta(u^\star, f)\|_{L^2(\Omega)} \\ &\leq \tilde{\epsilon} + \epsilon. \end{aligned}$$

□

12.4 Fast Convergence for Newton Method

Definition 38 (Frechet Derivative in $L^2(\Omega)$)**.** *For a continuous operator $F : L^2(\Omega) \to L^2(\Omega)$, the Frechet derivative at $u \in L^2(\Omega)$ is a linear operator $F'(u) : L^2(\Omega) \to L^2(\Omega)$ such that for all $v \in L^2(\Omega)$ we have*

$$\lim_{\|v\|_{L^2(\Omega)} \to 0} \frac{\|F(u+v) - F(u) - F'(u)(v)\|_{L^2(\Omega)}}{\|v\|_{L^2(\Omega)}} = 0.$$

Lemma 44. *Given the operator $L : L^2(\Omega) \to L^2(\Omega)$ with Frechet derivative L', such that for all $u, v \in L^2(\Omega)$, we have $\|L'(u)(v)\|_{L^2(\Omega)} \geq \lambda \|v\|_{L^2(\Omega)}$, then $L'(u)^{-1}$ exists and we have, for all $v_1, v_2 \in L^2(\Omega)$:*

1. $\|L'(u)^{-1}(v_1)\|_{L^2(\Omega)} \leq \frac{1}{\lambda}\|v_1\|_{L^2(\Omega)}$.
2. $\|v_1 - v_2\|_{L^2(\Omega)} \leq \frac{1}{\lambda}\|L(v_1) - L(v_2)\|_{L^2(\Omega)}$

Proof. Note that for all $u, v' \in L^2(\Omega)$ we have,

$$\|L'(u)v'\|_{L^2(\Omega)} \geq \lambda \|v'\|_{L^2(\Omega)}$$

Taking $v = L'(u)^{-1}(v')$, we have

$$\begin{aligned} \|L'(u)\big(L'(u)^{-1}(v)\big)\|_{L^2(\Omega)} &\geq \lambda \|L^{-1}(u)(v)\|_{L^2(\Omega)} \\ \implies \frac{1}{\lambda}\|v\|_{L^2(\Omega)} &\geq \|L^{-1}(u)(v)\|_{L^2(\Omega)}. \end{aligned}$$

For part 2, note that there exists a $c \in [0,1]$ such that

$$\|L(v_1) - L(v_2)\|_{L^2(\Omega)} \geq \inf_{c\in[0,1]} \|L'(cv_1 + (1-c)v_2)\|_2 \|v_1 - v_2\|_{L^2(\Omega)} \geq \lambda \|v_1 - v_2\|_{L^2(\Omega)}.$$

□

We now show the proof for Lemma 45. The proof is standard and can be found in Faragó and Karátson [2002], however we include the complete proof here for the sake of completeness.

We restate the Lemma here for the convenience of the reader.

Lemma 45 (Faragó and Karátson [2002], Chapter 5)**.** *Consider the PDE defined Definition 18, such that $d_u = d_v = d_f = 1$. such that $L'(u)$ defines the Frechet derivative of the operator L. If for all $u, v \in L^2(\Omega;\mathbb{R})$ we have $\|L'(u)v\|_{L^2(\Omega)} \geq \lambda \|v\|_{L^2(\Omega)}$ [1] and $\|L'(u) - L'(v)\|_{L^2(\Omega)} \leq \Lambda \|u - v\|_{L^2(\Omega)}$ for $0 < \lambda \leq \Lambda < \infty$, then for the Newton update, $u_{t+1} \leftarrow u_t - L'(u_t)^{-1}(L(u_t) - f)$, with $u_0 \in L^2(\Omega;\mathbb{R})$, there exists an $\epsilon > 0$, such that $\|u_T - u^\star\|_{L^2(\Omega)} \leq \epsilon$ if [2] $T \geq \log\left(\log\left(\frac{1}{\epsilon}\right) / \log\left(\frac{2\lambda^2}{\Lambda \|L(u_0) - f\|_{L^2(\Omega)}}\right)\right)$.*

Proof of Lemma 45. Re-writing the updates in Lemma 45 as,

$$u_{t+1} = u_t + p_t \tag{12.12}$$
$$L'(u_t)p_t = -(L(u_t) - f) \tag{12.13}$$

Now, upper bounding $L(u_{t+1}) - f$ for all $x \in \Omega$ we have,

$$\begin{aligned}
&L(u_{t+1}(x)) - f(x) \\
&= L(u_t(x)) - f(x) + \int_0^1 (L'(u_t(x) + t(u_{t+1}(x) - u_t(x))))(u_{t+1}(x) - u_t(x))dt \\
&= L(u_t(x)) - f(x) + L'(u_t(x))p_t(x) + \int_0^1 (L'(u_t(x) + t(u_{t+1}(x) - u_t(x))) - L'(u_t(x)))p_t(x)dt \\
&= \int_0^1 (L'(u_t(x) + t(u_{t+1}(x) - u_t(x))) - L'(u_t(x)))p_t(x)dt
\end{aligned}$$

where we use Equation 12.13 in the final step.

Taking $L^2(\Omega)$ norm on both sides and using the fact that $\|L'(u) - L'(v)\|_{L^2(\Omega)} \leq \Lambda \|u - v\|_{L^2(\Omega)}$, we have

$$\|L(u_{t+1}) - f\|_{L^2(\Omega)} \leq \int_0^1 \Lambda t \|u_{t+1} - u_t\|_{L^2(\Omega)} \|p_t\|_{L^2(\Omega)} dt$$

[1] We note that this condition is different from the condition on the inner-product in the submitted version of the paper, which had. $\langle L'(u), v\rangle_{L^2(\Omega)} \geq \lambda \|v\|_{L^2(\Omega)}$.

[2] We note that this rate is different from the one in the submitted version of the paper.

Noting that for all $x \in \Omega$, we have $u_{t+1} - u_t = p_t$, and using the fact that for all u, v $\|L'(u)^{-1}v\|_{L^2(\Omega)} \leq \frac{1}{\lambda}\|v\|_{L^2(\Omega)}$ we have, $\|L'(u_t)p_t\|_{L^2(\Omega)} \leq \frac{1}{\lambda}\|p_t\|_{L^2(\Omega)}$

$$\begin{aligned}
\|L(u_{t+1}) - f\|_{L^2(\Omega)} &\leq \int_0^1 \Lambda t \|u_{t+1} - u\|_{L^2(\Omega)} \|p_t\|_{L^2(\Omega)} dt \\
&\leq \Lambda/2 \|p_t\|^2_{L^2(\Omega)} \\
&\leq \Lambda/2 \| - L'(u_t)^{-1}(L(u_t) - f)\|^2_{L^2(\Omega)} \\
&\leq \frac{\Lambda}{2\lambda^2}\|L(u_t) - f)\|^2_{L^2(\Omega)}
\end{aligned}$$

where we use the result from Lemma 44 in the last step.

Therefore we have

$$\begin{aligned}
&\|L(u_{t+1}) - f\|_{L^2(\Omega)} \leq \left(\frac{\Lambda}{2\lambda^2}\right)^{2^t-1} (L(u_0) - f)^{2^t} \\
\implies &\|L(u_{t+1}) - f\|_{L^2(\Omega)} \leq \left(\frac{\Lambda}{2\lambda^2}\right)^{2^t-1} (L(u_0) - L(u^\star))^{2^t} \\
\implies &\|u_{t+1} - u^\star\|_{L^2(\Omega)} \leq \frac{1}{\lambda}\left(\frac{\Lambda}{2\lambda^2}\right)^{2^t-1} \|L(u_0) - L(u^\star)\|^{2^t}_{L^2(\Omega)}.
\end{aligned}$$

Therefore, if

$$\frac{\Lambda}{2\lambda^2}\|L(u_0) - L(u^\star)\|_{L^2(\Omega)} \leq 1,$$

then we have

$$\|u_{t+1} - u^\star\|_{L^2(\Omega)} \leq \epsilon,$$

for

$$T \geq \log\left(\log\left(\frac{1}{\epsilon}\right) / \log\left(\frac{2\lambda^2}{\Lambda\|L(u_0) - f\|_{L^2(\Omega)}}\right)\right).$$

□

12.5 Additional experimental results

We provide additional results for Navier-Stokes equation for noisy inputs and observations in Table 12.1 and Table 12.2. For these experiments, the maximum variance of Gaussian noise added to inputs and observations is 0.004. We observe that weight-tied FNO and FNO-DEQ outperform non-weight-tied architectures.

Convergence analysis of fixed point. We report variations in test error, absolute residual $\|G_\theta(\mathbf{z}_t) - \mathbf{z}_t\|_2$, and relative residual $\frac{\|G_\theta(\mathbf{z}_t) - \mathbf{z}_t\|_2}{\|\mathbf{z}_t\|_2}$ with an increase in the number of solver steps while solving for the fixed point in FNO-DEQ, for both Darcy Flow (See Table 12.3) and Steady-State Navier Stokes (See Table 12.4). We observe that all these values decrease with increase in the number of fixed point solver it-

Architecture	Parameters	#Blocks	Test error ↓		
			$\sigma^2_{\max} = 0$	$(\sigma^2_{\max})^i = 0.004$	$(\sigma^2_{\max})^t = 0.004$
FNO	2.37M	1	0.184 ± 0.002	0.238 ± 0.008	0.179 ± 0.004
FNO	4.15M	2	0.162 ± 0.024	0.196 ± 0.011	0.151 ± 0.010
FNO	7.71M	4	0.157 ± 0.012	0.216 ± 0.002	0.158 ± 0.009
FNO++	2.37M	1	0.199 ± 0.001	0.255 ± 0.002	0.197 ± 0.004
FNO++	4.15M	2	0.154 ± 0.005	0.188 ± 0.006	0.157 ± 0.006
FNO++	7.71M	4	0.151 ± 0.003	0.184 ± 0.008	0.147 ± 0.004
FNO-WT	2.37M	1	$\mathbf{0.123 \pm 0.004}$	0.141 ± 0.003	$\mathbf{0.125 \pm 0.007}$
FNO-DEQ	2.37M	1	$\mathbf{0.123 \pm 0.005}$	$\mathbf{0.139 \pm 0.007}$	0.127 ± 0.002

Table 12.1: **Results on incompressible Steady-State Navier-Stokes (viscosity=0.001)**: clean data (Col 4), noisy inputs (Col 5) and noisy observations (Col 6) with max variance of added noise being $(\sigma^2_{\max})^i$ and $(\sigma^2_{\max})^t$, respectively. Reported test error has been averaged on three different runs with seeds 0, 1, and 2. ‡ indicates that the network diverges during training for one of the seeds.

Architecture	Parameters	#Blocks	Test error ↓		
			$\sigma^2_{\max} = 0$	$(\sigma^2_{\max})^i = 0.004$	$(\sigma^2_{\max})^t = 0.004$
FNO	2.37M	1	0.181 ± 0.005	0.207 ± 0.003	0.178 ± 0.008
FNO	4.15M	2	0.138 ± 0.007	0.163 ± 0.003	0.137 ± 0.006
FNO	7.71M	4	0.152 ± 0.006	0.203 ± 0.055	0.151 ± 0.008
FNO++	2.37M	1	0.188 ± 0.002	0.217 ± 0.001	0.187 ± 0.005
FNO++	4.15M	2	0.139 ± 0.004	0.170 ± 0.005	0.138 ± 0.005
FNO++	7.71M	4	0.130 ± 0.005	0.168 ± 0.007	0.126 ± 0.007
FNO-WT	2.37M	1	0.089 ± 0.004	0.097 ± 0.008	$\mathbf{0.087 \pm 0.003}$
FNO-DEQ	2.37M	1	$\mathbf{0.085 \pm 0.005}$	$\mathbf{0.096 \pm 0.008}$	$\mathbf{0.087 \pm 0.004}$

Table 12.2: **Results on incompressible Steady-State Navier-Stokes (viscosity=0.01)**: clean data (Col 4), noisy inputs (Col 5) and noisy observations (Col 6) with max variance of added noise being $(\sigma^2_{\max})^i$ and $(\sigma^2_{\max})^t$, respectively. Reported test error has been averaged on three different runs with seeds 0, 1, and 2.
‡ indicates that the network diverges during training for one of the seeds.

erations and eventually saturate once we have a reasonable estimate of the fixed point. We observe that increasing the number of fixed point solver iterations results in a better estimation of the fixed point. For steady state PDEs, we expect the test error to reduce as the estimation of the fixed point improves. Furthermore, at inference time we observe that the test error improves (i.e. reduces) with increase in the number of fixed point solver iterations even though the FNO-DEQ is trained with fewer solver steps. For Navier-Stokes with viscosity 0.01, at inference time we get a test MSE loss of 0.0744 with 48 solver steps from 0.0847 when used with 24 solver steps.

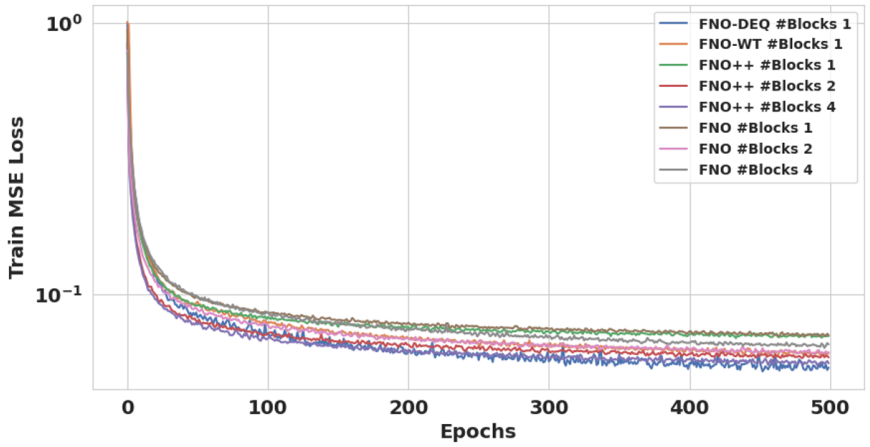
This further bolsters the benefits of DEQs (and weight-tied architectures in general) for training neural operators for steady-state PDEs. Moreover, performance saturates after a certain point once we have a reasonable estimate of the fixed point, hence showing that more solver steps stabilize to the same solution.

Solver steps	Absolute residual ↓	Relative residual ↓	Test Error ↓
2	212.86	0.8533	0.0777
4	18.166	0.0878	0.0269
8	0.3530	0.00166	0.00567
16	0.00239	1.13e-5	0.00566
32	0.000234	1.1e-6	0.00566

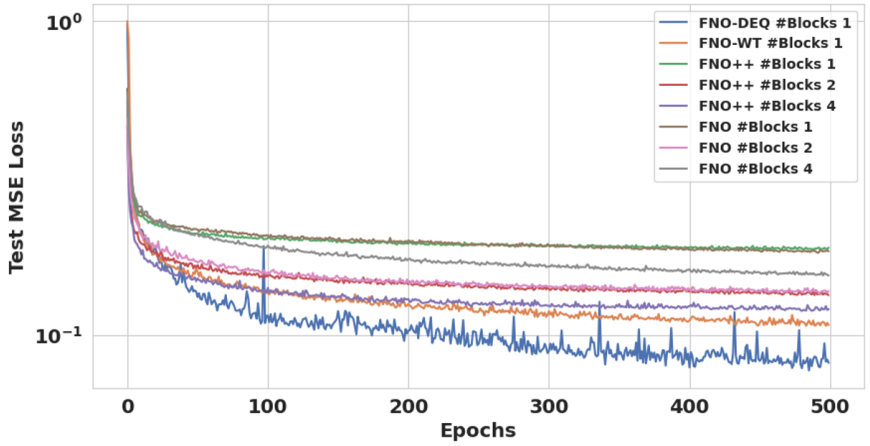
Table 12.3: Convergence analysis of fixed point for noiseless Darcy Flow: The test error, absolute residual $\|G_\theta(\mathbf{z}_t) - \mathbf{z}_t\|_2$ and relative residual $\frac{\|G_\theta(\mathbf{z}_t) - \mathbf{z}_t\|_2}{\|\mathbf{z}_t\|_2}$ decrease with increase in the number of fixed point solver iterations. The performance saturates after a certain point once we have a reasonable estimate of the fixed point. We consider the noiseless case, where we do not add any noise to inputs or targets.

Solver steps	Absolute residual ↓	Relative residual ↓	Test Error ↓
4	544.16	0.542	0.926
8	397.75	0.408	0.515
16	150.33	0.157	0.147
24	37.671	0.0396	0.0847
48	5.625	0.0059	0.0744
64	3.3	0.0034	0.0746

Table 12.4: Convergence analysis of fixed point for noiseless incompressible Steady-State Navier-Stokes with viscosity=0.01: The test error, absolute residual $\|G_\theta(\mathbf{z}_t) - \mathbf{z}_t\|_2$ and relative residual $\frac{\|G_\theta(\mathbf{z}_t) - \mathbf{z}_t\|_2}{\|\mathbf{z}_t\|_2}$ decrease with increase in the number of fixed point solver iterations. The performance saturates after a certain point once we have a reasonable estimate of the fixed point. We consider the noiseless case, where we do not add any noise to inputs or targets.



(a) Training Loss Curve



(b) Test Loss Curve

Figure 12.4: Training and Test Loss Curves for Steady-State Navier-Stokes with viscosity 0.01. The x axis is the number of epochs and y axis is the MSE loss in log scale. Note that while all the models converge to approximately the same MSE loss value while training, DEQs and weight-tied networks get a better test loss in fewer epochs.

Architecture	Parameters	#Blocks	Test error ↓		
			$\sigma^2_{\max} = 0$	$(\sigma^2_{\max})^i = 0.001$	$(\sigma^2_{\max})^t = 0.001$
FNO	2.37M	1	$0.0080 \pm$ 5e-4	$0.0079 \pm$ 2e-4	$0.0125 \pm$ 4e-5
FNO	4.15M	2	$0.0105 \pm$ 6e-4	$0.0106 \pm$ 4e-4	$0.0136 \pm$ 2e-5
FNO	7.71M	4	$0.2550 \pm$ 2e-8	$0.2557 \pm$ 8e-9	$0.2617 \pm$ 2e-9
FNO++	2.37M	1	$0.0075 \pm$ 2e-4	$0.0075 \pm$ 2e-4	$0.0145 \pm$ 7e-4
FNO++	4.15M	2	$0.0065 \pm$ 2e-4	$0.0065 \pm$ 9e-5	$0.0117 \pm$ 5e-5
FNO++	7.71M	4	$0.0064 \pm$ 2e-4	$0.0064 \pm$ 2e-4	**0.0109 ± 5e-4**
FNO-WT	2.37M	1	**0.0055 ± 1e-4**	**0.0056 ± 5e-5**	$0.0112 \pm$ 4e-4
FNO-DEQ	2.37M	1	**0.0055 ± 1e-4**	**0.0056 ± 7e-5**	$0.0112 \pm$ 4e-4

Table 12.5: Results on Darcy flow: clean data (Col 4),noisy inputs (Col 5) and noisy observations (Col 6) with max variance of added noise being $(\sigma^2_{\max})^i$ and $(\sigma^2_{\max})^t$, respectively. Reported test error has been averaged on three different runs with seeds 0, 1, and 2. Here, S-FNO++, S-FNO-WT and S-FNO-DEQ are shallow versions of FNO++, FNO-WT and FNO-DEQ respectively.

13 Appendix for Chapter 6

13.1 Training details

In this section, we will provide a detailed description of the training hyperparameters used in the KS experiments of Section 6.6.1, in the Burgers experimente of section **??** and the Navier Stokes experiments of section 6.6.2. We start with the training hyperparameters. All our experiments used a learning rate of 0.001. For the number of epochs, in KS and Burgers, the training was done over 200 epochs with cosine annealing learning scheduling [Loshchilov and Hutter, 2017]; whereas in Navier Stokes we trained for 300 epochs and halved the learning rate every 90. As for the number of samples, KS and Burgers were trained with 2048 samples and Navier Stokes with 1024 samples. Lastly, we observed that the batch size was a sensitive hyperparameter for both the memory and memoryless models (it seemed to affect both equally) so we ran a sweep at each experiment to select the best performing one. In the results shown in the paper, KS and Navier Stokes use a batch size of 32, and Burgers a batch size of 64.

Another relevant detail is the memory length in training, that is, the number of past states that were fed to the memory layer in the MemNO model. In the KS and Burgers experiments, the maximum memory lengths are 20 and 25 (which are the same as the number of timesteps of the dataset). That means that for the last timestep, the previous 19 or 24 states were fed into the memory layer. However, for GPU memory limitations in Navier Stokes the memory length was 16, half the number of timesteps of each trajectory in the dataset. In this case, the memory was reset after the 16th timestep, i.e. for the 16th timestep the 15 past states were fed to the memory model, yet for the 17th timestep only the 16th timestep was fed. Then, for the 18th timestep, the 17th and 16th were fed, and so on.

As in [Tran et al., 2023], experiments were trained using teacher forcing. This means that for the prediction of the i-th timestep during training, the ground truth of the $i-1$ previous steps was fed to the model (as opposed to the prediction of the model for such steps).

We ran our experiments on A6000/A6000-Ada GPUs. The Navier Stokes 2D experiments required around 34GB of GPU memory for the batch size of 32 and took around 5 hours to finish, whereas the rest of experiments in 1D required a lower GPU memory (less than 10GB) and each run took around 1 or 2 hours, depending on the resolution.

13.2 Ablations on the Memory layer

In this section we present two ablations regarding the memory layer of MemNO.

13.2.1 Ablation: Choice of sequential model

In section 6.5.3 we introduced MemNO as an architecture framework which allowed the introduction of memory through any choice of a sequential layer, which we chose as S4 in the previous experiments. In this section, we explore two other candidates for the sequential layers: a Transformer and an LSTM. We introduce **Transformer-FFNO (T-FFNO)** and **LSTM-FFNO** as two models that are identical to S4FFNO except in the sequential layer, where a Transformer and an LSTM are used respectively. The Transformer layer includes causal masking and a positional encoding, which is defined for pos across the time dimension and i across the hidden dimension by:

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{\text{dim\_model}}}}\right)$$
$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{\text{dim\_model}}}}\right)$$

We show results for the KS dataset with viscosity $\nu = 0.15$ and different resolutions. This dataset was generated using a resolution of 256 and contains 4096 samples, twice as many compared to the KS datasets of **??**, given that Transformers are known to perform better in high-data regimes. The results are shown in Figure 13.1. TFFNO performs significantly worse than S4FFNO across almost all resolutions, and even performs worse than FFNO. In contrast, LSTM-FFNO outperforms FFNO, which shows that MemNO can work with other sequential models apart from S4. The memory term in Equation 6.6 is a convolution in time, which is equivalent to the S4 layer and very similar to a Recurrent Neural Network (RNN) style layer, as showed in Gu et al. [2022b]. We believe that this inductive bias in the memory layer is the reason why both S4FFNO and LSTM-FFNO outperform FFNO. However, S4 was designed with a bias for continuous signals and has empirically proven better performance in these kind of tasks [Gu et al., 2022b], which is in agreement with its increased performance over LSTMs in this experiment. Additionally, we observed that LSTMs were unstable to train in Navier Stokes 2D datasets.

Lastly, we make two remarks. Firstly, we believe that Transformers performed worse due to overfitting, given that the train losses were normally comparable or even smaller than the train losses of the rest of the models at each resolution. We hypothesize that the full access to the past of Transformers models might lead to exploiting spurious correlations during training. Modifications of the Transformer layer or to the training hyperparameters as in other works [Hao et al., 2024a, Cao, 2021, Hao et al., 2023a] might solve this issue. Secondly, recently there has been a surge of new sequential models such as Mamba [Gu and Dao, 2023b, Dao and Gu, 2024b], RWQK [Peng et al., 2023], xLSTM [Beck et al., 2024] or LRU [Orvieto et al., 2023]. We chose S4 over Mamba-type architectures because in our experiments the PDE temporal dynamics do not change, and thus we do not expect the input-dependent selectivity mechanism to be necessary. However, we leave it as future work to study which of these sequential model has better overall performance, and hope that our study on the settings where the memory effect is relevant can help make accurate comparisons between them.

13.2.2 Ablation: memory layer configuration

In Section 6.5.3 we introduced the memory layer in MemNO as a single layer to be interleaved with neural operator layers. In our experiments, we inserted it after the second layer of a four layer neural operator.

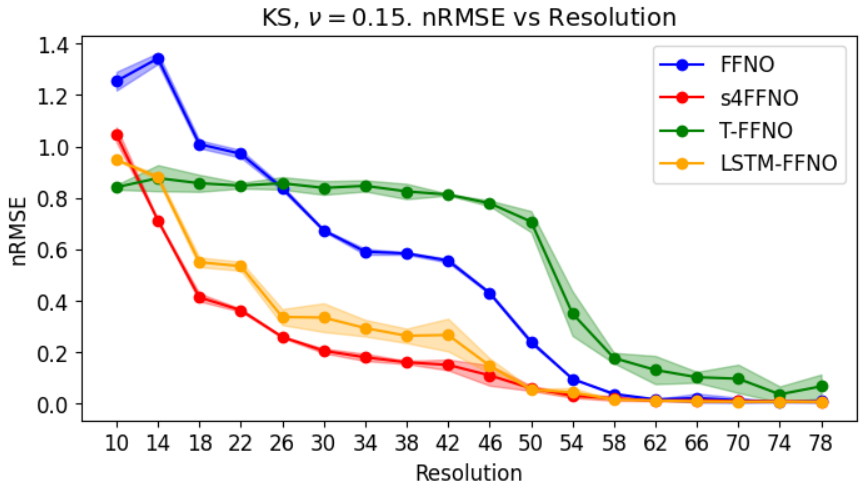


Figure 13.1: Performance of FFNO, S4FFNO and T-FFNO and LSTM-FFNO in KS with viscosity $\nu = 0.15$.

In this section, we explore the impact of having different layer configurations, including the possibility of having several memory layers. We will denote the configurations with a sequence of S and T letters. S means a neural operator layer (some sort of Spatial convolution), and T a memory layer (some sort of Time convolution). For example, *SSTSS* denotes the architecture of our experiments, where we have 2 neural operators layers, followed by a memory layer, followed by other 2 neural operator layers. Similarly, *SSSST* denotes 4 neural operators layers followed by a memory layer. In Table 13.1, we present the results for the KS dataset with $\nu = 0.1$ and final time of 4 seconds for several models. We include the S4FFNO model we used in previous experiments in the first row (with configuration *SSTSS*), and the FFNO model in the last row. In the middle rows, we show different configurations of memory and neural operator layers. It can be observed that all models with at least a memory layer outperform FFNO. There are slight differences between configurations, yet we focused mainly on the comparison to the memoryless model. For that reason, we fixed *SSTSS* configuration in our previous experiment, which was the most efficient (only one memory layer) and symmetric. We leave as further work determining if there are settings where a given configuration pattern can be substantially better than the rest.

13.3 Appendix: Quantifying the effect of memory

Proof. We proceed to the Equation 6.9 first. Note that $u_1(t), \forall t \geq 0$ can be written as $u_1(t) = a_0^{(t)}\mathbf{e}_0 + a_1^{(t)}\mathbf{e}_1$. Moreover, by Proposition 2, we have

$$\frac{\partial a_0^{(t)}}{\partial t} = 2Ba_1^{(t)} \tag{13.1}$$

$$\frac{\partial a_1^{(t)}}{\partial t} = a_1^{(t)} + Ba_0^{(t)} \tag{13.2}$$

Architecture	nRMSE ↓		
	Resolution 32	Resolution 48	Resolution 64
S4FFNO (*SSTSS*)	0.123 ± 0.011	0.086 ± 0.004	$\mathbf{0.015 \pm 0.001}$
S4FFNO (*SSSST*)	0.142 ± 0.009	0.069 ± 0.001	0.017 ± 0.001
S4FFNO (*STSSTS*)	0.141 ± 0.006	$\mathbf{0.064 \pm 0.002}$	0.019 ± 0.001
S4FFNO (*STSTSTST*)	$\mathbf{0.113 \pm 0.006}$	0.070 ± 0.004	0.017 ± 0.001
S4FFNO (*TSSSS*)	0.129 ± 0.007	0.080 ± 0.003	0.017 ± 0.001
FFNO	0.294 ± 0.004	0.138 ± 0.013	0.021 ± 0.002

Table 13.1: KS, $\nu = 0.1$. The final time is 4 seconds and the trajectories contain 20 timesteps. For each architecture, we tried 4 learning rates ($0.002, 0.001, 0.0005$ and 0.00025, each with three different seeds. We present the results of the learning rate with the lowest nRMSE averaged across the three seeds. The standard deviation is also with respect to the seeds.

In matrix form, these equations form a linear matrix ODE:

$$\frac{\partial}{\partial t}\begin{pmatrix} a_0^{(t)} \\ a_1^{(t)} \end{pmatrix} = \begin{pmatrix} 0 & 2B \\ B & 1 \end{pmatrix}\begin{pmatrix} a_0^{(t)} \\ a_1^{(t)} \end{pmatrix}$$

The solution of this ODE is given by $\begin{pmatrix} a_0^{(t)} \\ a_1^{(t)} \end{pmatrix} = \exp\left(t\begin{pmatrix} 0 & 2B \\ B & 1 \end{pmatrix}\right)\begin{pmatrix} a_0^{(0)} \\ a_1^{(0)} \end{pmatrix}$. By the first statement of Lemma 46 and the non-negativity of $a_0^{(0)}, a_1^{(0)}$, we get:

$$a_0^{(t)} \leq 10e^{\sqrt{2}Bt}\left(a_0^{(0)} + a_1^{(0)}\right), \tag{13.3}$$

$$a_1^{(t)} \leq 10e^{\sqrt{2}Bt}\left(a_0^{(0)} + a_1^{(0)}\right) \tag{13.4}$$

We proceed to Equation 6.10. Note that for any $s \geq 0$, we can write $u_2(s) = \hat{a}_0^{(s)}\mathbf{e}_0 + \hat{a}_1^{(s)}\mathbf{e}_1$ with $\hat{a}_0^{(0)} = a_0^{(0)}$ and $\hat{a}_1^{(0)} = a_1^{(0)}$. By Proposition 2, we have

$$\mathcal{Q}\mathcal{L}u_2(x) = B\hat{a}_1^{(s)}\mathbf{e}_2(x)$$

Moreover, given a function $v(x)$, the action of the operator $\exp \mathcal{Q}\mathcal{L}(\tilde{t})$ on v is given by the solution $w(\tilde{t}, x)$ to the PDE

$$\frac{\partial}{\partial t}w(t,x) = \mathcal{Q}\mathcal{L}w(t,x)$$
$$w(0,x) = v(x)$$

If $w(t,x) = \sum_{n\in\mathbb{N}_0} b_n^{(t)}\mathbf{e}_n$ and $\forall n \in \mathbb{N}_0, b_n^{(0)} \geq 0$, we are interested in solving the previous PDE with initial conditions $b_2^{(0)} = B\hat{a}_1^{(s)}$ and $b_n^{(0)} = 0\ \forall n \neq 2$.

We claim that the coefficients $\hat{a}_n^{(t)} \geq 0\ \forall t > 0$ and $\forall n \in \{0, 1\}$. For $t = 0$ this is by definition, and we will prove it for all t by way of contradiction. Suppose the claim is not true, then there exists a $t^* > 0$,

and some $n^* \in \{0,1\}$ such that $\hat{a}_{n^*}^{(t^*)} = 0$, and $\hat{a}_n^{(s)} > 0 \, \forall n \in \{0,1\}$ and $\forall s < t^*$. But from continuity this implies that there exists $0 < t' < t^*$ such that $\frac{\partial}{\partial t}\hat{a}_{n^*}^{(t')} < 0$. However, it can be easy to see that if $\hat{a}_n^{(s)} > 0 \, \forall s \leq t'$, then $\mathcal{P}_1 \mathcal{L} u_2(t') > 0$ and $\mathcal{P}_1 \mathcal{L} \int_0^{t'} \exp \mathcal{Q}\mathcal{L}(t-s) u_2(s) ds > 0$. Therefore, from Equation 6.10, $\frac{\partial}{\partial t}\hat{a}_{n^*}^{(t')} > 0$, which is a contradiction.

This claim implies that $b_n^{(0)} \geq 0 \, \forall n \in \mathbb{N}$, and in turn it implies that $b_n^{(t)} \geq 0 \, \forall n \in \mathbb{N}, t > 0$. Applying $\mathcal{Q}\mathcal{L}$ results in the following inequalities for the coefficients $b_1^{(t)}, b_2^{(t)}, b_3^{(t)}$:

$$\frac{\partial}{\partial t} b_1^{(t)} \geq b_1^{(t)} + B b_2^{(t)} \geq B b_2^{(t)} \tag{13.5}$$

$$\frac{\partial}{\partial t} b_2^{(t)} \geq B b_1^{(t)} + 4 b_2^{(t)} + B b_3^{(t)} \geq B b_1^{(t)} + B b_3^{(t)} \tag{13.6}$$

$$\frac{\partial}{\partial t} b_3^{(t)} \geq B b_2^{(t)} + 9 b_3^{(t)} \geq B b_2^{(t)} \tag{13.7}$$

Thus, we can write a linear matrix ODE for the vector $(b_1^{(t)}, b_2^{(t)}, b_3^{(t)})$:

$$\frac{\partial}{\partial t} \begin{pmatrix} b_1^{(t)} \\ b_2^{(t)} \\ b_3^{(t)} \end{pmatrix} \geq \begin{pmatrix} 0 & B & 0 \\ B & 0 & B \\ 0 & B & 0 \end{pmatrix} \begin{pmatrix} b_1^{(t)} \\ b_2^{(t)} \\ b_3^{(t)} \end{pmatrix} \tag{13.8}$$

Therefore, using Lemma 47, for sufficiently large B we have $b_2^{(t-s)} \geq \frac{B e^{\sqrt{2}B(t-s)}}{10} \hat{a}_1^{(s)}$.

Hence, if we write $\int_0^t \exp \mathcal{Q}\mathcal{L}(t-s) \mathcal{Q}\mathcal{L} u_2(s) ds$ in the basis $\{\mathbf{e}_n\}_{n \in \mathbb{N}_0}$, the coefficient for $\mathbf{e}_2$ will be lower bounded by

$$\int_0^t \frac{1}{10} B e^{B(t-s)} a_1^{(s)} ds$$

Applying the second statement of Lemma 46 and using the non-negativity of $a_0^{(0)}$ and $a_1^{(0)}$, we have $\hat{a}_1^{(s)} \geq \frac{1}{10} e^{\sqrt{2}Bs} \left(a_0^{(0)} + a_1^{(0)} \right)$. Hence, the coefficient for $\mathbf{e}_2$ is lower bounded by

$$\int_0^t \frac{1}{10} B e^{\sqrt{2}B(t-s)} \frac{1}{10} e^{\sqrt{2}Bs} \left(a_0^{(0)} + a_1^{(0)} \right) ds \geq \frac{Bt}{100} e^{\sqrt{2}Bt} \left(a_0^{(0)} + a_1^{(0)} \right)$$

We finally need to consider what happens after applying the outermost operator $\mathcal{P}_1 \mathcal{L}$. Because of Proposition 2 again, applying $\mathcal{L}$ makes the coefficient in front of $\mathbf{e}_1$ at least $\frac{B^2 t}{100} e^{\sqrt{2}Bt} \left(a_0^{(0)} + a_1^{(0)} \right)$. Finally, applying $\mathcal{P}_1$ preserves the coefficient in front of $\mathbf{e}_1$.

Hence, equation Equation 6.10 results in the following evolution inequalities:

$$\frac{\partial \hat{a}_0^{(t)}}{\partial t} \geq 2B\hat{a}_1^{(t)} \tag{13.9}$$

$$\frac{\partial \hat{a}_1^{(t)}}{\partial t} \geq \hat{a}_1^{(t)} + B\hat{a}_0^{(t)} + \frac{B^2 t}{100} e^{\sqrt{2}Bt}\left(a_0^{(0)} + a_1^{(0)}\right) \tag{13.10}$$

Using the second statement of Lemma 46 again we have that $\hat{a}_0(t) \geq \frac{1}{10} e^{\sqrt{2}Bs}\left(a_0^{(0)} + a_1^{(0)}\right)$. Thus, dropping the (positive) term $\hat{a}_1^{(t)}$ in equation 13.10, we have:

$$\frac{\partial \hat{a}_1^{(t)}}{\partial t} \geq \left(\frac{1}{10} + \frac{Bt}{100}\right) B e^{\sqrt{2}Bt}\left(a_0^{(0)} + a_1^{(0)}\right) \tag{13.11}$$

Integrating this equation yields:

$$\hat{a}_1^{(t)} \geq a_1^{(0)} + \frac{1}{200} e^{\sqrt{2}Bt}\left(\sqrt{2}Bt + 10\sqrt{2} - 1\right)\left(a_0^{(0)} + a_1^{(0)}\right) \tag{13.12}$$

Thus, we have $a_1^{(t)} \gtrsim Bte^{\sqrt{2}Bt}\left(a_0^{(0)} + a_1^{(0)}\right)$. Together with equation 13.3, the claim of the Theorem follows. □

Lemma 46. *There exists $B > 0$ sufficiently large such that for all $t > 0$ the matrix $\begin{pmatrix} 0 & 2Bt \\ Bt & t \end{pmatrix}$ satisfies:*

$$\forall i, j \in \{1, 2\}, \exp\left(\begin{pmatrix} 0 & 2Bt \\ Bt & t \end{pmatrix}\right)_{i,j} \leq 10 \exp(\sqrt{2}Bt) \tag{13.13}$$

$$\forall i, j \in \{1, 2\}, \exp\left(\begin{pmatrix} 0 & 2Bt \\ Bt & t \end{pmatrix}\right)_{i,j} \geq \frac{1}{10} \exp(\sqrt{2}Bt) \tag{13.14}$$

Proof. By direct calculation, we have:

$$\exp\left(\begin{pmatrix} 0 & 2Bt \\ Bt & t \end{pmatrix}\right) = \frac{1}{2\sqrt{8B^2+1}} \begin{pmatrix} \sqrt{8B^2+1}g(B,t) - h(B,t) & 4Bh(B,t) \\ 2Bh(B,t) & \sqrt{8B^2+1}g(B,t) + h(B,t) \end{pmatrix}$$

where:

$$g(B,t) = e^{\frac{1}{2}\left(\sqrt{8B^2+1}+1\right)t} + e^{-\frac{1}{2}\left(\sqrt{8B^2+1}-1\right)t}$$

$$h(B,t) = e^{\frac{1}{2}\left(\sqrt{8B^2+1}+1\right)t} - e^{-\frac{1}{2}\left(\sqrt{8B^2+1}-1\right)t}$$

Thus, the statement follows.

□

Lemma 47. *For all $B > 0$, the matrix* $\begin{pmatrix} 0 & B & 0 \\ B & 0 & B \\ 0 & B & 0 \end{pmatrix}$ *satisfies:*

$$\forall i, j \in \{1, 2, 3\}, \exp(\begin{pmatrix} 0 & B & 0 \\ B & 0 & B \\ 0 & B & 0 \end{pmatrix})_{i,j} \geq \frac{1}{10} \exp(\sqrt{2}B) \tag{13.15}$$

Proof. By direct calculation:

$$\exp(\begin{pmatrix} 0 & B & 0 \\ B & 0 & B \\ 0 & B & 0 \end{pmatrix})_{i,j} =$$

$$\frac{1}{4} e^{-\sqrt{2}B} \begin{pmatrix} 2e^{\sqrt{2}B} + e^{2\sqrt{2}B} + 1 & \sqrt{2}e^{2\sqrt{2}B} - \sqrt{2} & -2e^{\sqrt{2}B} + e^{2\sqrt{2}B} + 1 \\ \sqrt{2}e^{2\sqrt{2}B} - \sqrt{2} & 2(e^{2\sqrt{2}B} + 1) & \sqrt{2}e^{2\sqrt{2}B} - \sqrt{2} \\ -2e^{\sqrt{2}B} + e^{2\sqrt{2}B} + 1 & \sqrt{2}e^{2\sqrt{2}B} - \sqrt{2} & 2e^{\sqrt{2}B} + e^{2\sqrt{2}B} + 1 \end{pmatrix}$$

Thus, the statement follows.

□

14 Appendix for Chapter 7

14.0.1 Datasets

As mentioned in Section 9.8, we train our models using the datasets provided in the PDEBench [Takamoto et al., 2022]. The time-dependent PDE families considered by our models are: Burgers Equation (1D), Diffusion-Sportion (1D), Shallow-Water (2D), compressible Navier-Stokes (1D and 2D), incompressible Navier-Stokes (2D), and Diffusion-Reaction (1D and 2D). For each $s \in S$, the number of points in the n-point discretization W_n^s is 128, i.e, $n = 128$. For PDEs where the PDEbench-provided grid has more than 128 points in each dimension, we sample 128 equispaced points.

In this section, we provide few key properties and considerations for the PDEs used in this paper. The initial conditions $\bar{g}(0, x)$ for most of the datasets are sampled from a superposition of sinusoidal waves. The set of coefficients and number of trajectories used per PDE are reported in Appendix Table 14.1. For full details on the data generation process and the hyperparameters used to generate the PDE dataset, we refer the reader to Takamoto et al. [2022].

Burgers Equation (1D)

Burgers equation is commonly used to model the nonlinear dynamics of various fluid dynamics systems. Given the field $u(t, x) \in (0, 2] \times (0, 1) \to \mathbb{R}$ the PDE is defined as follows:

$$\partial_t u(t, x) + \partial_x \frac{u^2(t, x)}{2} = \frac{\nu}{\pi} \partial_{xx} u(t, x) \tag{14.1}$$

Here ν is the diffusion coefficient or the viscosity of the liquid, and π is the density of the liquid.

Diffusion-Sorption Equation (1D)

Diffusion-Sorption is a nonlinear diffusive process slowed down by an external force that is dependent of the state variable u R. This PDE is used to model groundwater contamination transport processes. The PDE is defined as the following:

$$\partial_t u(t, x) = \frac{D}{R(u)} \partial_{xx} u(t, x), \tag{14.2}$$

where $x \in (0, 1), t \in (0, 500]$, and $D = 5 \times 10^{-4}$. For more details on the initial conditions, boundary conditions and the function $R(u)$, we refer the reader to Takamoto et al. [2022]. For our training, we use 4500 trajectories for this PDE generated by varying the initial conditions.

Advection Equation (1D)

Given advection speed β, the advection equations are expressed as:

$$\begin{aligned} &\partial_t u(t,x) + \beta \partial_x u(t,x) = 0 \\ &u(0,x) = u_0(x) \end{aligned} \tag{14.3}$$

where $x \in (0,1)$ and $t \in (0,2]$. Various examples in this dataset are generated by sampling multiple initial conditions from a super-position of sinusoidal waves as used in Takamoto et al. [2022].

Compressible Navier-Stokes (1D and 2D)

Given density ρ, velocity $\bar{g}$, pressure p, internal energy of the system ϵ the compressible Navier-Stokes equations are defined as follows.

$$\begin{aligned} &\partial_t p + \nabla \cdot (\rho \bar{g}) = 0, \\ &\rho(\partial_t \bar{g} + \bar{g} \cdot \nabla \bar{g}) = -\nabla p + \eta \Delta \bar{g} + \left(\xi + \frac{\eta}{3}\right) \nabla(\nabla \cdot \bar{g}) \\ &\partial_t \left(\epsilon + \rho \frac{\|\bar{g}\|_2^2}{2}\right) + \nabla \cdot \left(\left(p + \epsilon + \rho \frac{\bar{g}^2}{2}\right)\bar{g} - \bar{g} \cdot \sigma'\right) = 0 \end{aligned} \tag{14.4}$$

Here, $x \in (-1,1)$ for 1D Navier-Stokes and $x \in (0,1)^2$ for 2D Navier-Stokes, and $t \in (0,1)$. Compressible Navier-Stokes stokes are used to model multiple real-world phenomena in aerodynamics and fluid dynamics.

Incompressible Fluid Navier-Stokes (2D)

We define the equations for incompressible fluid Navier-Stokes where we impose the condition that the fluid is "incommpressible." That is, the equation follows the following condition:

$$\nabla \cdot \boldsymbol{u} = 0 \tag{14.5}$$

For density ρ and pressure p, the equations used to generate the data in Takamoto et al. [2022] are as follows:

$$\rho(\partial_t \boldsymbol{u} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}) = -\nabla p \boldsymbol{u} + \eta \Delta \boldsymbol{u} + \boldsymbol{f} \tag{14.6}$$

where $\boldsymbol{f}$ is an external forcing function, and Dirichlet boundary conditions. Here $x \in [0,1]^2$ and the initial conditions $\boldsymbol{u}$ and the forcing term $\boldsymbol{f}$ are sampled from two-dimensional Gaussian random fields. Please refer to Takamoto et al. [2022] for more details on the data generation process.

Reaction Diffusion (1D and 2D)

Reaction Diffusion are diffusive processes with external force applied to the system that may or may not depend over the field variable $\bar{g}$. They are often used to model many thermodynamical systems.

1D reaction diffusion is defined as follows:

$$\partial_t u(t,x) - \nu \partial_{xx} u(t,x) = \rho u(t,x)(1 - u(t,x)) \tag{14.7}$$

for all $x \in (0, 1)$ and $t \in (0, 1]$.

For 2D reaction diffusion, let $\bar{g}(t, x) = [u_1(t, x), u_2(t, x)]$. Then the equations are defined as:

$$\begin{aligned} \partial_t u_1(t,x) &= \nu_1 \partial_{x_1 x_1} u_1 + \nu_1 \partial_{x_2 x_2} u_1 + u_1 - u_1^3 - k - u_2 \\ \partial_t u_1(t,x) &= \nu_2 \partial_{x_1 x_1} u_2 + \nu_2 \partial_{x_2 x_2} u_2 + u_1 - u_2 \end{aligned} \tag{14.8}$$

where $k = 5 \times 10^{-3}$ and ν_1 and ν_2 are diffusion coefficients. Here $x_1 \in (-1, 1)$ and $x_2 \in (-1, 1)$ and the initial conditions are sampled from a Gaussian random field.

Shallow-Water Equations (2D)

These are derived from Navier-Stokes and are a framework for modelling free-surface flow problems. We denote by $u_1(x)$, and $u_2(x)$ as the velocities in the horizontal and vertical directions and h as the height of the water and b defining the spatially varying bathymetry (the measurement of the depth of water in oceans, rivers, or lakes). The shallow-water equations are defined as follows:

$$\begin{aligned} &\partial_t h + \partial_{x_1} h u_1 + \partial_{x_2} h u_2 = 0, \\ &\partial_t h u_1 + \partial_{x_1} \left(u_1^2 h + \frac{1}{2} g_r h^2 \right) + \partial_{x_2} u_1 u_2 h = -g_r h \partial_{x_1} b, \\ &\partial_t h u_2 + \partial_{x_2} \left(u_2^2 h + \frac{1}{2} g_r h^2 \right) + \partial_{x_1} u_1 u_2 h = -g_r h \partial_{x_2} b, \end{aligned} \tag{14.9}$$

where $x \in [-2.5, 2.5]^2$ and g_r is the gravitational acceleration.

Summary

The following table summarizes the coefficients of the datasets used to train and test our model (note that 1D/2D Diffusion-Reaction only appear in the test set but not the training set). We also provide the number of training and test trajectories. We generate the input-output pairs using autoregressive teacher-forcing.

Table 14.1: For each PDE family, we select one set of coefficients and use the data for training and testing UPS.

Dimension	Dataset	Coefficients	Num Train Trajectories	Num Test Trajectories	Timesteps	Resolution
1D	Advection	$\beta = 0.4$	4500	1000	41	128
	Burgers	$\nu = 0.001$	4500	1000	41	128
	Diffusion-Reaction	$\nu = 0.5, \rho = 1.0$	4500	1000	21	128
	Diffusion-Sorption	-	4050	100	21	128
	Compressible Navier-Stokes	$\eta = \zeta = 0.1$, rand_periodic	4500	1000	21	128
2D	Shallow-Water	-	405	10	101	128
	Diffusion-Reaction	-	405	10	101	128
	Compressible Navier-Stokes	$M = \eta = \zeta = 0.1$, periodic	4500	1000	21	128
	Incompressible Navier-Stokes	$M = 0.1, \eta = \zeta = 1E-8$	4500	1000	21	128

14.0.2 Experiment Details

Training Hyperparameters

We use the following training hyperparameters for all of our experiments, unless otherwise specified. Due to time constraint, we have not performed exhausitive hyperparameter search or tailor the hyperparameters to each experiment setting.

- Batch size: 32
- Gradient accumulation: 1
- Gradient clipping: -1
- Dropout: 0
- Optimizer: Adam
- Learning rate: 5E-5
- Weight decay: 1E-5
- Training epoch: 20 for stage 1, 100 for stage 2

We use the CoNLL-2003 dataset [Sang and Meulder, 2003] as the reference dataset for alignment in stage 1.

Efficiency Analysis

We run all of our experiments on a single NVIDIA A6000. Table 14.2 show the detailed model size, per epoch training time (in seconds), and total training time (in hours) for different network configurations. Note that we train the models for 100 epochs.

Table 14.2: Trainable parameters and training time for each LLM backbone.

	RoBERTa-Base	RoBERTa-Large	Flan-T5-Base	CLIP-Base
Num Params	149M	387M	176M	132M
Per Epoch (s)	3200	7600	3500	3000
Total (hrs)	88	211	97	83

We reported additional metrics such as FLOPs and the time required for predicting a single step for a PDE instance in Table 14.3, assuming the input data is 2D with 4 channels and resolution 128. We mainly compared with unified models that have similar model sizes. Compared to these existing work, UPS has lower FLOPs and shorter inference time. This shows that our model is ideal for deployment in practical environments where both computational efficiency and speed are critical.

Table 14.3: Efficiency comparison for unified neural operators.

	UPS-B	MPP-B	DPOT-M
Num Params	149M	116M	122M
Per Forward Pass FLOPs (G)	72.66	102.12	75.44
Single Step Inference Time (ms)	1.77	2.34	1.88

14.0.3 Detailed Experiment Results

2D-Only UPS

Table 14.4: Training UPS with all of the 2D datasets in PDEBench and compare with MPP and DPOT. Note that beyond these PDEBench datasets, MPP is also pretrained on PDEArena [Gupta and Brandstetter, 2022] and DPOT is pretrained on PDEArena [Gupta and Brandstetter, 2022] as well as CFD-Bench [Yining et al., 2023]. Baseline results taken from Hao et al. [2024b]. '-' means that the result is not available.

		# Params	PDEBench 2D Navier Stokes-(η, ζ)						2D Diff-React	2D Shallow-Water
	(sorted within groups)		1,0.1	1,0.01	M1	0.1,0.1	0.1,0.01	M0.1		
Small-Sized	FNO	0.5M	0.098	0.096	0.097	0.360	0.170	0.265	0.12	0.0044
	FFNO	1.3M	0.0212	0.052	0.0366	0.162	0.0452	0.104	0.0571	0.0116
	GNOT	1.8M	0.0325	0.0420	0.0373	0.0228	0.0341	0.0285	0.0311	0.00678
	Oformer	1.9M	0.0417	0.0625	0.0521	0.0254	0.0205	0.0229	0.0192	0.00717
Medium-Sized	MPP-Ti	7M	–	–	0.0442	–	–	0.0312	0.0168	0.0066
	DPOT-Ti	7M	0.0173	0.0397	0.0285	0.0132	0.0220	0.0176	0.0321	0.00560
	MPP-S	30M	–	–	0.0319	–	–	0.0213	**0.0112**	0.0024
	DPOT-S	30M	0.0153	0.0337	0.0245	0.0119	0.0187	0.0153	0.0379	0.00657
	DPOT-M	122M	0.0116	**0.0238**	**0.0177**	0.00866	0.0129	**0.0108**	0.0292	0.0029
	UPS-B (Ours)	149M	**0.0112**	0.0605	0.0277	**0.0085**	**0.0124**	0.0211	0.0243	**0.0018**
Large-Sized	**UPS-L (Ours)**	387M	0.0102	0.0596	0.024	**0.0083**	**0.0102**	0.0209	0.0236	**0.0015**
	MPP-L	400M	–	–	0.0208	–	–	0.0147	**0.0098**	0.00220
	DPOT-L	500M	0.0100	0.0216	0.0158	0.00872	0.0115	0.0101	0.0232	0.00233
	DPOT-H	1.03B	**0.00961**	**0.0180**	**0.0138**	0.00847	0.0105	**0.00948**	0.0191	0.00199

Few-Shot Adaptation

Compared to full fine-tuning of stage 2, we lower the learning rate when performing few-shot adaptation to prevent catastrophic forgetting.

- Batch size: 32
- Gradient accumulation: 1
- Gradient clipping: -1
- Dropout: 0
- Optimizer: Adam

- Learning rate: 1E-5
- Weight decay: 1E-5
- Epoch: 100

The following table reports the time required for few-shot experiments. Note that for Burgers equation, we train the model using $\nu = 0.001$, but the results here are for $\nu = 1.0$.

Table 14.5: Time for few-shot experiments. Our model outperforms most existing baselines on these tasks by using fewer than 500 samples and much shorter adaptation time.

Num Samples	1D Diffusion-Reaction		2D Diffusion-Reaction		Burgers $\nu = 1.0$	
	Per Epoch (s)	Total (hrs)	Per Epoch (s)	Total (hrs)	Per Epoch (s)	Total (hrs)
10	2	0.05	12	0.33	3	0.08
50	10	0.28	48	1.33	10	0.28
100	23	0.64	112	3.11	40	1.11
500	112	3.11	512	14.22	96	2.67

Ablation on Longer Sequence Length

We studied the effect of embedding sequence length in Section 7.5.3 paragraph S5 of the main paper. The results show that among $l = \{8, 20, 32\}$, larger l indeed leads to better performance. However, since LLMs can support sequence lengths much longer than $l = 32$, we consider expanding the feature length (the number of "tokens") used to represent PDE data. See results below.

	Advection 1D	Burgers 1D	Diffusion-Sorption 1D	Navier-Stokes 1D	Shallow-Water 2D	Navier-Stokes 2D	Incomp Navier-Stokes 2D
$l = 32$	**0.0027**	0.0399	**0.0009**	0.0056	0.0016	**0.0153**	**0.0931**
$l = 64$	0.0034	**0.038**	**0.0009**	**0.0054**	**0.0015**	0.0162	0.0988

While $l = 64$ performs slightly better on some tasks, increasing the sequence length means that (i) the embedding network is going to be larger (since l also corresponds to the width of the FNO layers), and (ii) the training time will increase as each sequence is longer. Both increase the training cost. Hence, we want to select the l that achieves a balance between efficiency and effectiveness. That's why we use $l = 32$ for our main experiments.

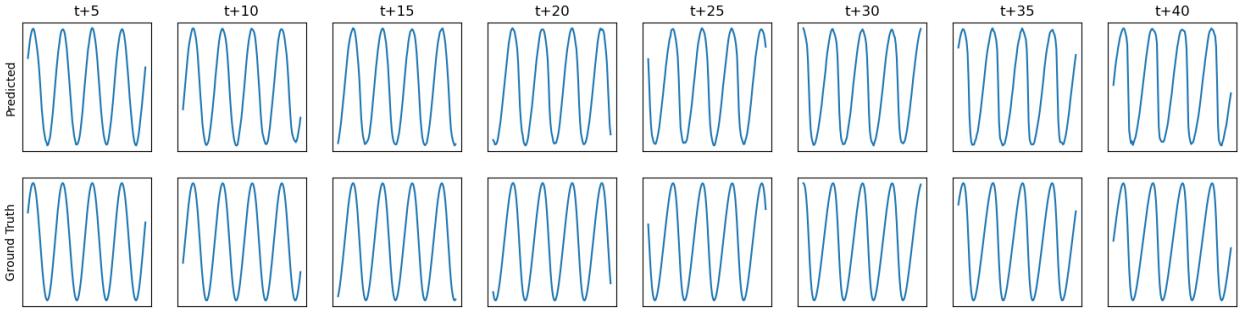
Long-Horizon Prediction

As stated in Section 7.3, our method mainly focuses on predicting the next step from the current step, i.e., $\hat{\bar{g}}^s_{t+1}(\mathbf{x}) = \mathcal{G}_\theta(\bar{g}^s_t(\mathbf{x}))$. However, we are also interested in the prediction capacity of our method over a longer period of time. Thus, we study an additional setting that predicts $\hat{\bar{g}}^s_{t+10}(\mathbf{x})$. We show non-autogressive evaluation results since otherwise we will only have very few time steps for each test PDE trajectory. The table below shows that UPS is still effective for long-horizon prediction compared to baselines like FNO. Even though the prediction interval is longer, the error rates only slightly increase possible because we use non-autoregressive evaluation, so the errors do not accumulate.

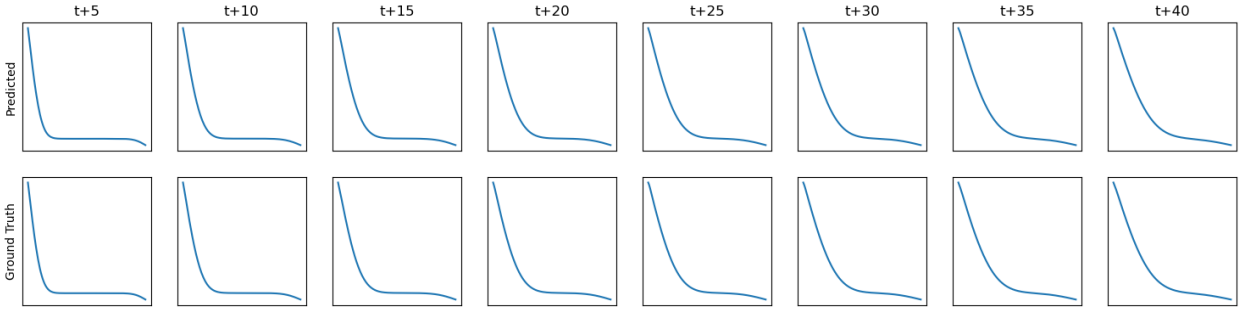
	Advection 1D	Burgers 1D	Diffusion-Sorption 1D	Navier-Stokes 1D	Shallow-Water 2D	Navier-Stokes 2D	Incomp Navier-Stokes 2D
$\Delta t = 1$	0.0027	0.0399	0.0009	0.0056	0.0016	0.0153	0.0931
$\Delta t = 10$	0.0034	0.04	0.0011	0.0074	0.0026	0.0189	0.134

14.0.4 Visualization

Burgers Equation

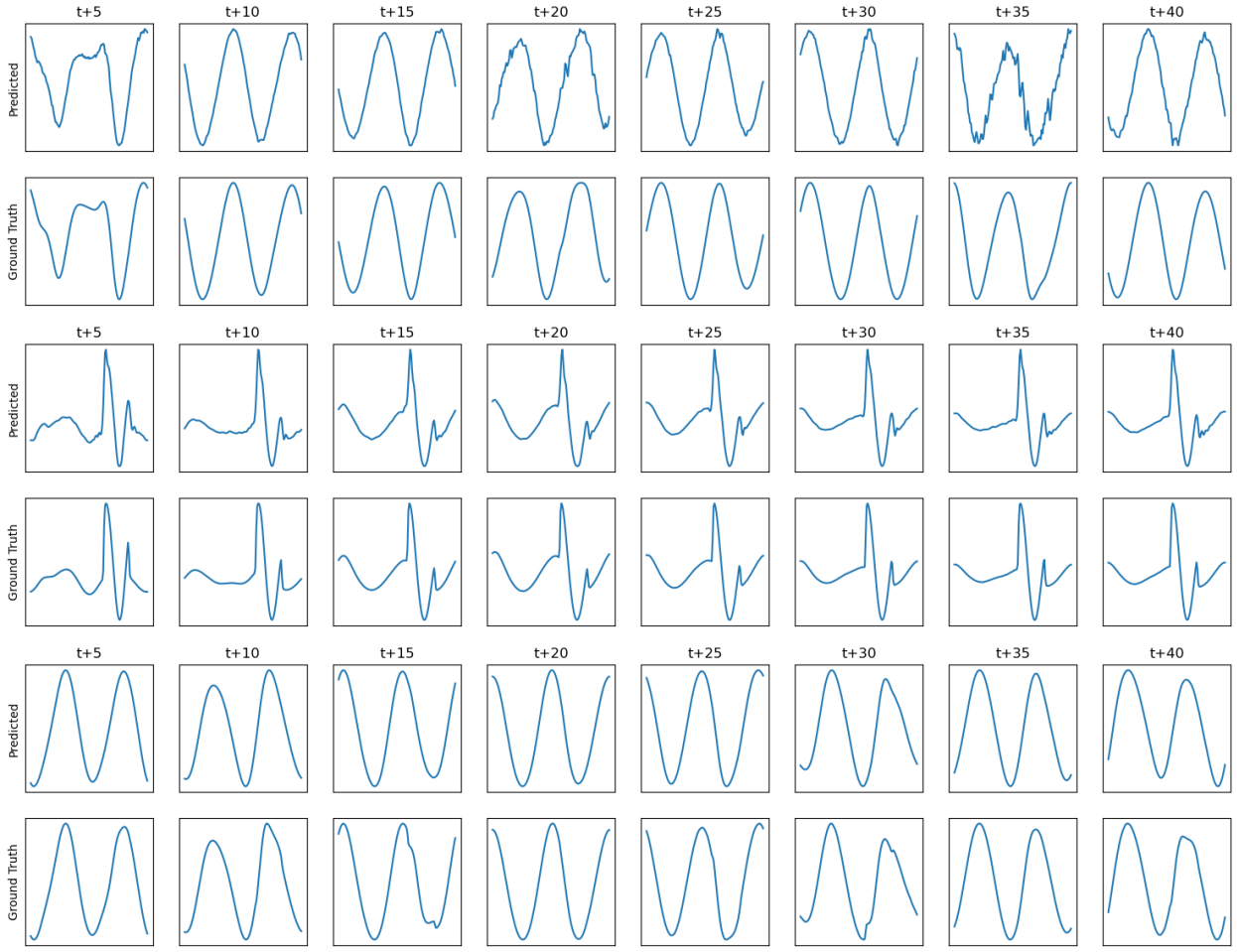


Diffusion-Sorption

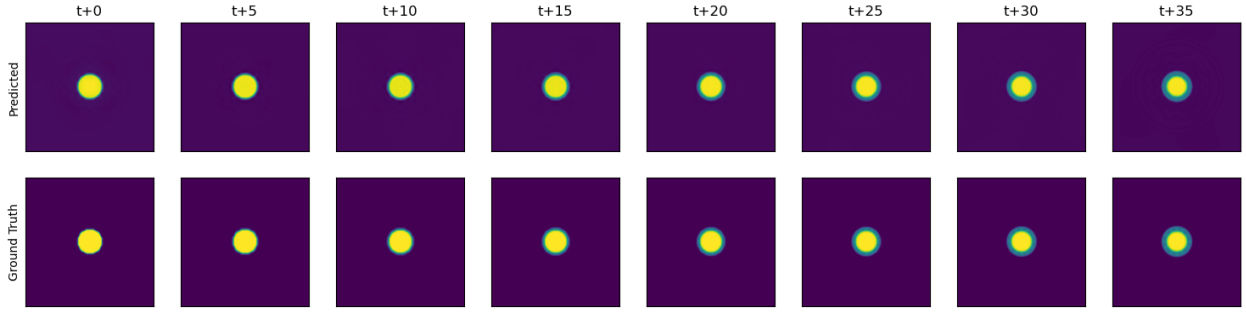


1D Navier Stokes

We show V_x, density, and pressure.

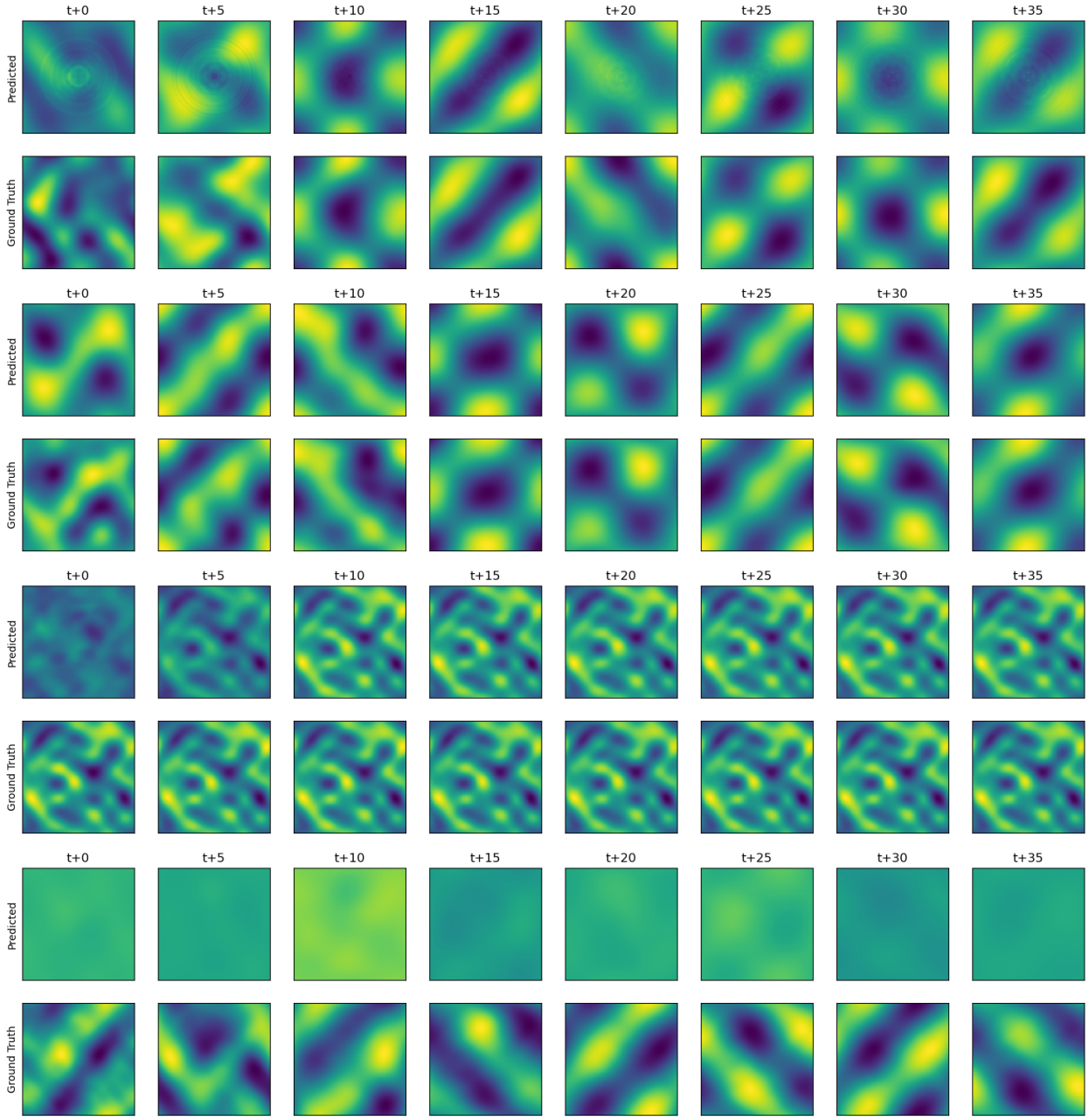


Shallow Water



2D Navier Stokes

We show V_x, V_y, density, and pressure.



In the prediction for 2D compressible Navier-Stokes we see a few artifacts in our generation. Furthermore, for quantities like pressure, our network often seems to generate an overly smoothened output. This could be because the 2D Navier-Stokes is the only PDE in our dataset that requires us to model pressure, and therefore the network is biased towards predicting a uniform value, which in our case is 0. We believe this can be avoided by adding more families of PDEs that model pressure, and is a fertile ground for future work.

15 Appendix for Chapter 8

15.1 Deferred Proofs

15.1.1 Proof of Proposition 6

Proof. Let $\boldsymbol{\epsilon}_i \sim \mathcal{T}(\mathbf{0}, \mathbf{I}_T)$ be T i.i.d. random Gaussian vectors. Assuming Gaussian initialization for the adjacency matrix $\mathbf{A}$, it can be expressed as:

$$\mathbf{A}[i,:] = \frac{\gamma \boldsymbol{\epsilon}_i}{\|\boldsymbol{\epsilon}_i\| + \exp(-\Psi_i)}. \tag{15.1}$$

We first show that $\|\boldsymbol{A}\| \leq \gamma < 1$. From the concentration of the norm of a Gaussian random vector, with high probability $\|\boldsymbol{\epsilon}_i\| \geq \sqrt{T}$ for all tokens i. Since $\exp(-\Psi_i) \geq 0$, $\|\boldsymbol{\epsilon}_i\| + \exp(-\Psi_i) \geq \sqrt{T}$. Consider any unit vector $\boldsymbol{u}$, then

$$\|\boldsymbol{A}\boldsymbol{u}\| = \sum_{i=1}^{T} \frac{\gamma \boldsymbol{\epsilon}_i^T \boldsymbol{u}}{\|\boldsymbol{\epsilon}_i\| + \exp(-\Psi_i)} \leq \gamma \sum_{i=1}^{T} \frac{\epsilon_i}{\sqrt{T}} \leq \gamma \frac{\sqrt{T}\epsilon}{\sqrt{T}} = \gamma\epsilon < 1, \tag{15.2}$$

with probability greater than $1 - \Phi(\frac{-1}{\gamma})$, were $\epsilon_i, \epsilon \sim \mathcal{N}(0,1)$. Finally, since the operator norm of $\|\boldsymbol{A}\|$ is less than one, we apply Banach's Lemma to get,

$$\|(\boldsymbol{I} - \boldsymbol{A})^{-1}\| \leq \frac{1}{1 - \|\boldsymbol{A}\|}, \tag{15.3}$$

which implies that the inverse exists. □

15.1.2 Proof of Proposition 9

Proof.

$$\text{Var}(\boldsymbol{C}_i^T \boldsymbol{h}_i) = \frac{1}{|p(i)|} \left(\sum_{j \in p(i)} \boldsymbol{A}_{ij} \text{Var}(\boldsymbol{C}_i^T \boldsymbol{h}_j) + \ln(\boldsymbol{A}_{ij}) \text{Var}(\boldsymbol{C}_i^T \boldsymbol{B}_i v_i) \right), \tag{15.4}$$

$$= \frac{1}{|p(i)|} \left(\sum_{j \in p(i)} \boldsymbol{A}_{ij} \text{Var}(\boldsymbol{C}_j^T \boldsymbol{h}_j) + \frac{2}{d} \ln(\boldsymbol{A}_{ij}) \right), \tag{15.5}$$

where we have used the fact that $\mathrm{Var}(\boldsymbol{C}_j^T \boldsymbol{h}_j) = \mathrm{Var}(\boldsymbol{C}_i^T \boldsymbol{h}_j)$, and that the variance of $\mathcal{X}^2$ distribution with d degrees of freedom is $2d$. Let $d \geq 4$, then

$$\mathrm{Var}(\boldsymbol{C}_i^T \boldsymbol{h}_i) \leq \frac{1}{|p(i)|} \left(\sum_{j \in p(i)} \boldsymbol{A}_{ij} + \frac{2}{d} \ln(\boldsymbol{A}_{ij}) \right) \leq \frac{1}{|p(i)|} \sum_{j \in p(i)} 1 \leq 1, \tag{15.6}$$

where we have used the fact that $\boldsymbol{A}_{ij} \in [0, 1]$. □

15.1.3 Proof of Proposition 10

Proof. In the structured masked attention (SMA) framework Dao and Gu [2024c], the computational complexity is the cost of the matrix-vector multiplication by the mask matrix $\boldsymbol{L} = (\boldsymbol{I} - \boldsymbol{A})^{-1}$. In the case of DAGs, $\boldsymbol{A}$ is (up to conjugation by a permutation) a *lower-triangular* matrix with $|\mathcal{E}|$ (number of edges) non-zero entries. It suffices to analyze the cost of computing the multiplication $\boldsymbol{y} = (\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{x}$. Rewriting as $(\boldsymbol{I} - \boldsymbol{A})\boldsymbol{y} = \boldsymbol{x}$, $\boldsymbol{y}$ can be computed through Gaussian elimination on the matrix $\boldsymbol{I} - \boldsymbol{A}$, which takes time proportional to the number of non-zero entries or $|\mathcal{V}| + |\mathcal{E}|$.

In graph terminology, this operation can be viewed as a dynamic programming algorithm to propagate features through the SSM update, where the ordering of edges to perform the update rule is given by the Gaussian elimination ordering. □

15.2 Additonal Experiments

15.2.1 MLM: Chimera on Undirected Line Graphs

For an undirected line graph (Figure 8.4, left), the adjacency matrix $\boldsymbol{A}$ takes the following form:

$$\boldsymbol{A} = \begin{bmatrix} 0 & a_{12} & 0 & \cdots & 0 \\ a_{21} & 0 & a_{23} & \cdots & 0 \\ 0 & a_{32} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 \cdots 0 & 0 \cdots 0 & 0 & a_{T-1,T} & 0 \end{bmatrix}.$$

As discussed in Section 8.3.3, to ensure the existence of $(\boldsymbol{I} - \boldsymbol{A})^{-1}$, we introduced a row-wise sum normalization strategy, wherein we normalized each row of the adjacency matrix with $\sum_j \boldsymbol{A}_{ij} + \Psi_i$. However, since this constraint is designed for general graphs, it is not sufficiently expressive. Therefore, we instead use a strictyly more expressive constraint for line graphs which enforces $\boldsymbol{A}_{ij} \cdot \boldsymbol{A}_{ji} + \Psi_i \leq \frac{1}{4}$ on each simple cycle of the graph.

Proposition 15. *Under the above constraint, the inverse $(\boldsymbol{I} - \boldsymbol{A})^{-1}$ exists as for any two nodes, the sum of all paths between them is upper bounded by $\sum_i (1/4)^i \leq 1/3$.*

15.2.2 Imagenet: Parameter Sharing Ablation

We study the trade-off between sharing parameters for $\boldsymbol{B}, \boldsymbol{C}$ across different graphs as a domain-dependent design choice. We explore four settings: *No sharing, Complete sharing, Row-wise sharing, and Diagonal sharing* across the four DAGs. From Table 15.1, we observe that diagonal sharing achieves the best performance, indicating it strikes the optimal tradeoff between parameter sharing and other modes of increasing expressivity for modeling image data.

Method (22M)	Top-1 (%)		Top-5 (%)	
	Acc	Acc_{EMA}	Acc	Acc_{EMA}
None	77.10	76.13	93.55	93.15
Complete	77.25	76.09	93.75	93.21
Row-wise	77.46	76.57	93.76	93.37
Diagonal	**77.80**	**76.69**	**93.87**	**93.53**

Table 15.1: Ablation: Diagonal parameter sharing works best.

15.3 Architectural Details

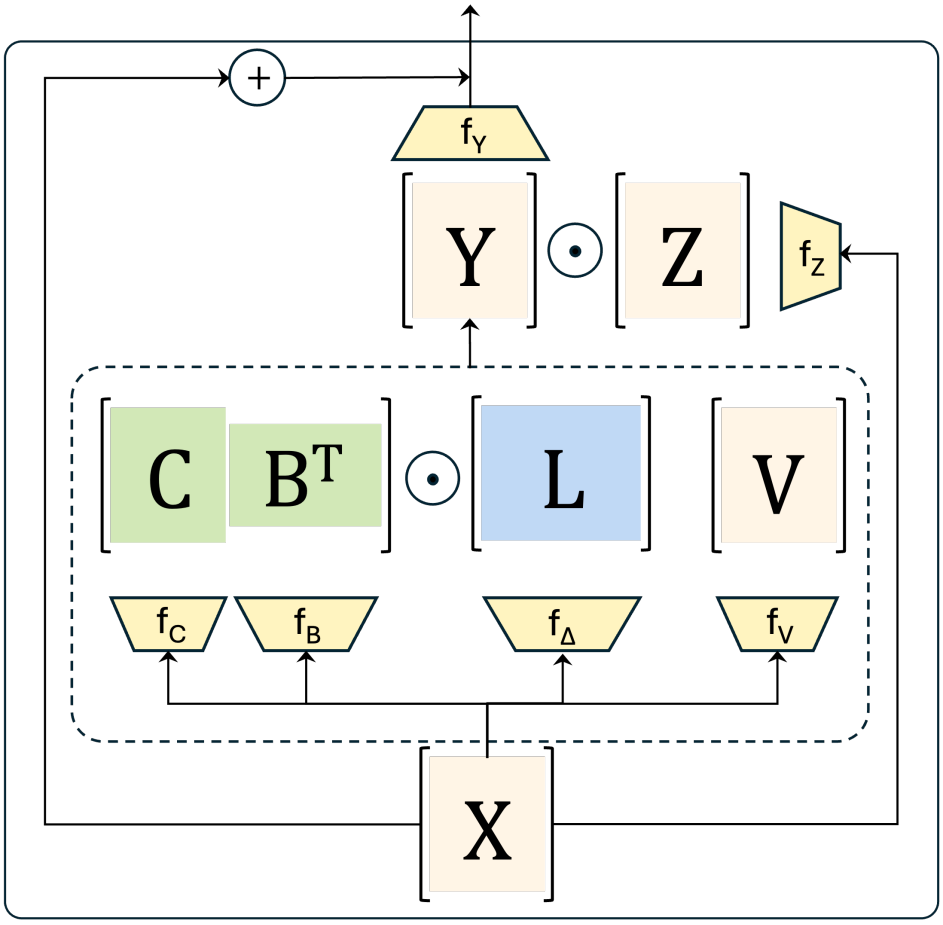


Figure 15.1: Chimera's Architecture: The output of the Chimera layer is embedded within the gated block introduced in Mamba-2 [Dao and Gu, 2024a]. Here $\boldsymbol{X}$ matrix denotes the input to the block, and f_c, f_B, f_Δ and f_V are data dependent projections defined in Section 8.2. The operator $\odot$ denotes element-wise multiplications between matrices, and $\oplus$ defines addition. The output from the Chimera layer is passed through a Gated-MLP, a final projection f_Y, followed by a residual connection.

15.3.1 Masked Language Modeling

In Table 15.2, we provide the architectural and training details for BERT-B and Chimera on the MLM task. For both the models, we follow the M2 recipe from Fu et al. [2023], adjusting the number of layers to 12 for BERT-B and 23 for Chimera to control for the number of parameters. We conducted a small sweep to fine-tune the learning rate for Chimera, choosing $8e-4$ over BERT-B's $5e-4$.

15.3.2 ImageNet-1k Classification

For the image classification experiments, we largely follow the ViT-B recipe with the following adjustments as shown in Table 15.4: To control for the number of parameters, we adjust the number of layers

Table 15.2: Architectural and Training Details for BERT-B and Chimera on MLM

Parameter	**BERT-B** (110M)	**Chimera** (110M)
Model dimension (d_{model})	768	768
Layers	12	23
Max sequence length	128	128
Num Heads	12	12
Head size	64	64
Optimizer	Decoupled AdamW	Decoupled AdamW
Learning rate	$5e-4$	$8e-4$
Optimizer momentum	$\beta_1 = 0.9, \beta_2 = 0.98$	$\beta_1 = 0.9, \beta_2 = 0.98$
Weight decay	$1e-5$	$1e-5$
Batch size	4096	4096
Learning rate schedule	Linear decay with warmup	Linear decay with warmup
Training steps	70k	70k
MLM Probability	0.3	0.3

from 12 for ViT-B to 22 for Chimera. Additionally, we reduce the Cutmix augmentation from 1.0 to 0.1, as Chimera's stronger inductive bias mitigates the risk of overfitting.

In Table 15.4, we present the reduced setting used for our ablation studies in Tables 15.1 and 8.3, where we match the number of parameters of ViT-S (22M).

15.3.3 Long Range Graph Benchmark

To train Chimera on the Long Range Graph Benchmark we follow a similar training recipe to that provided in Rampášek et al. [2022] where we replace the Transformer layers with Chimera layers. Moreover, in line with the baselines, we make sure that our models have less than $500k$ parameters. While training Chimera on graphs we remove the Gated-MLP layer Z defined in Figure 15.1. We did this to keep our training recipe as close to that provided in Rampášek et al. [2022] and highlight the effectiveness of Chimera. The hyperparameters used to train Chimera are provided in Table 15.5.

Table 15.3: Hyperparameters used for ViT-B and Chimera for ImageNet-1k classification task

Parameter	**ViT-B** (88M)	**Chimera (88M)**
Image size	224^2	224^2
Optimizer	AdamW	AdamW
Optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.999$	$\beta_1, \beta_2 = 0.9, 0.999$
Weight init	trunc. normal (std=0.02)	trunc. normal (std=0.02)
Learning rate	$1e-3$	$1e-3$
Weight decay	0.05	0.05
Batch size	1024	1024
Training epochs	310	310
Learning rate schedule	cosine decay	cosine decay
Warmup epochs	10	10
Warmup schedule	linear	linear
Patch Size	16	16
Layers	12	22
Num Heads	12	12
Droppath	0.3	0.3
Randaugment	(9,0.5,layers=2)	(9,0.5,layers=2)
Mixup	0.8	0.8
Cutmix	1.0	0.1
Random erasing	0.25	0.25
Label smoothing	0.1	0.25
Stochastic depth	0.1	0.25
Exp. mov. avg (EMA)	0.99996	0.99996

Table 15.5: Hyperparameters running Chimera on the Long Range Graph Benchmark

	Peptides-Func	**Peptides-Struct**	**PascalVOC-SP**	**COCO-SP**
Learning Rate	0.001	0.001	0.001	0.001
Optimizer	Adam	Adam	Adam	Adam
dropout	0.1	0.1	0.1	0.1
#layers	2	2	4	4
hidden dim.	256	256	128	128
head depth	2	2	2	2
batch size	32	32	32	32
#epochs	250	250	200	200
norm	BatchNorm	BatchNorm	BatchNorm	BatchNorm
MPNN	GCN	GCN	GCN	GCN
#Param.	461k	447k	498k	498k

Table 15.4: Key differences between the original and the ablation setting for Chimera

Parameter	**Chimera-S (2D)**
Model dimension (d_{model})	384
Number of layers	22
Number of Heads	3
Droppath	0.1

16 Appendix for Chapter 9

16.1 Omitted Proofs from Section 9.5

In this section we give omitted proofs and lemmas from Section 9.5.

Lemma 48. *Fix $n \in \mathbb{N}$. Let G, Φ be as defined in Theorem 11. Then there is an $O(n)$-time algorithm that computes a MAP evaluator for G with potential function class Φ.*

Proof. Fix any $J \in \Phi^E$. As preliminary notation, for each $c, c_0 \in \{0,1\}$ and $i, j \in \sqrt{n}$, let $V(i,j) := \{0\} \cup \{(k,j) : 1 \le k \le i\}$, and let $E(i,j)$ be the edge set of the induced subgraph $G[V(i,j)]$. Let

$$\hat{x}_{i,j}(c, c_0; J) := \operatorname*{argmin}_{\substack{x \in \{0,1\}^{V(i,j)}: \\ x_0 = c_0 \wedge x_{(i,j)} = c}} \sum_{(a,b) \in E(i,j)} J_{\{a,b\}}(x_a, x_b),$$

$$\hat{C}_{i,j}(c, c_0; J) := \min_{\substack{x \in \{0,1\}^{V(i,j)}: \\ x_0 = c_0 \wedge x_{(i,j)} = c}} \sum_{(a,b) \in E(i,j)} J_{\{a,b\}}(x_a, x_b).$$

For each $j \in [\sqrt{n}]$, let

$$\hat{x}_j(c_0; J) := \hat{x}_{\sqrt{n},j}\left(\left(\operatorname*{argmin}_{c \in \{0,1\}} \hat{C}_{\sqrt{n},j}(c, c_0; J)\right), c_0; J\right).$$

Finally, let $\hat{x}(c_0; J) \in \{0,1\}^V$ be the vector which takes value c_0 on vertex 0, and value $\hat{x}_j(c_0; J)_i$ on vertex (i,j) for all $i, j \in \sqrt{n}$. Let

$$\hat{x}(J) := \arg\max_{c_0 \in \{0,1\}} p_J(\hat{x}(c_0; J)).$$

We claim that $\hat{x}(J)$ is a maximizer of $p_J(x)$. Indeed, for any fixed $c_0 \in \{0,1\}$, $\hat{x}(c_0; J)$ is a maximizer of $p_J(x)$ subject to $x_0 = c_0$, because under this constraint the maximization problem decomposes into $\sqrt{n}$ independent maximization problems, one for each path in G, which by definition are solved by $\hat{x}_1(c_0; J), \ldots, \hat{x}_{\sqrt{n}}(c_0; J)$.

Moreover, it's straightforward to see that for any fixed j, $\hat{C}_j(c_0; J)$ can be computed in $O(\sqrt{n})$ time by dynamic programming. Indeed for any i, j, $\hat{C}_{i,j}(c, c_0; J)$ can be computed in $O(1)$ time from $\hat{C}_{i-1,j}(0, c_0; J)$ and $\hat{C}_{i-1,j}(1, c_0; J)$ as well as $J_{\{0,(i,j)\}}$ and $J_{\{(i-1,j),(i,j)\}}$. Once the values $\hat{C}_{i,j}(c, c_0; J)$ have been computed for all $i \in [\sqrt{n}]$ and $c \in \{0,1\}$, the vector $\hat{x}_j(c_0; J)$ can be computed in $O(\sqrt{n})$ time via a reverse scan over $i = \sqrt{n}, \ldots, 1$. It follows that $\hat{x}(J)$ can be computed in $O(n)$ time. □

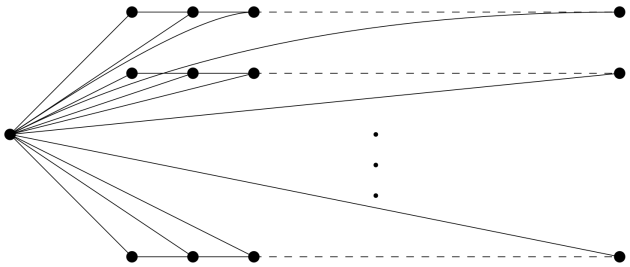


Figure 16.1: The graph G for which Theorem 11 exhibits a separation between edge message-passing and node message-passing. The graph consists of $\sqrt{n}$ paths of length $\sqrt{n}$, as well as a single "hub vertex" connected to all other vertices.

Proof of Proposition 12. We claim that there is a node message-passing protocol P' on G with $T+1$ rounds that at each time $t \in [T+1]$ has computed

$$P'_t(v; I) = (P_{t-1}(e; I))_{e \in M_G(v)}.$$

We argue inductively. Since $P_0 \equiv 0$, it's clear that this can be achieved for $t = 1$. Fix any $t > 1$ and suppose that $P'_{t-1}(u; I) = (P_{t-2}(e; I))_{e \in M_G(u)}$ for all $u \in V$ and inputs I. For each $v \in V$, we define a function $f'_{t,v}$ by

$$f'_{t,v}((c(v'))_{v' \in N_G(v)}, (I(e))_{e \in M_G(v)})_{e^\star} := f_{t-1,e^\star}((c(v)_e)_{e \in M_G(v)}, (c(v^\star)_e)_{e \in M_G(v^\star)}, I(e^\star))$$

for each $e^\star = (v, v^\star) \in M_G(v)$. Then by definition and the inductive hypothesis, we have

$$\begin{aligned} P'_t(v; I)_{e^\star} &= f'_{t,v}((P'_{t-1}(v'; I))_{v' \in N_G(v)}, (I(e))_{e \in M_G(v)})_{e^\star} \\ &= f_{t-1,e^\star}((P'_{t-1}(v; I)_e)_{e \in M_G(v)}, (P'_{t-1}(v^\star; I)_e)_{e \in M_G(v^\star)}, I(e^\star)) \\ &= f_{t-1,e^\star}((P_{t-2}(e; I))_{e \in M_G(v)}, (P_{t-2}(e; I)_e)_{e \in M_G(v^\star)}, I(e^\star)) \\ &= P_{t-1}(e^\star; I) \end{aligned}$$

for any edge $e^\star = (v, v^\star) \in E$, since $M_G(e) = M_G(v) \cup M_G(v^\star)$. This completes the induction and shows that $P'_{T+1}(v; I) = (P_T(e; I))_{e \in M_G(v)}$ for all v, I. Replacing $f'_{T+1,v}$ by $\tilde{f}_{T,v} \circ f'_{T+1,v}$ completes the proof. □

16.2 Omitted Proofs from Section 9.7

Proof of Theorem 13. Without loss of generality, we may assume that the functions $(f_t^{\mathsf{sym}})_{t \in [T]}$ and $\tilde{f}^{\mathsf{sym}}$ are all the identity function (on the appropriate domains). The reason is that any symmetric edge message-passing protocol $\tilde{P}$ on T rounds may be simulated by running P and then applying a universal function (depending only on $\tilde{P}$) to each node's output value – see Lemma 49.

We argue by induction that for each $t \in [T]$, there is a $(t+1)$-round symmetric node message-passing protocol that, on any input I, computes the function $Q_t(u; I) := \{\!\{P_t(e; I) : e \in M_G(u)\}\!\}$ for every node $u \in V$. Consider $t = 1$. For any $e = (u, v) \in E$, we have by symmetry and the initial assumption that

$$P_1(e; I) = (I(e), 0, \{\!\{\{\!\{0 : v' \in N_G(u)\}\!\}, \{\!\{0 : u' \in N_G(v)\}\!\}\}\!\}).$$

We define a two-round node message-passing protocol on G where the first update at node u computes

$$P_1'(u; I) = \{\!\{I(\{u, v\}) : v \in N_G(u)\}\!\}$$

and the second update at node u computes

$$\begin{aligned}(P_1'(u; I), \{\!\{(P_1'(v; I), I(\{u, v\})) : v \in N_G(u)\}\!\}) &\mapsto \{\!\{(I(\{u, v\}), 0, |N_G(u)|, |P_1'(v; I)|) : v \in N_G(u)\}\!\} \\ &\mapsto \{\!\{(I(\{u, v\}), 0, \{\!\{|N_G(u)|, |P_1'(v; I)|\}\!\}) : v \in N_G(u)\}\!\} \\ &= \{\!\{P_1(\{u, v\}; I) : v \in N_G(u)\}\!\} =: P_2'(u; I)\end{aligned}$$

since $|P_1'(v; I)| = |N_G(v)|$. By construction, this protocol is symmetric, which proves the induction for step $t = 1$.

Now pick any $t > 1$. For any $e = \{u, v\} \in E$, we have

$$P_t(e; I) = (I(e), P_{t-1}(e; I), \{\!\{Q_{t-1}(u; I), Q_{t-1}(v; I)\}\!\})$$

By the induction hypothesis, there is a t-round symmetric node message-passing protocol P' that, at node v on input I, computes

$$P_t'(v; I) = \{\!\{P_{t-1}(\{v, v'\}; I) : v' \in N_G(v)\}\!\} = Q_{t-1}(v; I).$$

Note that since $P_{t-1}(e; I)$ is an element of the tuple $P_t(e; I)$, for each $1 \le s \le t - 1$ there is a fixed function γ_s such that $\gamma_s(Q_{t-1}(v; I)) = Q_s(v; I)$ for all v, I. Using this fact, we extend P' to $t + 1$ rounds, defining the update at round $t + 1$ and node u as follows:

$$\begin{aligned}&(P_t'(u; I), \{\!\{(P_t'(v; I), I(\{u, v\})) : v \in N_G(u)\}\!\}) \\ &= (Q_{t-1}(u; I), \{\!\{(Q_{t-1}(v; I), I(\{u, v\})) : v \in N_G(u)\}\!\}) \\ &\mapsto (Q_{1:t-1}(u; I), \{\!\{(Q_{1:t-1}(v; I), I(\{u, v\})) : v \in N_G(u)\}\!\}) \\ &\mapsto (Q_{1:t-1}(u; I), \{\!\{(Q_{1:t-1}(v; I), I(\{u, v\})) : v \in N_G(u)\}\!\}) \\ &= \{\!\{(I(\{u, v\}), \{\!\{Q_{1:t-1}(u; I), Q_{1:t-1}(v; I)\}\!\}) : v \in N_G(u)\}\!\} \\ &\mapsto \{\!\{(I(\{u, v\}), P_{t-1}(\{u, v\}; I), \{\!\{Q_{t-1}(u; I), Q_{t-1}(v; I)\}\!\}) : v \in N_G(u)\}\!\} =: P_{t+1}'(u; I)\end{aligned}$$

where $Q_{1:t-1}(u; I)$ refers to the tuple $(Q_1(u; I), \ldots, Q_{t-1}(u; I))$. The first map is well-defined due to the existence of the functions $\gamma_1, \ldots, \gamma_{t-1}$, and the final map is well-defined because the definition of $P_{t-1}(\{u, v\}; I)$ can be iteratively unpacked, and it is ultimately a function of

$$(I(\{u, v\}), \{\!\{Q_{1:t-1}(u; I), Q_{1:t-1}(v; I)\}\!\}).$$

This shows that P' computes $Q_t(v; I)$ at node u on input I. By construction, P' is symmetric. This completes the induction. Since $Q_T(u; I)$ is precisely the output of P at node u on input I (after the node aggregation step), this shows that P can be simulated by a $(T+1)$-round symmetric node message-passing protocol on G. □

Lemma 49. *Let $T \ge 1$, and let $P = ((f_{t,e})_{t\in[T],e\in E}, (\tilde{f}_v)_{v\in V})$ be a symmetric edge message-passing protocol on $G = (V, E)$ with T rounds. Consider the T-round edge message-passing protocol $P^\circ = ((f^\circ_{t,e})_{t\in[T],e\in E}, (\tilde{f}^\circ_v)_{v\in V})$ where for all t, e,*

$$f^\circ_{t,e}((c(e'))_{e'\in M_G(e)}, I(e)) := (I(e), c(e), \{\!\{c(\{u, v'\}) : v' \in N_G(u)\}\!\}, \{\!\{c(\{u', v\}) : u' \in N_G(v)\}\!\}),$$

and for every $v \in V$,

$$\tilde{f}^{\circ}_v((c(e))_{e\in M_G(v)}) := \{\!\{c(e) : e \in M_G(v)\}\!\}.$$

Then there is a function h *such that* $\tilde{f}_v((P_T(e;I))_{e\in M_G(v)}) = h(\tilde{f}^{\circ}_v((P^{\circ}_T(e;I))_{e\in M_G(v)}))$ *for all* v, I.

Proof. We prove by induction that for each $t \in \{0,\dots,T\}$ there is a function h_t such that $P_t(e;I) = h_t(P^{\circ}_t(e;I))$ for all e, I. For $t = 0$ this is immediate from the convention that $P_0 \equiv P^{\circ}_0 \equiv 0$. Fix any $t \in \{1,\dots,T\}$. Since P is symmetric, there is a function f^{sym}_t so that for all $e = (u,v) \in E$ and inputs I,

$$\begin{aligned} P_t(e;I) &= f^{\mathsf{sym}}_t(I(e), P_{t-1}(e;I), \{\!\{P_{t-1}(\{u,v'\};I) : v' \sim u\}\!\}, \{\!\{P_{t-1}(\{u',v\};I) : u' \sim v\}\!\}) \\ &= f^{\mathsf{sym}}_t(I(e), h_{t-1}(P^{\circ}_{t-1}(e;I)), \{\!\{h_{t-1}(P^{\circ}_{t-1}(\{u,v'\};I)) : v' \sim u\}\!\}, \{\!\{h_{t-1}(P^{\circ}_{t-1}(\{u',v\};I)) : u' \sim v\}\!\}) \end{aligned}$$

which is indeed a well-defined function (independent of e, I) of

$$P^{\circ}_t(e;I) = (I(e), P^{\circ}_{t-1}(e;I), \{\!\{P^{\circ}_{t-1}(\{u,v'\};I) : v' \sim u\}\!\}, \{\!\{P^{\circ}_{t-1}(\{u',v\};I) : u' \sim v\}\!\}).$$

This completes the induction. Finally, since P is symmetric, there is a function $\tilde{f}^{\mathsf{sym}}$ such that $\tilde{f}_v((P_T(e;I))_{e\in M_G(v)}) = \tilde{f}^{\mathsf{sym}}(\{\!\{P_T(e;I) : e \in M_G(v)\}\!\})$ for all v, I. Hence we can write

$$\begin{aligned} \tilde{f}_v((P_T(e;I))_{e\in M_G(v)}) &= \tilde{f}^{\mathsf{sym}}(\{\!\{P_T(e;I) : e \in M_G(v)\}\!\}) \\ &= \tilde{f}^{\mathsf{sym}}(\{\!\{h_T(P^{\circ}_T(e;I)) : e \in M_G(v)\}\!\}) \end{aligned}$$

which is a well-defined function (independent of v, I) of $\{\!\{P^{\circ}_T(e;I) : e \in M_G(v)\}\!\}$ as needed. □

16.3 A quantitatively tight depth/memory separation

For each $n \in \mathbb{N}$, let $K_n := ([n], E_n)$ be the complete graph on $[n]$. In this section we show that there is a function that can be computed by an edge message-passing protocol on K_n with constant rounds and constant memory per processor, but for which any node message-passing protocol with T rounds and B bits of memory requires $TB \geq \Omega(n)$. We remark that this separation is quantitatively tight due to Proposition 12, although it is possible that a larger (e.g. even super-polynomial in n) depth separation may be possible if the node message-passing protocol is restricted to constant memory per processor.

At a technical level, the lower bound proceeds via a reduction from the *set disjointness problem* in communication complexity, similar to the lower bounds in Loukas [2019].

Definition 39. *Fix* $m \in \mathbb{N}$. *The set disjointness function* $\mathsf{DISJ}_m : \{0,1\}^m \times \{0,1\}^m \to \{0,1\}$ *is defined as*

$$\mathsf{DISJ}_m(A,B) := \mathbb{1}[\forall i \in [m] : A_i B_i = 0].$$

The following fact is well-known; see e.g. discussion in Håstad and Wigderson [2007].

Lemma 50. *In the two-party deterministic communication model, the deterministic communication complexity of* DISJ_m *is at least* m.

The main result of this section is the following:

Theorem 16. *Fix any even $n \in \mathbb{N}$. Define $g : \{0,1\}^{E_n} \to \{0,1\}^n$ by*

$$g(I)_v := \mathbb{1}[\exists \{i,j\} \in E_n : i,j \leq n/2 \wedge I(\{i,j\}) = I(\{n+1-i, n+1-j\}) = 1]$$

for all $I \in \{0,1\}^{E_n}$ and $v \in [n]$. Then the following properties hold:

- *Any node message-passing protocol on K_n with T rounds and B bits of memory that computes g requires $TB \geq \Omega(n)$*
- *There is an edge message-passing protocol on K_n with $O(1)$ rounds and $O(1)$ bits of memory that computes g.*

Proof. Let $m := \binom{n/2}{2}$. Let $P = (f_{t,v})_{t,v}$ be a node message-passing protocol on K_n that computes g with T rounds and B bits of memory. We design a two-party communication protocol for DISJ_m as follows. Suppose that Alice holds input $X \in \{0,1\}^m$ and Bob holds input $Y \in \{0,1\}^m$. Let us index the edges $\{i,j\} \in E_n$ with $i,j \leq n/2$ by $[m]$, and similarly index the edges $\{i,j\} \in E_n$ with $i,j > n/2$ by $[m]$, in such a way that edge $\{i,j\}$ has the same index as edge $\{n+1-i, n+1-j\}$. Let $I \in \{0,1\}^{E_n}$ be defined by

$$I(\{i,j\}) := \begin{cases} X_{\{i,j\}} & \text{if } i,j \leq n/2 \\ Y_{\{i,j\}} & \text{if } i,j > n/2 \\ 0 & \text{otherwise} \end{cases}.$$

Initially, Alice computes $\hat{P}_0(v) := 0$ for all $v \in \{1, \ldots, n/2\}$, and Bob computes $\hat{P}_0(v) := 0$ for all $v \in \{n/2+1, \ldots, n\}$. The communication protocol then proceeds in T rounds. At round $t \in [T]$, Alice sends $(\hat{P}_{t-1}(v))_{1 \leq v \leq n/2}$ to Bob, and Bob sends $(\hat{P}_{t-1}(v))_{n/2+1 \leq v \leq n}$ to Alice. Alice then computes

$$\hat{P}_t(v) := f_{t,v}((\hat{P}_{t-1}(v'))_{v' \in [n]}, (I(e))_{e \in M_{K_n}(v)})$$

for each $1 \leq v \leq n/2$, and Bob computes the same for each $n/2 < v \leq n$. Note that for any $i \leq n/2$ and edge $e \in M_{K_n}(i)$, Alice can compute $I(e)$. Similarly, for any $i > n/2$ and edge $e \in M_{K_n}(i)$, Bob can compute $I(e)$. Thus, this computation is well-defined. After round T, Alice and Bob output $1 - \hat{P}_T(1)$ and $1 - \hat{P}_T(n)$ respectively.

This defines a communication protocol. Since $\hat{P}_t(v) \in \{0,1\}^B$ for each $v \in [n]$ and $t \in [T]$, the total number of bits communicated is at most nBT. Moreover, by induction it's clear that Alice and Bob output $1 - P_T(1; I)$ and $1 - P_T(n; I)$ respectively. By assumption that P computes g and the fact that $g(I)_v = 1 - \mathsf{DISJ}_m(X, Y)$ for all $v \in [n]$, we have that $1 - P_T(1; I) = 1 - P_T(n; I) = 0$ if $\mathsf{DISJ}_m(I) = 0$, and $1 - P_T(1; I) = 1 - P_T(n; I) = 1$ if $\mathsf{DISJ}_m(I) = 1$. Thus, this communication protocol computes DISJ_m. By Lemma 50, it follows that $nBT \geq m = \Omega(n^2)$, so $BT = \Omega(n)$ as claimed.

Next, we exhibit an edge message-passing protocol on K_n that computes g with six rounds and one bit of memory. For $1 \le t \le 6$ and $e \in E_n$, define $f_{t,e} : \{0,1\}^{M_G(e)} \times \{0,1\} \to \{0,1\}$ as follows:

$$\begin{aligned}
f_{1,\{i,j\}}(x,y) &:= y \\
f_{2,\{i,j\}}(x,y) &:= x_{\{n+1-i,j\}} \\
f_{3,\{i,j\}}(x,y) &:= x_{\{i,n+1-j\}} \\
f_{4,\{i,j\}}(x,y) &:= \mathbb{1}[y = x_{\{i,j\}} \wedge i,j \le n/2] \\
f_{5,\{i,j\}}(x,y) &:= \mathbb{1}[\exists k \in [n] : x_{\{i,k\}} = 1] \\
f_{6,\{i,j\}}(x,y) &:= \mathbb{1}[\exists k \in [n] : x_{\{i,k\}} = 1].
\end{aligned}$$

Also define $\tilde{f}_v : \{0,1\}^{M_G(v)} \to \{0,1\}$ for each $v \in [n]$ by $\tilde{f}_v(x) := x_{\{x,1\}}$. It can be checked that the computation of P at timestep $t = 6$ is

$$P_6(\{i,j\}; I) := \mathbb{1}[\exists k, \ell \in [n/2] : I(\{k,\ell\}) = I(\{n+1-k, n+1-\ell\})] = g(I).$$

From the definition of $\tilde{f}$, it follows that P computes g. □

16.4 Further details on synthetic task over Ising models

16.4.1 Background on belief propagation

A classical way to calculate the marginals $\{\mathbb{E}[x_i]\}$ of an Ising model, when the associated graph is a tree, is to iterate the message passing algorithm:

$$\nu_{i\to j}^{(t+1)} = \tanh\left(h_i + \sum_{k\in\partial_i\setminus j} \tanh^{-1}\Big(\tanh(J_{ik})\nu_{k\to i}^{(t)}\Big)\right) \tag{16.1}$$

When the graph is a tree, it is a classical result ([Mezard and Montanari, 2009], Theorem 14.1) that the above message-passing algorithm converge to values ν^* that yield the correct marginals, namely:

$$\mathbb{E}[x_i] = \tanh\left(h_i + \sum_{k\in\partial_i} \tanh^{-1}(\tanh(J_{ik})\nu_{k\to i}^*)\right).$$

The reason the updates converge to the correct values on a tree topology is that they implicitly simulate a dynamic program. Namely, we can write down a recursive formula for the marginal of node i which depends on sums spanning each of the subtrees of the neighbors of i (i.e., for each neighbor j, the subgraph containing j that we would get if we removed edge $\{i,j\}$).

If we root the tree at an arbitrary node r, we can see that after completing a round of message passing from the leaves to the root, and another from the root to the leaves, each subtree of i will be (inductively) calculated correctly.

Moreover, even though the updates Equation 16.1 are written over edges, the dynamic programming view makes it clear an equivalent message-passing scheme can be written down where states are maintained over the *nodes* in the graph. Namely, for each node v, we can maintain two values $h_{v,\text{down}}$ and $h_{v,\text{up}}$,

which correspond to the values that will be used when v sends a message upwards (towards the root) or downwards (away from the root). Then, for appropriately defined functions F, G (depending on the potentials J and h), one can "simulate" the updates in Equation 16.1:

$$h^{(t+1)}_{v,\text{up}} \leftarrow F\left(\{h^{(t)}_{w,\text{up}} : w \in v \cup \text{Children}(v)\}\right) \tag{16.2}$$

$$h^{(t+1)}_{v,\text{down}} \leftarrow G\left(h^{(t)}_{\text{Parent}(v),\text{down}}, \left\{h^{(t)}_{w,\text{up}}\right\}_{w\in\text{Children}(v)}\right) \tag{16.3}$$

Intuitively, $h_{v,\text{up}}$ captures the effective external field induced by the subtree rooted at v on $\text{Parent}(v)$. After the upward messages propagate, the root r can compute its correct marginal. Once $h_{\text{Parent}(v),\text{down}}$ is the correct marginal for $\text{Parent}(v)$ at some step, $h_{v,\text{down}}$ will be the correct marginal for v at all subsequent steps.

16.4.2 GCN-based architectures to calculate marginals

The belief-propagation updates Equation 16.1 naturally fit the general edge-message passing paradigm from Equation 9.2. In fact, they fit even more closely a "directed" version of the paradigm, in which each edge $\{i, j\}$ maintains two embeddings $h_{i\to j}, h_{j\to i}$, such that the embedding for direction $h_{i\to j}$ depends on the embeddings $\{h_{k\to i}\}_{\{k,i\}\in E}$. With this modification to the standard edge GCN architecture Equation 9.4, it is straightforward to implement Equation 16.1 with one layer, using a particular choice of activation functions and weight matrices W (since, in particular, in our dataset all edge potentials $J_{i,j}$ are set to 1). Similarly, with a directed version of the node GCN architecture Equation 9.3, where each node maintains an "up" embedding as well as a "down" embedding, it is straightforward to implement the "node-based" dynamic programming solution Equation 16.2-Equation 16.3.

We call the architectures that do not maintain directionality Node-U and Edge-U (depending on whether they use a node-based or edge-based GCN). We call the "directed" architectures Node-D and Edge-D respectively. Since there are only initial node features (input as node potentials $\{h_i\}_{i\in}$), for the edge based architectures we initialize the edge features as a concatenation of the node features of the endpoints of the edge. The results we report for each architecture are the best over a sweep of depth $\in \{5, 10, 15, 20, 25, 30\}$ and width $\in \{10, 32, 64\}$.

16.4.3 Edge-based models improve over node-based models

In Figure 16.2 we show the results for several tree topologies: a complete binary tree (of size 31), a path graph (of size 30), and uniformly randomly chosen trees of size 30 (the results in Figure 16.2 are averaged over 3 samples of tree). The architectures in the legend (Node-U, Edge-U, Node-D, Edge-D) are based on a standard GCN, and detailed in Section 16.4.2

We can see that for both the undirected and directed versions, adding edge embeddings improves performance. The improved performance of all directed versions compared to their undirected counterpart is not very surprising: the standard, undirected GCN architecture treats all neighbors symmetrically —

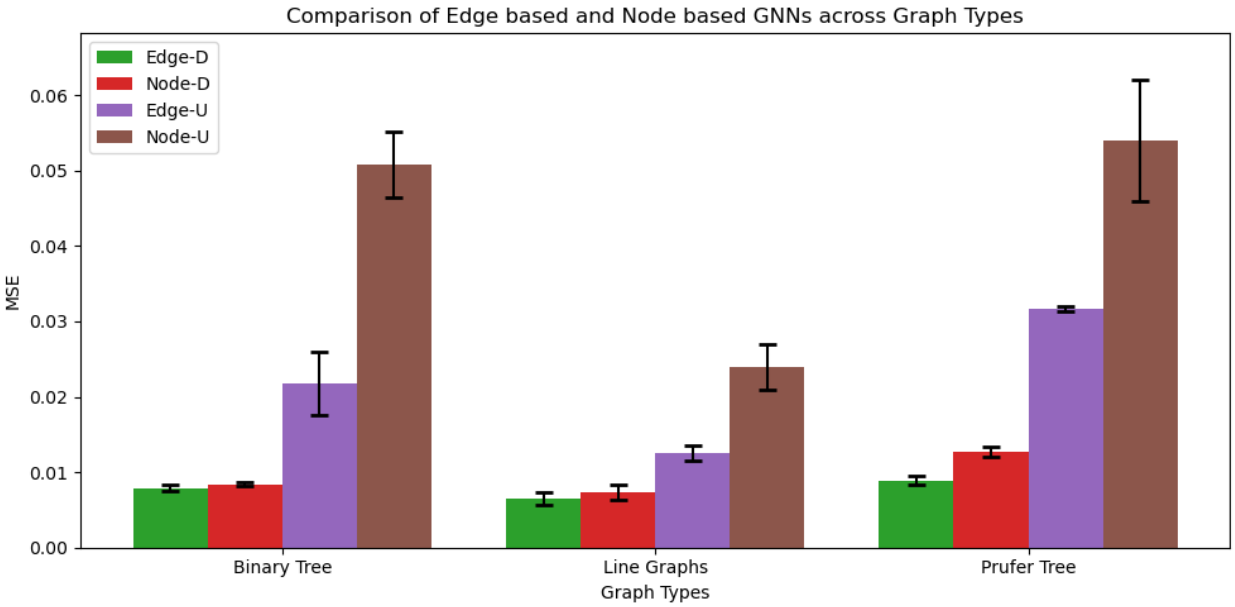


Figure 16.2: Comparison of four architectures for calculating node marginals in an Ising model. The architectures considered are node-embedding Equation 9.3 and edge-embedding Equation 9.4 versions of a GCN (correspondingly labeled Node-U and Edge-U), as well as their "directed" counterparts, as described in Section 16.4.2, correspondingly labeled Node-D and Edge-D. The x-axis groups results according to the topology of the graph, the y-axis is MSE (lower is better). The mean and variances are reported over 3 runs for the best choice of depth and width over the sweep described in Section 16.4.2.

hence, the directed versions can more easily simulate something akin to the belief propagation updates Equation 16.1 as well as the node-based dynamic programming Equation 16.2-Equation 16.3.

Bibliography

Karl Abrahamson. Time-space tradeoffs for algebraic problems on general sequential machines. *Journal of Computer and System Sciences*, 43(2):269–289, 1991.

William F Ames. *Numerical methods for partial differential equations*. Academic press, 2014.

Donald G Anderson. Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)*, 1965.

John David Anderson and J Wendt. *Computational fluid dynamics*, volume 206. Springer, 1995.

Cem Anil, Ashwini Pokle, Kaiqu Liang, Johannes Treutlein, Yuhuai Wu, Shaojie Bai, J Zico Kolter, and Roger B Grosse. Path independent equilibrium models can better exploit test-time computation. *Advances in Neural Information Processing Systems*, 35:7796–7809, 2022.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *Advances in Neural Information Processing Systems*, 32, 2019.

Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019.

Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.

Cx K Batchelor and George Keith Batchelor. *An introduction to fluid dynamics*. Cambridge university press, 1967.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29, 2016.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory, 2024.

Ali Behrouz and Farnoosh Hashemi. Graph mamba: Towards learning on graphs with state space models, 2024.

Abderrahmane Bendali and Keddour Lemrabet. The effect of a thin coating on the scattering of a time-harmonic wave for the helmholtz equation. *SIAM Journal on Applied Mathematics*, 56(6):1664–1693, 1996.

Kaushik Bhattacharya, Bamdad Hosseini, Nikola B Kovachki, and Andrew M Stuart. Model reduction and neural networks for parametric pdes. *arXiv preprint arXiv:2005.03180*, 2020.

Kaushik Bhattacharya, Bamdad Hosseini, Nikola B Kovachki, and Andrew M Stuart. Model reduction and neural networks for parametric PDEs. *The SMAI journal of computational mathematics*, 7:121–157, 2021.

Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of political economy*, 81(3):637–654, 1973.

Mitchell Black, Zhengchao Wan, Amir Nayyeri, and Yusu Wang. Understanding oversquashing in gnns through the lens of effective resistance. In *International Conference on Machine Learning*, pages 2528–2547. PMLR, 2023.

John P Boyd. *Chebyshev and Fourier spectral methods*. Courier Corporation, 2001.

Andres M Bran, Sam Cox, Andrew D White, and Philippe Schwaller. Chemcrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*, 2023.

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

Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

Heinz-Peter Breuer and Francesco Petruccione. *The Theory of Open Quantum Systems*. Oxford University Press, 2002.

Charles G Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of computation*, 1965.

Joan Bruna, Benjamin Peherstorfer, and Eric Vanden-Eijnden. Neural galerkin schemes with active learning for high-dimensional evolution equations. *Journal of Computational Physics*, 496:112588, 2024.

Johannes Martinus Burgers. *The nonlinear diffusion equation: asymptotic solutions and statistical problems*. Springer Science & Business Media, 2013.

Lei Cai, Jundong Li, Jie Wang, and Shuiwang Ji. Line graph neural networks for link prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):5103–5113, 2021.

Shuhao Cao. Choose a transformer: Fourier or Galerkin. *Advances in Neural Information Processing Systems (NeurIPS 2021)*, 34, 2021. URL https://openreview.net/forum?id=ssohLcmn4-r.

Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995.

Zhengdao Chen, Xiang Li, and Joan Bruna. Supervised community detection with line graph neural networks. *arXiv preprint arXiv:1705.08415*, 2017.

Ziang Chen, Jianfeng Lu, and Yulong Lu. On the representation of solutions to elliptic PDEs in Barron spaces. *Advances in Neural Information Processing Systems*, 34, 2021.

Ziang Chen, Jianfeng Lu, Yulong Lu, and Shengxuan Zhou. A regularity theory for static Schrödinger equations on $\mathbb{R}^d$ in spectral Barron spaces. *arXiv preprint arXiv:2201.10072*, 2022.

Hongwei Cheng, William Y Crutchfield, Zydrunas Gimbutas, Leslie F Greengard, J Frank Ethridge, Jingfang Huang, Vladimir Rokhlin, Norman Yarvin, and Junsheng Zhao. A wideband fast multipole method for the helmholtz equation in three dimensions. *Journal of Computational Physics*, 216(1): 300–325, 2006.

Kamal Choudhary and Brian DeCost. Atomistic line graph neural network for improved materials property predictions. *npj Computational Materials*, 7(1):185, 2021.

Demetrios Christodoulou. *The formation of shocks in 3-dimensional fluids*, volume 2. European Mathematical Society, 2007.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models, 2022.

John Crank and Phyllis Nicolson. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 43, pages 50–67. Cambridge University Press, 1947.

Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. *arXiv preprint arXiv:2003.04630*, 2020.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024a.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning (ICML)*, 2024b.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning (ICML)*, 2024c.

Chandler Davis and William Morton Kahan. The rotation of eigenvectors by a perturbation. iii. *SIAM Journal on Numerical Analysis*, 7(1):1–46, 1970.

Memoria di Ennio De Giorgi. Sulla differenziabilitae l'analiticita delle estremali degli integrali multipli regolari. *Ennio De Giorgi*, page 167, 1957.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *NAACL*, 2019.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *International Conference on Learning Representations*, 2021a.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021b.

Gideon Dresdner, Dmitrii Kochkov, Peter Norgaard, Leonardo Zepeda-Núñez, Jamie A. Smith, Michael P. Brenner, and Stephan Hoyer. Learning to correct spectral methods for simulating turbulent flows. 2022. doi: 10.48550/ARXIV.2207.00556. URL https://arxiv.org/abs/2207.00556.

Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. *arXiv preprint arXiv:2110.07875*, 2021.

Vijay Prakash Dwivedi, Ladislav Rampášek, Michael Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. *Advances in Neural Information Processing Systems*, 35:22326–22340, 2022.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023.

Matthias Ehrhardt and Ronald E Mickens. A fast, stable and accurate numerical method for the black–scholes equation of american options. *International Journal of Theoretical and Applied Finance*, 11 (05):471–501, 2008.

Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on learning theory*, pages 907–940. PMLR, 2016.

Björn Engquist and Lexing Ying. Fast directional multilevel algorithms for oscillatory kernels. *SIAM Journal on Scientific Computing*, 29(4):1710–1737, 2007.

Lawrence C Evans. *Partial Differential Equations*. graduate studies in mathematics. american mathematical society, 1998. ISBN 9780821807729.

Lawrence C Evans. *Partial differential equations*, volume 19. American Mathematical Soc., 2010.

I Faragó and J Karátson. The gradient-finite element method for elliptic problems. *Computers & Mathematics with Applications*, 42(8-9):1043–1053, 2001.

István Faragó and János Karátson. *Numerical solution of nonlinear elliptic problems via preconditioning operators: Theory and applications*, volume 11. Nova Publishers, 2002.

Xavier Fernández-Real and Xavier Ros-Oton. Regularity theory for elliptic PDE. *Forthcoming book*, 2020.

Daniel Y Fu, Simran Arora, Jessica Grogan, Isys Johnson, Sabri Eyuboglu, Armin W Thomas, Benjamin Spector, Michael Poli, Atri Rudra, and Christopher Ré. Monarch mixer: A simple sub-quadratic gemm-based architecture. *NeurIPS*, 2023.

Samy Wu Fung, Howard Heaton, Qiuwei Li, Daniel McKenzie, Stanley Osher, and Wotao Yin. Jfb: Jacobian-free backpropagation for implicit networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 6648–6656, 2022.

Zhengyang Geng, Xin-Yu Zhang, Shaojie Bai, Yisen Wang, and Zhouchen Lin. On training implicit models. *Advances in Neural Information Processing Systems*, 34:24247–24260, 2021.

David Gilbarg and Neil S Trudinger. Elliptic partial differential equations of second order. 2001.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

A. Gouasmi, E.J. Parish, and K. Duraisamy. A priori estimation of memory effects in reduced-order models of nonlinear systems using the mori–zwanzig formalism. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 473(2205):20170385, 2017. doi: 10.1098/rspa.2017.0385. URL http://dx.doi.org/10.1098/rspa.2017.0385.

Dmitrii Yur'evich Grigor'ev. Application of separability and independence notions for proving lower bounds of circuit complexity. *Zapiski Nauchnykh Seminarov POMI*, 60:38–48, 1976.

Philipp Grohs and Lukas Herrmann. Deep neural network approximation for high-dimensional elliptic pdes with boundary conditions. *arXiv preprint arXiv:2007.05384*, 2020.

Philipp Grohs, Fabian Hornung, Arnulf Jentzen, and Philippe Von Wurstemberger. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of black-scholes partial differential equations. *arXiv preprint arXiv:1809.02362*, 2018.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023a.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023b.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *ICLR*, 2022a.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *The International Conference on Learning Representations (ICLR)*, 2022b.

Albert Gu, Isys Johnson, Aman Timalsina, Atri Rudra, and Christopher Re. How to train your HIPPO: State space models with generalized orthogonal basis projections. In *International Conference on Learning Representations*, 2023.

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

Jayesh K Gupta and Johannes Brandstetter. Towards multi-spatiotemporal-scale generalized PDE modeling. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856.

Jiequn Han, Arnulf Jentzen, and E Weinan. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

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, 2023a.

Zhongkai Hao, Chengyang Ying, Zhengyi Wang, Hang Su, Yinpeng Dong, Songming Liu, Ze Cheng, Jun Zhu, and Jian Song. Gnot: A general neural operator transformer for operator learning. *arXiv preprint arXiv:2302.14376*, 2023b.

Zhongkai Hao, Chang Su, Songming Liu, Julius Berner, Chengyang Ying, Hang Su, Anima Anandkumar, Jian Song, and Jun Zhu. DPOT: Auto-regressive denoising operator transformer for large-scale PDE pre-training. March 2024a. URL https://github.com/thu-ml/DPOT.

Zhongkai Hao, Chang Su, Songming Liu, Julius Berner, Chengyang Ying, Hang Su, Anima Anandkumar, Jian Song, and Jun Zhu. Dpot: Auto-regressive denoising operator transformer for large-scale pde pre-training, 2024b.

Johan Håstad and Avi Wigderson. The randomized communication complexity of set disjointness. *Theory of Computing*, 3(1):211–219, 2007.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Byeongho Heo, Song Park, Dongyoon Han, and Sangdoo Yun. Rotary position embedding for vision transformer. *arXiv preprint arXiv:2403.13298*, 2024.

Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers. *International Conference on Learning Representations*, 2020.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural networks*, 3(5):551–560, 1990.

Jun-Ting Hsieh, Shengjia Zhao, Stephan Eismann, Lucia Mirabella, and Stefano Ermon. Learning neural pde solvers with convergence guarantees. *arXiv preprint arXiv:1906.01200*, 2019.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

Ningyuan Teresa Huang and Soledad Villar. A short tutorial on the weisfeiler-lehman test and its variants. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8533–8537. IEEE, 2021.

Martin Hutzenthaler, Arnulf Jentzen, Thomas Kruse, and Tuan Anh Nguyen. A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations. *SN Partial Differential Equations and Applications*, 1:1–34, 2020.

Sukjun Hwang, Aakash Lahoti, Tri Dao, and Albert Gu. Hydra: Bidirectional state space models through generalized matrix mixers. *arXiv preprint arXiv:2407.09941*, 2024.

Ernst Ising. *Beitrag zur theorie des ferro-und paramagnetismus*. PhD thesis, Grefe & Tiedemann Hamburg, Germany, 1924.

Arnulf Jentzen, Diyora Salimova, and Timo Welti. A proof that deep artificial neural networks overcome the curse of dimensionality in the numerical approximation of kolmogorov partial differential equations with constant diffusion and nonlinear drift coefficients. *arXiv preprint arXiv:1809.07321*, 2018.

Marcin P Joachimiak, J Harry Caufield, Nomi L Harris, Hyeongsik Kim, and Christopher J Mungall. Gene set summarization using large language models. *ArXiv*, 2023.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *ICML*, 2020.

Aaron Kelly, Andrés Montoya-Castillo, Lu Wang, and Thomas E. Markland. Generalized quantum master equations in and out of equilibrium: When can one win? *The Journal of Chemical Physics*, 144 (18):184105, 05 2016. ISSN 0021-9606. doi: 10.1063/1.4948612. URL https://doi.org/10.1063/1.4948612.

Yuehaw Khoo and Lexing Ying. Switchnet: a neural network model for forward and inverse scattering problems. *SIAM Journal on Scientific Computing*, 41(5):A3182–A3201, 2019.

Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric pde problems with artificial neural networks. *arXiv preprint arXiv:1707.03351*, 2017.

Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric PDE problems with artificial neural networks. *European Journal of Applied Mathematics*, 32(3):421–435, 2021.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

Miglena N Koleva and Lubin G Vulkov. Numerical solution of the Monge-Ampère equation with an application to fluid dynamics. In *AIP Conference Proceedings*, volume 2048, page 030002. AIP Publishing LLC, 2018.

David A Kopriva. *Implementing spectral methods for partial differential equations: Algorithms for scientists and engineers*. Springer Science & Business Media, 2009.

Nikola Kovachki, Samuel Lanthaler, and Siddhartha Mishra. On universal approximation and error bounds for Fourier neural operators. *The Journal of Machine Learning Research*, 22(1):13237–13312, 2021a.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021b.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to PDEs. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. Curran Associates, Inc., 2012.

Gitta Kutyniok, Philipp Petersen, Mones Raslan, and Reinhold Schneider. A theoretical analysis of deep neural networks and parametric pdes. *arXiv preprint arXiv:1904.00377*, 2019.

Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

Isaac E Lagaris, Aristidis C Likas, and Dimitris G Papageorgiou. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, 11(5):1041–1049, 2000.

Samuel Lanthaler, Siddhartha Mishra, and George E Karniadakis. Error estimates for deeponets: A deep learning framework in infinite dimensions. *Transactions of Mathematics and Its Applications*, 6(1): tnac001, 2022.

Peter D Lax and Arthur N Milgram. Parabolic equations, volume 33 of annals of mathematics studies, 1954.

Holden Lee, Rong Ge, Tengyu Ma, Andrej Risteski, and Sanjeev Arora. On the ability of neural nets to express distributions. *arXiv preprint arXiv:1702.07028*, 2017.

James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontanon. Fnet: Mixing tokens with fourier transforms. *NAACL*, 2022.

AA Leman and Boris Weisfeiler. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9):12–16, 1968.

Randall J LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM, 2007.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35: 3843–3857, 2022.

Zijie Li, Kazem Meidani, and Amir Barati Farimani. Transformer for partial differential equations' operator learning. *arXiv preprint arXiv:2205.13671*, 2022.

Zijie Li, Kazem Meidani, and Amir Barati Farimani. Transformer for partial differential equations' operator learning. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=EPPqt3uERT.

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*, 2020a.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020b.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *Proceedings of the International Conference on Learning Representations (ICLR)*. ICLR, 2021a.

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. *arXiv preprint arXiv:2111.03794*, 2021b.

Jinbi Liang and Cunlai Pu. Line graph neural networks for link weight prediction. *arXiv preprint arXiv:2309.15728*, 2023.

Renjie Liao, Yuwen Xiong, Ethan Fetaya, Lisa Zhang, KiJung Yoon, Xaq Pitkow, Raquel Urtasun, and Richard Zemel. Reviving and improving recurrent back-propagation. In *International Conference on Machine Learning*, pages 3082–3091. PMLR, 2018.

Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on computing*, 21(1):193–201, 1992.

Phillip Lippe, Bastiaan S Veeling, Paris Perdikaris, Richard E Turner, and Johannes Brandstetter. Pde-refiner: Achieving accurate long rollouts with neural pde solvers. *arXiv preprint arXiv:2308.05732*, 2023a.

Phillip Lippe, Bastiaan S. Veeling, Paris Perdikaris, Richard E Turner, and Johannes Brandstetter. PDE-refiner: Achieving accurate long rollouts with neural PDE solvers. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023b.

Xinliang Liu, Bo Xu, and Lei Zhang. Mitigating spectral bias for the multiscale operator learning with hierarchical attention. October 2022. URL http://arxiv.org/abs/2210.10890.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin Transformer: Hierarchical vision Transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

Andreas Loukas. What graph neural networks cannot learn: depth vs width. *arXiv preprint arXiv:1907.03199*, 2019.

Andreas Loukas. How hard is to distinguish graphs with graph neural networks? *Advances in neural information processing systems*, 33:3465–3476, 2020.

Jianfeng Lu and Yulong Lu. A priori generalization error analysis of two-layer neural networks for solving high dimensional Schrödinger eigenvalue problems. *arXiv preprint arXiv:2105.01228*, 2021.

Jianfeng Lu, Yulong Lu, and Min Wang. A priori generalization analysis of the deep ritz method for solving high dimensional elliptic equations. *arXiv preprint arXiv:2101.01708*, 2021.

Jiasen Lu, Christopher Clark, Sangho Lee, Zichen Zhang, Savya Khosla, Ryan Marten, Derek Hoiem, and Aniruddha Kembhavi. Unified-io 2: Scaling autoregressive multimodal models with vision, language, audio, and action. *ArXiv*, abs/2312.17172, 2023. URL https://api.semanticscholar.org/CorpusID:266573555.

Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. Frozen pretrained transformers as universal computation engines. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(7):7628–7636, Jun. 2022.

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.

Chao Ma, Jianchun Wang, and E Weinan. Model reduction with memory and the machine learning of dynamical systems. August 2018. URL http://arxiv.org/abs/1808.04258.

Alaeddin Malek and R Shekari Beidokhti. Numerical solution for high order differential equations using a hybrid neural network—optimization method. *Applied Mathematics and Computation*, 183(1): 260–271, 2006.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *Advances in neural information processing systems*, 32, 2019.

Tanya Marwah, Zachary C Lipton, and Andrej Risteski. Parametric complexity bounds for approximating pdes with neural networks. *arXiv preprint arXiv:2103.02138*, 2021.

Tanya Marwah, Zachary C Lipton, Jianfeng Lu, and Andrej Risteski. Neural network approximations of PDEs beyond linearity: Representational perspective. *arXiv preprint arXiv:2210.12101*, 2022.

Tanya Marwah, Ashwini Pokle, J Zico Kolter, Zachary Lipton, Jianfeng Lu, and Andrej Risteski. Deep equilibrium based neural operators for steady-state pdes. *Advances in Neural Information Processing Systems*, 36:15716–15737, 2023.

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.

Nick McGreivy and Ammar Hakim. Weak baselines and reporting biases lead to overoptimism in machine learning for fluid-related partial differential equations. *Nature Machine Intelligence*, 2024. doi: 10.1038/s42256-024-00897-5. URL https://doi.org/10.1038/s42256-024-00897-5. Published on September 25, 2024.

Marc Mezard and Andrea Montanari. *Information, physics, and computation*. Oxford University Press, 2009.

Siddhartha Mishra and Roberto Molinaro. Estimates on the generalization error of physics informed neural networks (PINNs) for approximating PDEs. *arXiv preprint arXiv:2006.16144*, 2020.

Andrés Montoya-Castillo and David R. Reichman. Approximate but accurate quantum dynamics from the Mori formalism: I. Nonequilibrium dynamics. *The Journal of Chemical Physics*, 144(18):184104, 05 2016. ISSN 0021-9606. doi: 10.1063/1.4948408. URL https://doi.org/10.1063/1.4948408.

Hazime Mori. Transport, collective motion, and brownian motion. *Progress of theoretical physics*, 33(3): 423–455, 1965.

Ben Moseley, Andrew Markham, and Tarje Nissen-Meyer. Solving the wave equation with physics-informed deep learning. *arXiv preprint arXiv:2006.11894*, 2020.

Fadl Moukalled, Luca Mangani, Marwan Darwish, F Moukalled, L Mangani, and M Darwish. *The finite volume method*. Springer, 2016.

Sadao Nakajima. On Quantum Theory of Transport Phenomena: Steady Diffusion. *Progress of Theoretical Physics*, 20(6):948–959, 12 1958. ISSN 0033-068X. doi: 10.1143/PTP.20.948. URL https://doi.org/10.1143/PTP.20.948.

John Nash. Parabolic equations. *Proceedings of the National Academy of Sciences*, 43(8):754–758, 1957.

John Nash. Continuity of solutions of parabolic and elliptic equations. *American Journal of Mathematics*, 80(4):931–954, 1958.

Claude Navier. *Mémoire sur les lois du mouvement des fluides*. éditeur inconnu, 1822.

CLMH Navier. Sur les lois des mouvement des fluides, en ayant egard a l'adhesion des molecules. In *Annales de Chimie et de Physique*, volume 19, page 1821. Lavoisier Paris, France, 1821.

John Neuberger. *Sobolev gradients and differential equations*. Springer Science & Business Media, 2009.

Louis Nirenberg. Remarks on strongly elliptic partial differential equations. *Communications on pure and applied mathematics*, 8(4):648–674, 1955.

Silvia Noschese, Lionello Pasquini, and Lothar Reichel. Tridiagonal toeplitz matrices: properties and novel applications. *Numerical linear algebra with applications*, 20(2):302–326, 2013.

Bernt Oksendal. *Stochastic differential equations: an introduction with applications*. Springer Science & Business Media, 2013.

Lars Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Physical Review*, 65(3-4):117, 1944.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, 2019.

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. 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 26670–26698. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/orvieto23a.html.

M Necati Özişik, Helcio RB Orlande, Marcelo J Colaço, and Renato M Cotta. *Finite difference methods in heat transfer*. CRC press, 2017.

Eric Parish and Karthik Duraisamy. Non-markovian closure models for large eddy simulations using the mori-zwanzig formalism. *Physical Review Fluids*, 2:014604, 01 2017. doi: 10.1103/PhysRevFluids.2.014604.

Suhas V Patankar and D Brian Spalding. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. In *Numerical prediction of flow, heat transfer, turbulence and combustion*, pages 54–73. Elsevier, 1983.

Ravi G Patel, Nathaniel A Trask, Mitchell A Wood, and Eric C Cyr. A physics-informed operator regression framework for extracting data-driven continuum models. *Computer Methods in Applied Mechanics and Engineering*, 373:113500, 2021.

Lawrence E Payne and Hans F Weinberger. An optimal Poincaré inequality for convex domains. *Archive for Rational Mechanics and Analysis*, 5(1):286–292, 1960.

David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran Gv, Xuzheng He, Haowen Hou, Jiaju Lin, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartlomiej Koptyra, Hayden Lau, Krishna Sri Ipsit Mantri, Ferdinand Mom, Atsushi Saito, Guangyu Song, Xiangru Tang, Bolun Wang, Johan S Wind, Stanislaw Wozniak, Ruichong Zhang, Zhenyuan Zhang, Qihang Zhao, Peng Zhou, Qinghua Zhou, Jian Zhu, and Rui-Jie Zhu. RWKV: Reinventing RNNs for the transformer era. May 2023. URL http://arxiv.org/abs/2305.13048.

Dmytro Perekrestenko, Philipp Grohs, Dennis Elbrächter, and Helmut Bölcskei. The universal approximation power of finite-width deep relu networks. *arXiv preprint arXiv:1806.01528*, 2018.

Henri Poincaré. Sur les équations aux dérivées partielles de la physique mathématique. *American Journal of Mathematics*, pages 211–294, 1890.

Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y. Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models. In *ICML*, 2023.

Stephen B Pope. Turbulent flows. *Measurement Science and Technology*, 12(11):2020–2021, 2001.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 2020a.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020b.

Md Ashiqur Rahman, Zachary E Ross, and Kamyar Azizzadenesheli. U-NO: U-shaped neural operators. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856.

Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

Maziar Raissi, Paris Perdikaris, and George 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.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515, 2022.

Michael Reed and Barry Simon. *Methods of modern mathematical physics. vol. 1. Functional analysis*. Academic San Diego, 1980.

Max Revay, Ruigang Wang, and Ian R Manchester. Lipschitz bounded equilibrium networks. *arXiv preprint arXiv:2010.01732*, 2020.

Nicholas Roberts, Samuel Guo, Cong Xu, Ameet Talwalkar, David Lander, Lvfang Tao, Linhang Cai, Shuaicheng Niu, Jianyu Heng, Hongyang Qin, Minwen Deng, Johannes Hog, Alexander Pfefferle, Sushil Ammanaghatta Shivakumar, Arjun Krishnakumar, Yubo Wang, Rhea Sanjay Sukthanker, Frank Hutter, Euxhen Hasanaj, Tien-Dung Le, Mikhail Khodak, Yuriy Nevmyvaka, Kashif Rasul, Frederic Sala, Anderson Schneider, Junhong Shen, and Evan R. Sparks. Automl decathlon: Diverse tasks, modern methods, and efficiency at scale. In *Neural Information Processing Systems*, 2021. URL https://api.semanticscholar.org/CorpusID:265536645.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *ArXiv*, abs/1505.04597, 2015. URL https://api.semanticscholar.org/CorpusID:3719281.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Clayton Sanford, Daniel Hsu, and Matus Telgarsky. Transformers, parallel computation, and logarithmic depth. *arXiv preprint arXiv:2402.09268*, 2024a.

Clayton Sanford, Daniel J Hsu, and Matus Telgarsky. Representational strengths and limitations of transformers. *Advances in Neural Information Processing Systems*, 36, 2024b.

Erik Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Conference on Computational Natural Language Learning*, 2003. URL https://api.semanticscholar.org/CorpusID:2470716.

John E Savage. *Models of computation*, volume 136. Addison-Wesley Reading, 1998.

Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

Junhong Shen, Mikhail Khodak, and Ameet Talwalkar. Efficient architecture search for diverse tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Junhong Shen, Liam Li, Lucio M. Dery, Corey Staten, Mikhail Khodak, Graham Neubig, and Ameet Talwalkar. Cross-modal fine-tuning: align then refine. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Junhong Shen, Neil Tenenholtz, James Brian Hall, David Alvarez-Melis, and Nicolo Fusi. Tag-llm: Repurposing general-purpose llms for specialized domains, 2024.

Qiang Shi and Eitan Geva. A new approach to calculating the memory kernel of the generalized quantum master equation for an arbitrary system–bath coupling. *The Journal of chemical physics*, 119(23): 12063–12076, 2003.

Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J Sutherland, and Ali Kemal Sinop. Exphormer: Sparse transformers for graphs. In *International Conference on Machine Learning*, pages 31613–31632. PMLR, 2023.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

J Sirignano and K Spiliopoulos DGM. A deep learning algorithm for solving partial differential equations. *ArXiv e-prints*, 2017.

Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

Casper Kaae Sønderby, Lasse Espeholt, Jonathan Heek, Mostafa Dehghani, Avital Oliver, Tim Salimans, Shreya Agrawal, Jason Hickey, and Nal Kalchbrenner. Metnet: A neural weather model for precipitation forecasting. *arXiv preprint arXiv:2003.12140*, 2020.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023. URL https://arxiv.org/abs/2104.09864.

Shashank Subramanian, Peter Harrington, Kurt Keutzer, Wahid Bhimji, Dmitriy Morozov, Michael Mahoney, and Amir Gholami. Towards foundation models for scientific machine learning: Characterizing scaling and transfer behavior. *arXiv preprint arXiv:2306.00258*, 2023.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models, 2023.

Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Daniel MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. Pdebench: An extensive benchmark for scientific machine learning. *Advances in Neural Information Processing Systems*, 35:1596–1611, 2022.

Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Dan MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. PDEBENCH: An extensive benchmark for scientific machine learnin. In *ICLR 2023 Workshop on Physics for Machine Learning*, 2023.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020.

Matus Telgarsky. Benefits of depth in neural networks. In *Conference on learning theory*, pages 1517–1539. PMLR, 2016.

Matus Telgarsky. Neural networks and rational functions. *arXiv preprint arXiv:1706.03301*, 2017.

Roger Temam. *Navier-Stokes equations: theory and numerical analysis*, volume 343. American Mathematical Soc., 2001.

Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *NeurIPS*, 2021.

Jan Tönshoff, Martin Ritzert, Eran Rosenbluth, and Martin Grohe. Where did the gap go? reassessing the long-range graph benchmark. *arXiv preprint arXiv:2309.00367*, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *ArXiv*, abs/2302.13971, 2023a. URL https://api.semanticscholar.org/CorpusID:257219404.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023b. URL https://arxiv.org/abs/2302.13971.

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

Alasdair Tran, Alexander Mathews, Lexing Xie, and Cheng Soon Ong. Factorized fourier neural operators. In *The Eleventh International Conference on Learning Representations*, 2023.

Renbo Tu, Nicholas Roberts, Mikhail Khodak, Junhong Shen, Frederic Sala, and Ameet Talwalkar. NAS-bench-360: Benchmarking neural architecture search on diverse tasks. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2022.

Kathryn Tunyasuvunakool, Jonas Adler, Zachary Wu, Tim Green, Michal Zielinski, Augustin Žídek, Alex Bridgland, Andrew Cowie, Clemens Meyer, Agata Laydon, et al. Highly accurate protein structure prediction for the human proteome. *Nature*, 596(7873):590–596, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 2017a.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017b.

Ria Vinod, Pin-Yu Chen, and Payel Das. Reprogramming pretrained language models for protein sequence representation learning. *arXiv preprint arXiv:2301.02120*, 2023.

Alex Wang. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*, 2019.

Chloe Wang, Oleksii Tsepa, Jun Ma, and Bo Wang. Graph-mamba: Towards long-range graph sequence modeling with selective state spaces, 2024.

Ziming Wang, Tao Cui, and Xueshuang Xiang. A neural network with plane wave activation for helmholtz equation, 2020.

E Weinan and Bing Yu. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

E Weinan, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, 2017.

Hilary Weller, Philip Browne, Chris Budd, and Mike Cullen. Mesh adaptation on the sphere using optimal transport and the numerical solution of a Monge–Ampère type equation. *Journal of Computational Physics*, 308:102–123, 2016.

Ezra Winston and J Zico Kolter. Monotone operator equilibrium networks. *Advances in neural information processing systems*, 33:10718–10728, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Tuo Xu and Lei Zou. Rethinking and extending the probabilistic inference capacity of gnns. In *The Twelfth International Conference on Learning Representations*, 2023.

Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103–114, 2017.

Luo Yining, Chen Yingfa, and Zhang Zhen. Cfdbench: A large-scale benchmark for machine learning methods in fluid dynamics. 2023. URL https://arxiv.org/abs/2310.05963.

Bing Yu et al. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *arXiv preprint arXiv:1710.00211*, 2017.

Bing Yu et al. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

Ming-Liang Zhang, Being J Ka, and Eitan Geva. Nonequilibrium quantum dynamics in the condensed phase via the generalized quantum master equation. *The Journal of chemical physics*, 125(4), 2006.

Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. From stars to subgraphs: Uplifting any gnn with local structure awareness. *arXiv preprint arXiv:2110.03753*, 2021.

Robert Zwanzig. Memory effects in irreversible thermodynamics. *Physical Review*, 124(4):983, 1961.

Robert Zwanzig. *Nonequilibrium Statistical Mechanics*. Oxford University Press, New York, 2001. ISBN 9780195140187.