THE UNIVERSITY OF CHICAGO

IMPROVING SCIENTIFIC MACHINE LEARNING WITH ALGORITHMIC INSIGHTS FROM NUMERICAL ANALYSIS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
OWEN JAMES MELIA

CHICAGO, ILLINOIS
AUGUST 2025

ABSTRACT

Scientific machine learning offers methods of building approximate but efficient solutions to challenging tasks in scientific computing, such as simulating a physical phenomenon or inferring high-dimensional model parameters from data. Algorithms from numerical analysis provide classical computational solutions, but these algorithms are not designed to learn from large datasets, and they can be expensive to run for certain problems and levels of error tolerance. Machine learning solutions offer approximations to these classical methods, often using neural networks, which can learn from large datasets. While general-purpose neural network architectures and training algorithms are frequently a natural first choice for such problems, scientific computing tasks are often ill-conditioned and require a level of accuracy unattainable by general-purpose methods. At the same time, numerical analysis research provides a wealth of insights for principled methods of computing solutions to these problems.

This thesis develops state-of-the-art machine learning methods for scientific computing problems by applying insights from contemporary algorithms research in numerical analysis. We consider three different problem settings in scientific computing. The first example uses insights from applied harmonic analysis to derive rotation-invariant random feature models, which provide an attractive alternative to deep neural networks or hand-designed kernels. In another setting, insights from optimization theory and recursive linearization algorithms allow us to design simple yet accurate neural networks for multi-frequency inverse scattering problems in a highly nonlinear regime, when training data is available. In the final setting, this thesis considers a broader class of partial differential equation (PDE)-based inverse problems, in a setting where training data is scarce. In this setting, the thesis contributes develops software and algorithms for accelerating highly-efficient and highly-accurate fast direct solvers of elliptic PDEs on hardware acceleration devices. The final chapter of this thesis develops optimization methods applying these novel algorithm and software tools to

multiple PDE-based inverse problems in imaging.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

The projects described in this thesis are the results of collaborations with many researchers. Some chapters of this thesis appear as published works.

Chapter 2 appears as the publication "Rotation-Invariant Random Features Provide a Strong Baseline for Machine Learning on 3D Point Clouds", published in Transactions on Machine Learning Research in 2023 [Melia et al., 2023]. This was the result of a collaboration with Eric Jonas and Rebecca Willett (thesis advisor).

Chapter 3 was published as "Multi-Frequency Progressive Refinement for Learned Inverse Scattering" in the Journal of Computational Physics in 2025 [Melia et al., 2025b]. This project was a collaboration with Olivia Tsang, Vasileios Charisopoulos, Yuehaw Khoo, Jeremy Hoskins (thesis committee member), and Rebecca Willett (thesis advisor).

Chapter 4 is from the manuscript "Hardware Acceleration for HPS Algorithms in Two and Three Dimensions", which is available online [Melia et al., 2025a] and is under review at the Journal of Computational Physics. This work was the result of a collaboration with Daniel Fortunato, Jeremy Hoskins (thesis committee member), and Rebecca Willett (thesis advisor). Chapter 5 is from the manuscript "jaxhps: An Elliptic PDE Solver Built with Machine Learning in Mind", which is authored by the same collaboration and is under review at the Journal of Open Source Software.

Chapter 6 is an early version of a manuscript in preparation. This is a result of a collaboration with Vasileios Charisopoulos and Rebecca Willett (thesis advisor).

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Rebecca Willett. Becca has been a constant source of advice and support over the past five years, and I can't imagine having a better advisor. When I first started in grad school, Becca suggested that I work on scientific machine learning, and a year later, when I wasn't making much progress, Becca had the patience and faith to switch directions. I have received incredible training from Becca, and I have learned far too many things to list here. Becca's example has taught me how to "do", and also how to "be": hardworking, curious, respectful, rigorous, and optimistic. Becca sends extremely well-prepared students out into the world, and I count myself lucky to be one of them.

I am incredibly grateful to Jeremy Hoskins, one of my collaborators and committee members. Jeremy has introduced me to research in applied math and PDEs. I am grateful for Jeremy's curiosity to work on deep learning experiments, and his patience when the experiments weren't conducted well. Over the past years, I have learned a wealth of technical skills from Jeremy. I've also received invaluable advice on scientific communication, how to choose research directions, and where to find hill training opportunities in Chicago. At many points in my Ph.D., issues that seemed insurmountable were solved by simple texts from Jeremy, saying "call me."

I am also grateful for the instruction and advice of my committee member, Risi Kondor. Risi has been a perennial source of advice about scientific machine learning and how to choose research directions. I took two classes from Risi during my Ph.D., which have significantly shaped how I think about Fourier analysis and machine learning on non-Euclidean data. I hope to one day develop deep intuition like Risi's.

I would also like to thank my collaborators: Vasilis Charisopoulos, Daniel Fortunato, Jeremy Hoskins, Eric Jonas, Yuehaw Khoo, Olivia Tsang, and Rebecca Willett. I have had the privilege to work with fantastic and generous collaborators who have shared their

expertise in all parts of the research process. As a member of Becca's research group, I have had the honor to see the ongoing research of a fantastic group of students and postdocs. Everyone in Becca's group has been supportive and encouraging to the maximum. I fondly remember our countless research meetings, conferences, and lunches. I've also been lucky to make friends across campus, especially the students at TTIC, who I met in Nati Srebro's reading group. I will be lucky if I can recreate a similar group of coworkers throughout my career.

My family has been an unwavering source of support over the past five years; they've listened when things were hard and celebrated when things went well. Growing up, my grandfather always asked me the same two questions: "Do you have friends in school?" and "Did you learn anything today?" Years later, the answer is still yes.

And last but not least, I want to thank Katie, the love of my life, whom I met while in grad school. Without Katie, this dissertation would have been neither possible nor worthwhile. Katie, I am so excited to start our life together.

CHAPTER 1

INTRODUCTION

Contemporary machine learning methods, especially those using deep neural networks, have been successfully applied in a range of settings. For example, modern image recognition, speech recognition, and automatic translation systems generally use deep neural network models. These models are trained to automatically solve tasks using a large set of training data, and they are designed to generalize to unseen problem examples. The systems replaced by these data-driven models are often hand-designed for specific settings or are slow to compute an appropriate output. It is therefore unsurprising that problems in scientific computing have become targets for innovation via deep learning research. There are many problems in scientific computing that require hand-designed, highly specific solution methods. In some settings, even the best of these 'classical' solution methods from scientific computing require days or weeks on a supercomputer to compute an output. In other settings, especially in the field of numerical analysis, algorithms are well-developed but are often inflexible and do not scale well to data-rich settings. This thesis contributes machine learning methods for scientific computation problems guided by algorithms in numerical analysis.

The rest of this chapter gives a high-level overview of the thesis by organizing the contributions into three themes:

- Developing Machine Learning Methods with Insights from Numerical Analysis (Section 1.1)
- Developing Numerical PDE Solvers for use in Deep Learning Settings (Section 1.2)
- Building a Computational Toolbox for PDE-Based Inverse Problems in Scientific Imaging (Section 1.3)

In the rest of the document, Chapters 2 to 6 describe the individual projects comprising the

contributions of this thesis. Chapter 7 concludes with an outlook on the field and future work.

1.1 Theme 1: Developing Machine Learning Methods with Insights from Numerical Analysis

The approach of pairing deep neural networks with large datasets has proven successful in multiple fields in scientific computation. Google Deepmind's Alphafold2 model [Jumper et al., 2021] was trained on a large dataset of known protein structures, and can now predict the structure of the protein formed by novel amino acid sequences, a groundbreaking advancement in structural biology. In high-energy particle physics experiments, deep neural networks are able to distinguish between interesting particle collisions and uninteresting 'background' ones, which is critical as thousands of these collisions happen each second in particle colliders [Kasieczka et al., 2019]. These neural networks were trained on millions of simulated collision events. Finally, 'neural operators' have been designed to compute solutions to parametric partial differential equations (PDEs), when training data is available [Kovachki et al., 2023].

Typically, these deep neural networks are designed to solve a specific computational task, and prior physical knowledge about the task is used to inform network design. For example, Alphafold2 is designed to view the structure prediction task as a graph prediction task in 3D space, where individual amino acids in the sequence are labeled adjacent if they are nearby in 3D space. This allows the network to find a representation of the structure that is invariant to rigid transformations of 3D space. Neural networks for high-energy particle physics experiments are designed to use similar symmetries, called Lorentz invariances, to ensure that the prediction on any given collision event does not depend on the measurement frame of reference. The Fourier neural operator [Li et al., 2021b], a popular network in the class of neural operators, takes the insight that many PDEs exhibit a nonlocality of

information, meaning that all parts of the input may affect all parts of the output. To design a network that has similar properties, the Fourier neural operator uses convolutional blocks in the Fourier domain. In all of these examples, knowledge of the system of interest is critical to the network design.

In addition to designing methods that reflect our knowledge of the system of interest, we look to numerical analysis to suggest designs. The overarching hypothesis is that because numerical analysis offers insights about classical ways to efficiently and accurately compute solutions to problems in the physical sciences, these methods can offer suggestions to improve the state of the art in scientific machine learning. Two different chapters in this thesis contribute evidence toward this hypothesis.

In Chapter 2, we consider the design of rotation-invariant machine learning methods. Machine learning methods for molecular property prediction often use a deep neural network constrained to be invariant to rotations of the input molecule, similar to Alphafold2. Using insights from applied harmonic analysis, we design a general-purpose rotation-invariant machine learning method based on random Fourier features [Rahimi and Recht, 2007]. Our method was designed to have an extremely shallow computational graph, which allows it to perform inference on new samples quickly. In the small molecule energy regression setting, we find our method expands the accuracy-latency tradeoff frontier.

In Chapter 2, we also use rotation-invariant random features to introduce non-deep learning baselines to the field of machine learning with rotational invariance constraints. In particular, we investigate whether the success of rotationally-invariant deep learning methods is due to their rotational invariance or to the deep neural network. By introducing a novel rotation-invariant machine learning method, we are able to disentangle these two factors and use our newly-designed method as a baseline against which we can quantify the benefit of deep learning in rotation-invariant machine learning. We find that our rotation-invariant random features are competitive with general-purpose rotation-invariant deep learning methods

on two different tasks, small molecule energy regression and point cloud shape classification.

In Chapter 3, we design a deep neural network and training method for inverse scattering problems in which we recover an inhomogeneous medium from scattered waves. While other neural networks for inverse scattering problems are designed based on an analysis of the physical problem, we take a different approach by designing a network to emulate a particular inversion algorithm from the numerical analysis community. The recursive linearization algorithm [Chen, 1995, Bao and Liu, 2003, Borges et al., 2017] is stable and accurate in challenging problem settings. There is a clear flow of information through this algorithm, which we use to design a series of network blocks and a training procedure. The resulting network sets state-of-the-art results in a highly challenging nonlinear problem setting.

1.2 Theme 2: Developing Numerical PDE Solvers for use in Deep Learning Settings

There are many settings in scientific machine learning in which researchers may wish to build a PDE solver into a deep learning system. Examples include unrolled optimization algorithms applied to PDE-based inverse problems [Li et al., 2024], in which a PDE solver appears in-between trainable neural network blocks, or PDE-constrained design problems, where the PDE solver is used to compute a loss value. This thesis proposes to make progress in scientific machine learning by developing PDE solvers which can be leveraged in a deep learning setting. We identify the class of fast direct solvers [Martinsson, 2019] as a class of algorithms from numerical analysis which offer promise in this setting. Fast direct solvers are designed for high accuracy and low computational burden making them good candidates for high-throughput use when applied to large datasets. Because they directly compute solution operators, as opposed to iteratively solve a system of equations, these solvers have simple computational graphs, which allows automatic differentiation software to compute gradients without backpropagating through the steps of an iterative algorithm. Furthermore, the

precomputed solution operators built by the direct solvers may be used to compute gradients via adjoint methods. However, these algorithms must be modified from their original form to efficiently interface with machine learning algorithms.

We design GPU-accelerated versions of the Hierarchical Poincaré–Steklov (HPS) method for solving linear elliptic PDEs. Chapter 4 details the required modifications of the original HPS algorithms to scale efficiently on general-purpose GPUs. In addition to fast execution on a GPU, we design these computational methods to be compatible with automatic differentiation. This makes it possible, for instance, to rapidly compute and differentiate through a loss function that depends on the solution of a PDE, or to train a neural network architecture that uses a PDE solver in between trainable neural network blocks. We provide an open-source software package implementing these methods and describe the software package in Chapter 5.

In Chapter 6, we develop regularized optimization algorithms that use the software from Chapter 5 to solve PDE-based inverse problems. While we do not use learned regularization components in our experiments, these optimization algorithms are designed to allow for the use of learned components in a variety of problem settings. We expect these design choices will allow for the rapid development of hybrid classical-learned methods for PDE-based inverse problems.

1.3 Theme 3: Building a Computational Toolbox for PDE-Based Inverse Problems in Scientific Imaging

Chapters 3 and 6 consider inverse imaging problems defined by a PDE forward model. Many imaging systems can be modeled by linear elliptic PDEs. In this setting, the target of the imaging method often appears as a coefficient field in the linear partial differential operator, and the measurements are the solution of the PDE, evaluated at a set of receiver points. In general, the mapping from target to measurement is nonlinear, even though the PDE is

linear. This often leads to ill-posed, poorly-conditioned, and nonconvex inverse problems.

The field of inverse problems for imaging is well developed, particularly for linear forward models. There are different methods appropriate for different problem settings. In Figure 1.1, we provide a landscape of approaches for inverse problems along two axes of complexity. A more complete description is available in Ongie et al. [2020]. The first axis measures how much is known about the type of data one is trying to reconstruct. Possibly, nothing is known about the reconstruction targets. In other settings, one may have some prior knowledge about the distribution of targets, which motivates the use of a certain regularization function. In some settings, ground-truth examples of targets from this distribution are available to train a machine learning model. Similarly, one can imagine a range of knowledge about the forward model generating the measurements, and a range difficulty computing the forward model. Some forward models, such as explicit linear forward models, are easy to represent and compute. Other forward models are difficult to compute, as they may be too large to fit into a computer's memory or only defined implicitly.

One theme of this thesis is the development of computational methods to address PDE-based inverse problems in a variety of settings appearing in Figure 1.1. For instance, in Chapter 3, we assume we have access to a set of training pairs of matched measurements and targets, which puts us on the far right side of the horizontal axis. In Chapter 3, we build a deep neural network with an architecture and training method with knowledge of the inverse scattering problem; we use a fully-learned approach with a specially-designed network, which corresponds to the grey box in Figure 1.1. In Chapter 6, we solve PDE-based inverse problems with regularized optimization approaches, a setting which corresponds to the orange "Optimization with regularization" box. In Chapter 7, we describe how future work can build upon the contributions of this thesis in the pink "Unrolled optimization" setting.

For PDE-based inverse problems, the vertical axis of Figure 1.1, our ability to compute

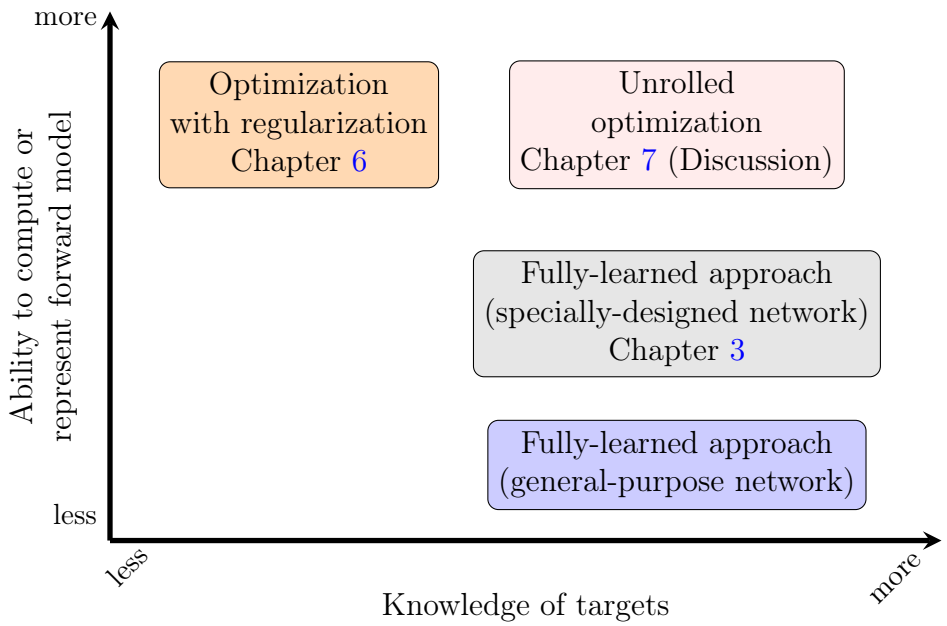


Figure 1.1: Landscape of machine learning methods for inverse imaging problems. This thesis makes contributions to problems in multiple parts of this landscape, focusing on cases when the forward model is defined by an elliptic PDE.

the forward model, is particularly important. Even when the forward model is completely known, one must account for the computational complexity of evaluating the forward model during an iterative algorithm. Because analytical solutions of the coefficient-to-solution map are in general unavailable, the PDE forward models considered in this theme only provide an *implicit* description of the map and require a large amount of computational effort to resolve. The contributions in Chapters 4 and 5 are designed to greatly reduce the complexity along this axis for a broad class of problems, allowing researchers to focus on other parts of the problem, such as modeling knowledge of the targets.

Figure 1.1 omits an important notion of complexity in inverse problems, the conditioning of the forward model. Well-conditioned forward models satisfy stability estimates which guarantee that two different imaging targets will generate two different sets measurements that are well separated in the space of measurements. A strong stability estimate generally means that the forward model provides enough information to stably and accurately re-

construct the image. Forward models which lack such guarantees are called ill-conditioned; these ill-conditioned forward models map distinct images to the very similar measurements, meaning there is not enough information provided by the forward model to distinguish between these images. The PDE-based inverse problems considered in this thesis are often ill-conditioned. The qualitative effects of poor conditioning are often problem-specific and require problem-specific solutions which reflect knowledge of the forward model or knowledge of the imaging targets. This thesis contributes computational methods for PDE-based inverse problems which leverage existing problem-specific solutions in new settings, such as homotopy through frequency for inverse scattering problems in a deep learning setting (Chapter 3) and total variation regularization for piecewise-constant imaging targets in an optimization setting using fast direct solvers (Chapter 6).

1.4 Summary

The themes presented in this chapter are all specific instances of hypothesis that algorithmic insights from numerical analysis can improve the state of the art in applied machine learning for problems in the physical sciences. Early parts of this thesis develop individual machine learning methods with insights from algorithms in numerical analysis; these insights allow the proposed machine learning methods greater accuracy or faster inference, depending on the application. Later parts of this thesis advance the overarching hypothesis by identifying promising algorithms in numerical analysis and developing them into tools applicable in deep learning settings. With exceptions in Chapters 2 and 4, this thesis focuses on applications in PDE-based inverse imaging problems. However, we believe the contributions of this thesis will lead to improvements in other application domains, and we discuss these potential applications in Chapter 7.

CHAPTER 2

ROTATION-INVARIANT RANDOM FEATURES PROVIDE A STRONG BASELINE FOR MACHINE LEARNING ON POINT CLOUDS

2.1 Introduction

Many common prediction tasks where the inputs are three-dimensional physical objects are known to be rotation-invariant; the ground-truth label does not change when the object is rotated. Building rotation invariance into machine learning models is an important inductive bias for such problems. The common intuition is that restricting the learning process to rotation-invariant models will remove any possibility of poor generalization performance due to rotations of test samples and may improve sample efficiency by reducing the effective complexity of learned models. These ideas have inspired a line of research begun by Kondor et al. [2018], Cohen et al. [2018a], Esteves et al. [2018] into building general-purpose deep neural network architectures that are invariant to rotations of their input. However, in these studies, it is not clear whether the reported high accuracies are due to the expressive power of the neural networks or primarily attributable to rotation invariance. We introduce a rotation-invariant random feature model which helps us explore the impact of rotation invariance alone, outside of the neural network framework. Our method is general-purpose and does not require expert knowledge for feature or architecture design. For certain prediction tasks, our proposed method can be computed with very low prediction latency with only a small reduction in accuracy compared to neural network methods, making it a viable method in a range of applications.

In this chapter, we consider prediction problems that are *rotation-invariant*, that is, the ground-truth response $f^*(x) = y$ does not change when an arbitrary rotation is applied to the input x. We represent the input data x as a 3D point cloud, an unordered set of points in

$\mathbb{R}^3$ possibly with accompanying labels. Common examples of rotation-invariant prediction problems on 3D point clouds include molecular property prediction, where the point cloud consists of the positions of a molecule's constituent atoms, and 3D shape classification, where the points are sampled from the surface of an object. Section 4.2 provides a detailed overview of methods used for rotation-invariant prediction, their underlying insights, and their various advantages and disadvantages.

We propose a new method for learning rotation-invariant functions of point cloud data, which we call rotation-invariant random features. We extend the random features method of Rahimi and Recht [2007] by deriving a version that is invariant to three-dimensional rotations and showing that it is fast to evaluate on point cloud data. The rotation invariance property of our features is clear from their definition, and computing the features only requires two simple results from the representation theory of $SO(3)$. Despite its simplicity, our method achieves performance near state-of-the-art on small-molecule energy regression and 3D shape classification. The strong performance on both tasks gives evidence that our method is general-purpose. An overview of our method can be found in Figure 2.1.

Our experiments in Section 2.5.1 show that our model is an order of magnitude faster than state-of-the-art kernel methods when predicting on new samples for energy regression tasks. Thus we conclude our model is a promising candidate for tasks which require real-time rotation-invariant predictions. Minimizing prediction latency is a growing concern for many machine learning methods; there are multiple applications requiring low-latency predictions on point clouds. Using machine learning predictions for real-time system control places hard requirements on prediction latencies. For example, trigger algorithms in the ATLAS experiment at the CERN Large Hadron Collider require prediction latencies ranging from $2\mu s$ to $40ms$ at different stages of the event filtering hierarchy [Collaboration, 2008]. The input to these trigger algorithms is a particle jet, which can be interpreted as a point cloud in four-dimensional spacetime. The majority of data-driven trigger algorithms either impose

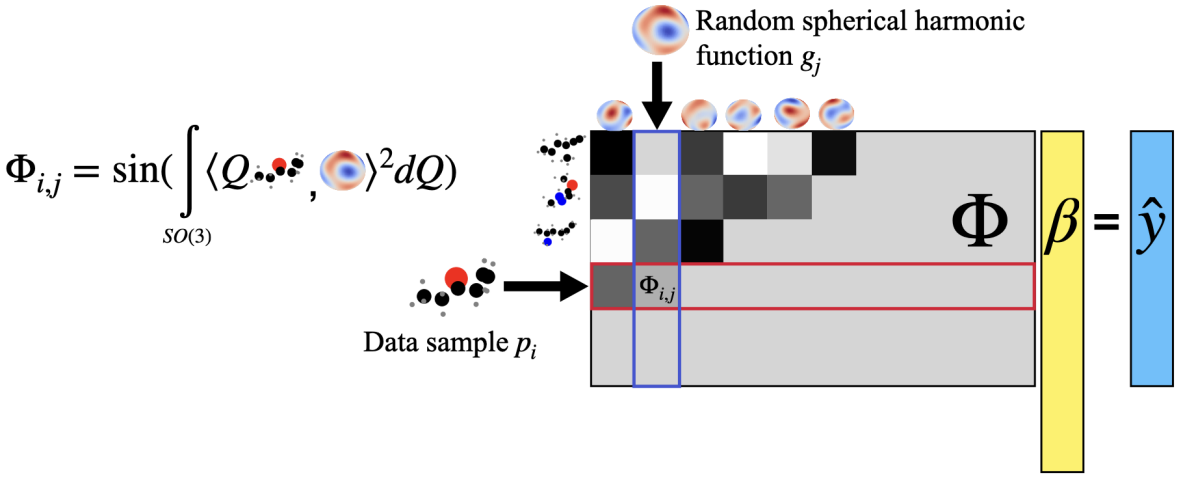


Figure 2.1: An overview of the rotation-invariant random features method. First we construct a feature matrix Φ, defined in Equation (2.7). The rows of Φ correspond to p_i, different samples in our training set, and the columns of Φ correspond to g_j, different random sums of spherical harmonic functions, which are defined in Equation (2.10). The entries of the feature matrix are rotation-invariant random features. To compute the random features, we analytically evaluate the integral over all possible rotations of the data sample p_i. We describe our method for evaluating the random features in Section 2.4.1. After constructing the feature matrix, we fit a set of linear weights β and make predictions by evaluating the linear model $\Phi\beta = \hat{y}$.

rotational invariance in phase space [Komiske et al., 2018, Thaler and Van Tilburg, 2011] or enforce invariance to the Lorentz group, which includes three-dimensional rotations and relativistic boosts [Bogatskiy et al., 2022, Gong et al., 2022]. These definitions of invariance are compatible with our framework, with slight adjustments. Another use-case for low-latency models is as replacements for force fields in molecular dynamics simulations [Gilmer et al., 2017]. These calculations make serial subroutine calls to force field models, meaning latency improvements of these force field models are necessary to speed up the outer simulations. Other rotation-invariant machine learning models either rely on deep computational graphs, which provide a fundamental limit to speedup via parallelization, or they rely on kernel methods [Christensen et al., 2020] which have prediction-time complexity linear in the number of training samples.

2.1.1 Contributions

- We derive a rotation-invariant version of the standard random features approach presented in Rahimi and Recht [2007]. By using simple ideas from the representation theory of $SO(3)$, our rotation-invariant random feature method is easy to describe and implement, and requires minimal expert knowledge in its design (Section 3.4).

- We show with experiments that our rotation-invariant random feature method outperforms or matches the performance of general-purpose equivariant architectures optimized for small-molecule energy regression tasks. In particular, we outperform or match the performance of Spherical CNNs [Cohen et al., 2018a] on the QM7 dataset and Cormorant [Anderson et al., 2019] on the QM9 dataset. We compare our test errors with state-of-the-art methods on the QM9 dataset to quantify the benefit of designing task-specific rotation invariant architectures (Section 2.5.1).

- We show that our method makes predictions an order of magnitude faster than kernel

methods at test time (Section 2.5.1).

- We show the rotation-invariant random feature method is general-purpose by achieving low test errors using the same method on a different task, shape classification on the ModelNet40 benchmark dataset (Section 2.5.2).

- We make our code publicly available at `https://github.com/meliao/rotation-invariant-random-features`.

2.2 Related Work

In this section, we describe two major types of rotation-invariant prediction methods. The first class of methods extracts rotation-invariant features using expert chemistry knowledge. The second class of methods uses deep neural networks to build general-purpose invariant architectures. Finally, we briefly survey the random features work that inspire our method.

Descriptors of Atomic Environments In the computational chemistry community, significant effort has been invested in using physical knowledge to generate "descriptors of atomic environments," or rotation-invariant feature embeddings of molecular configurations. One such feature embedding is the Coulomb matrix, used by Montavon et al. [2012]. In Rupp et al. [2012], the sorted eigenvalues of the Coulomb matrix are used as input features. The sorted eigenvalues are invariant to rotations of the molecule and give a more compact representation than the full Coulomb matrix. Noticing that the Coulomb matrix can relatively faithfully represent a molecule with only pairwise atomic interactions has been an important idea in this community. Many methods have been developed to generate descriptions of atomic environments by decomposing the representation into contributions from all atomic pairs, and other higher-order approximations consider triplets and quadruplets of atoms and

beyond [Christensen et al., 2020, Bartók et al., 2013, Kovács et al., 2021, Drautz, 2019].[1] The FCHL19 [Christensen et al., 2020] method creates a featurization that relies on chemical knowledge and is optimized for learning the energy of small organic molecules. The FCHL19 feature embedding depends on 2-body and 3-body terms. There are multiple different approaches, such as Smooth Overlap of Atomic Potentials (SOAP) [Bartók et al., 2013] and Atomic Cluster Expansion (ACE) [Drautz, 2019, Kovács et al., 2021] which first compute a rotation-invariant basis expansion of a chemical system, and then learn models in this basis expansion. In Appendix A.2, we show that our proposed method can be interpreted as using a highly-simplified set of ACE basis functions to learn a randomized nonlinear model.

Deep learning is an effective tool in computational chemistry as well. Task-specific deep neural network architectures for learning molecular properties of small organic molecules outperform other approaches on common energy learning benchmarks. Multiple methods [Schütt et al., 2018, Unke and Meuwly, 2019] design neural networks to take atomic charges and interatomic distances as input. These methods use highly-optimized neural network architectures, including specially-designed atomic interaction blocks and self-attention layers. OrbNet [Qiao et al., 2020], another neural network method, takes the output of a low-cost density functional theory calculation as rotation-invariant input features and uses a message-passing graph neural network architecture.

General-Purpose Invariant Architectures While many general-purpose invariant architectures operate directly on point clouds, the first examples of such architectures, Spherical CNNs [Cohen et al., 2018a, Esteves et al., 2018, Kondor et al., 2018] operate on "spherical images", or functions defined on the unit sphere. These networks define spherical convolutions between a spherical image and a filter. These convolutions are performed in Fourier

1. Many general-purpose invariant methods follow this design as well. Cormorant [Anderson et al., 2019] has internal nodes which correspond to all pairs of atoms in the molecule, and Spherical CNNs [Cohen et al., 2018a] render spherical images from molecular point clouds using ideas from pairwise electrostatic (Coulombic) repulsion.

space, and the input spherical image is transformed using a fast Fourier transform on the unit sphere. Much like a traditional convolutional neural network, these networks are comprised of multiple convolutional layers in series with nonlinearities and pooling layers in between. To accommodate multiple different types of 3D data, including 3D point clouds, these methods take a preprocessing step to render the information of a 3D object into a spherical image.

Other methods operate on point clouds directly. SPHNet [Poulenard et al., 2019] computes the spherical harmonic power spectrum of the input point cloud, which is a set of rotation-invariant features, and passes this through a series of point convolutional layers. Cormorant [Anderson et al., 2019] and Tensor Field Networks [Thomas et al., 2018] both design neural networks that at the first layer have activations corresponding to each point in the point cloud. All activations of these networks are so-called "spherical tensors," which means they are equivariant to rotations of the input point cloud. To combine the spherical tensors at subsequent layers, Clebsch-Gordan products are used. Many other deep neural network architectures have been designed to operate on point clouds, but the majority have not been designed to respect rotational invariance. A survey of this literature can be found in Guo et al. [2021].

There are many ways to categorize the broad field of group-invariant deep neural network architectures, and we have chosen to form categories based on the input data type. Another helpful categorization is the distinction between *regular* group-CNNs and *steerable* group-CNNs introduced by Cohen et al. [2018b]. In this distinction, *regular* group-CNNs form intermediate representations that are scalar functions of the sphere or the rotation group, and examples of this category are Cohen et al. [2018a], Kondor et al. [2018], Poulenard et al. [2019]. *Steerable* group-CNNs form intermediate representations that are comprised of "spherical tensors", and examples of this category are Anderson et al. [2019], Weiler et al. [2018], Thomas et al. [2018].

More recently, some works have investigated methods that endow non-invariant point cloud neural network architectures with a group-invariance property, either through a modification of the architecture or a data preprocessing step. Xiao et al. [2020] suggests using a preprocessing step that aligns all point clouds in a canonical rotation-invariant alignment; Puny et al. [2022] expands this idea to general transformation groups. After the preprocessing step, any point cloud neural network architecture can be used to make rotation-invariant predictions of the input. Vector Neurons [Deng et al., 2021] endows neural network architectures with rotation equivariance by using three-dimensional vector activations for each neuron in the network and cleverly modifying the equivariant nonlinearities and pooling. These methods have been successfully applied to dense point clouds that arise in computer vision problems such as shape classification and segmentation. Molecular point clouds are qualitatively different; they are quite sparse, and the Euclidean nearest neighbor graph does not always match the graph defined by molecular bond structure. Because of these differences, the alignment methods and network architectures designed for solving shape classification and segmentation tasks are not *a priori* good models for learning functions of molecular point clouds. To the best of our knowledge, they have not been applied successfully in molecular property prediction tasks. Finally, Villar et al. [2021] show that any rotation-invariant function of a point cloud can be expressed as a function of the inner products between individual points; this insight reduces the problem of designing rotation-invariant models to one of designing permutation-invariant architectures.

Random Features Random Fourier features [Rahimi and Recht, 2007] are an efficient, randomized method of approximating common kernels. For kernel methods, prediction time scales linearly with the size of the dataset. Random Fourier feature methods require time linear in d, an approximation parameter. The approximation works by drawing a random vector ξ and defining a low-dimensional feature embedding $\varphi(x;\xi)$, called a random Fourier feature. The random Fourier features approximate a shift-invariant kernel $k(x, x') = k(x-x')$

in the sense that the inner product of two random Fourier feature evaluations is an unbiased estimate of the kernel:

$$\mathbb{E}_{\xi}\left[\langle\varphi(x;\xi),\varphi(x';\xi)\rangle\right] = k(x,x')$$

This approximation holds when the random vectors ξ are drawn from the Fourier transform of the kernel k and the random Fourier features have the following functional form:

$$\varphi(x;\xi) = [\cos(\langle x,\xi\rangle),\ \ \sin(\langle x,\xi\rangle)]^{\top}$$

To reduce the variance of this approximation, one can draw d such random vectors $\{\xi_1,...,\xi_d\}$ and concatenate the resulting random features. As a practical method, Rahimi and Recht [2007] proposes building a feature matrix $\Phi \in \mathbb{R}^{n\times d}$ by drawing d different random Fourier features $\varphi(\,\cdot\,;\xi_j)$ and evaluating the random Fourier features at each of the n samples x_i in the training dataset:

$$\Phi_{i,j} = \varphi(x_i;\xi_j)$$

Once the feature matrix is formed, a linear model is trained to fit a response vector y using regularized linear regression, such as ridge regression:

$$\underset{\beta}{\operatorname{argmin}}\ \ \|\Phi\beta - y\|_2^2 + \lambda\|\beta\|_2^2 \tag{2.1}$$

Experiments in Rahimi and Recht [2007] show this is an effective method for fitting data, even when $d \ll n$. When $d \ll n$, the model can be evaluated much faster than the original kernel.

Follow-up work, including Rahimi and Recht [2008], suggested that interpreting random feature methods as kernel approximators is not necessary. This work uses random nonlinear features with varying functional forms, including random decision stumps and randomly-initialized sigmoid neurons $\sigma(\langle\xi,x\rangle)$. These random nonlinear features form effective models

for diverse types of data. In Section 3.4, we introduce random features similar to those in Rahimi and Recht [2007]. Our method does not approximate any explicit kernel, and it is designed to be invariant to any rotation of the input data, so we call our method rotation-invariant random features.

Finally, the notion of group-invariant random feature models also appears in Mei et al. [2021] as a technical tool to understand the sample complexity benefit of enforcing group invariances in overparameterized models. In this work, the invariances considered are transformations on one-dimensional signals that arise from cyclic and translation groups.

2.3 Rotational Invariance and Spherical Harmonics

In this section, we will introduce rotational invariance and the spherical harmonics. In our method, we use the definitions presented in this section and only two simple facts about the spherical harmonics and rotations, which we list at the end of this section. We say a function f mapping a point cloud p to an element of the vector space $\mathcal{Y}$ is rotation-equivariant if there exists some group action on the vector space $\mathcal{Y}$ such that

$$f(Q \circ p) = Q \circ f(p) \quad \forall Q \in SO(3)$$

Intuitively, this means that when an input to f is rotated, f preserves the group structure and the output changes predictably. We say that a function f is rotation-invariant if the group action on the output vector space $\mathcal{Y}$ is the identity:

$$f(p) = f(Q \circ p) = f(\{Qx_1, Qx_2, \ldots, Qx_N\}) \quad \forall Q \in SO(3) \tag{2.2}$$

Finally, our goal is to learn a function over point clouds that does not change when any rotation is applied to the point cloud. Formally, given some distribution $\mathcal{D}$ generating pairs

of data (p, y) our learning goal is to find

$$f^* = \underset{f}{\operatorname{argmin}} \ \mathbb{E}_{(p,y)\sim\mathcal{D}} \left[l(f(p), y) \right]$$

where l is a classification or regression loss depending on the task and the minimization is over all functions that satisfy a rotational invariance constraint Equation (2.2).

2.3.1 Spherical Harmonics

When decomposing a periodic function $f : [0, 2\pi] \to \mathbb{R}$, a natural choice of basis functions for the decomposition is the complex exponentials e^{imx}. In this basis, $f(x)$ may be expressed as $f(x) = \sum_m a_m e^{imx}$ where $a_m = \langle f(x), e^{imx} \rangle$. When decomposing a function on the unit sphere S^2, a similar set of basis functions emerges, and they are called the spherical harmonics. The complex exponentials are indexed by a single parameter m, and because the spherical harmonics span a more complicated space of functions, they require two indices, ℓ and m. By convention, the indices are restricted to $\ell \geq 0$ and $-\ell \leq m \leq \ell$. A single spherical harmonic function $Y_m^{(\ell)}$ maps from the unit sphere S^2 to the complex plane $\mathbb{C}$. For a function $f(x) : S^2 \to \mathbb{R}$ we can compute an expansion in the spherical harmonic basis:

$$f(x) = \sum_{m,\ell} a_m^{(\ell)} Y_m^{(\ell)}(x)$$
$$a_m^{(\ell)} = \langle f(x), Y_m^{(\ell)}(x) \rangle$$

In designing our method, we use two simple facts about spherical harmonics:

- If one has evaluated the spherical harmonics of some original point $x \in S^2$ and wants to evaluate those spherical harmonics at a new rotated point Qx for some rotation matrix Q, the rotated evaluation is a linear combination of the un-rotated spherical

harmonics at the same index ℓ:

$$Y_m^{(\ell)}(Qx) = \sum_{m'=-\ell}^{\ell} Y_{m'}^{(\ell)}(x) D^{(\ell)}(Q)_{m,m'} \tag{2.3}$$

Here $D^{\ell}(Q)$ is a Wigner-D matrix; it is a square matrix of size $(2\ell+1 \times 2\ell+1)$.

- Evaluating the inner product between two elements of the Wigner D-matrices is simple. In particular, when dQ is the uniform measure over $SO(3)$, we have the following expression:

$$\int_{SO(3)} D^{(\ell_1)}(Q)_{m_1,k_1} D^{(\ell_2)}(Q)_{m_2,k_2} dQ = (-1)^{m_1-k_1} \frac{8\pi^2}{2\ell+1} \delta_{-m_1,m_2} \delta_{-k_1,k_2} \delta_{\ell_1,\ell_2} \tag{2.4}$$

This is a corollary of Schur's lemma from group representation theory, and we discuss this fact in Appendix A.3.

2.4 Rotation-Invariant Random Features

In our setting, an individual data sample is a point cloud $p_i = \{x_{i,1}, \ldots, x_{i,N_i}\}$ with points $x_{i,j} \in \mathbb{R}^3$. We model the data as a function: $p_i(x) = \sum_{j=1}^{N_i} \delta(x - x_{i,j})$ where $\delta(\cdot)$ is a delta function centered at 0. We can then use a functional version of the random feature method where our data p_i is a function, g is a random function drawn from some pre-specified distribution, and $\langle \cdot, \cdot \rangle$ is the L^2 inner product:

$$\tilde{\varphi}(p_i; g) = \sin\left(\langle p_i, g \rangle\right) \tag{2.5}$$

We want the random feature to remain unchanged after rotating the point cloud, but Equation (2.5) does not satisfy this. For general functions g, $\langle Q \circ p_i, g \rangle \neq \langle p_i, g \rangle$. We achieve

rotational invariance by defining the following rotation-invariant random feature:

$$\varphi(p_i; g) = \sin\left(\int_{SO(3)} \langle Q \circ p_i, g\rangle^2 dQ\right) \tag{2.6}$$

We "symmetrize" the inner product by integrating over all possible orientations of the point cloud p_i, eliminating any dependence of $\varphi(\cdot; g_j)$ on the data's initial orientation. Because $\varphi(\cdot; g_j)$ does not depend on the initial orientation of the data, it will not change if this initial orientation changes, i.e., if p_i is rotated by Q. To build a regression model, we construct a feature matrix

$$\Phi_{i,j} = \varphi(p_i; g_j) \tag{2.7}$$

and fit a linear model to this feature matrix using ridge regression, as in Equation (2.1).

2.4.1 Evaluating the Random Features

In this section, we describe how to efficiently evaluate the integral in Equation (2.6). $SO(3)$ is a three-dimensional manifold and the integral has a nonlinear dependence on the data, so methods of evaluating this integral are not immediately clear. However, representation theory of $SO(3)$ renders this integral analytically tractable and efficiently computable. The main ideas used to evaluate this integral are exploiting the linearity of inner products and integration, and choosing a particular distribution of random functions g. First, we observe that because the rotated data function $Q \circ p_i$ is the sum of a few delta functions, we can expand the inner product as a sum of evaluations of the random function g:

$$\langle Q \circ p_i, g\rangle = \sum_{j=1}^{N_i} g(Qx_{i,j}) \tag{2.8}$$

Expanding the inner product and the quadratic that appear in Equation (2.6) and using the linearity of the integral, we are left with a sum of simpler integrals:

$$\int_{SO(3)} \langle Q \circ p_i, g \rangle^2 dQ = \sum_{j_1, j_2 = 1}^{N_i} \int_{SO(3)} g(Q x_{i,j_1}) g(Q x_{i,j_2}) dQ \tag{2.9}$$

Next, we choose a particular distribution for g which allows for easy integration over $SO(3)$. We choose to decompose g as a sum of randomly-weighted spherical harmonics with maximum order L and K fixed radial functions:

$$g(x) = \sum_{k=1}^{K} \sum_{\ell=0}^{L} \sum_{m=-\ell}^{\ell} w_{m,k}^{(\ell)} Y_m^{(\ell)}(\hat{x}) R_k(\|x\|) \tag{2.10}$$

where $w_{m,k}^{(\ell)}$ are random weights drawn from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$, $\hat{x} = \frac{x}{\|x\|}$, and $R_k : \mathbb{R}_+ \to \mathbb{R}$ is a radial function. Two examples of such radial functions are shown in Figure 2.2. This definition of random functions gives us a few hyperparameters; L, the maximum frequency of the spherical harmonics; σ, the standard deviation of the randomly-drawn weights; and the choice of radial functions. In Appendix A.5, we show that the performance of our method is most sensitive to σ.

By repeated application of the linearity of the integral and the two facts about the spherical harmonics mentioned in Section 2.3.1, we are able to write down a closed-form expression for the integral on the left-hand side of Equation (2.9). We include these algebraic details in Appendix A.1. Computing this integral has relatively low complexity; it requires evaluating a table of spherical harmonics for each point $x_{i,j}$ up to maximum order L, and then performing a particular tensor contraction between the array of spherical harmonic evaluations and the array of random weights. This tensor contraction has complexity $O(N^2 L^3 K^2)$.

2.5 Experiments

We conduct multiple experiments comparing our method with other landmark rotation-invariant machine learning methods. We find that for predicting the atomization energy of small molecules, our method outperforms neural networks in the small dataset setting, and we have competitive test errors in the large dataset setting. Our method is an order of magnitude faster than competing kernel methods at test time. We also find that our method performs competitively on a completely different task, 3D shape classification. We perform these experiments to show that our method can be used as a fast, simple, and flexible baseline for rotation-invariant prediction problems on 3D point cloud data.

2.5.1 Small-Molecule Energy Regression

A common target for machine learning in chemistry is the prediction of a potential energy surface, which maps from 3D atomic configurations to the atomization energy of a molecule. Large standardized datasets such as QM7 [Blum and Reymond, 2009, Rupp et al., 2012] and QM9 [Ruddigkeit et al., 2012, Ramakrishnan et al., 2014] offer a convenient way to test the performance of these machine learning models across a wide chemical space of small organic molecules. Both datasets contain the 3D atomic coordinates at equilibrium and corresponding internal energies. The 3D coordinates and molecular properties are obtained by costly quantum mechanical calculations, so there may be settings which require high-throughput screening where an approximate but fast machine learning model may provide an advantageous alternative to classical methods. To adapt our method to this task, we make two small changes.

Element-Type Encoding The rotation-invariant random feature method defined above works for general, unlabeled point clouds. However, in the chemistry datasets mentioned above, we are given more information than just the 3D coordinates of particles in the

molecule: the particles are individual atoms, and we know their element type. Incorporating the element type of individual atoms in a machine learning method is crucial for accurate prediction. Different elements interact in quantitatively different ways, as predicted by the laws of gravitation and electrostatic repulsion, and they interact in qualitatively different ways due to their electron configurations.

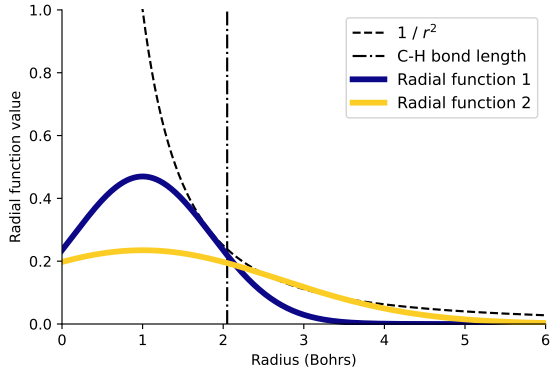
We use an element-type encoding method inspired by FCHL19 [Christensen et al., 2020] and ACE [Kovács et al., 2021]. To construct this element-type encoding, we look at the local view of a molecule created by centering the atomic coordinates at a given atom, separate the atoms into different point clouds for each element type, and compute one random feature per element type. We then repeat this procedure for all elements of a given type and sum their feature vectors.

Expressed mathematically, we are given a sample $p_i = \{x_{i,1}, \ldots, x_{i,N_i}\}$ with charges $\{c_{i,1}, \ldots, c_{i,N_i}\}$. Let $p_i^{(c_j)}$ be the collection of atoms with charge c_j, and let $p_i^{(c_j)} - x_{i,h}$ denote the point cloud of atoms in p_i with charge c_j centered at point $x_{i,h}$. In this notation, $p_i^{(c_j)} - x_{i,h}$ specifies a point cloud, which we can treat as unlabeled because they all have the same charge. As before, a random feature is denoted $\varphi(\cdot; g_j)$. Then for all possible element-type pairs (c_1, c_2), we compute individual entries in our feature matrix

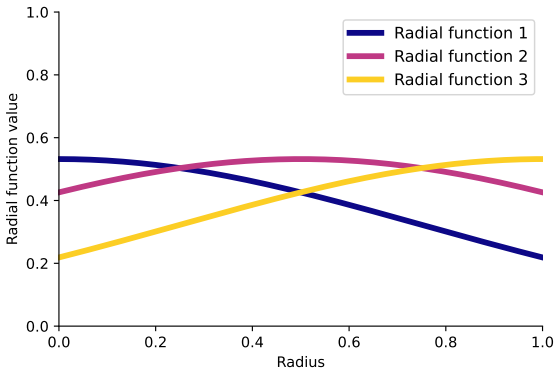
$$\Phi_{i,j'} = \sum_{h:\ c_{i_h}=c_1} \varphi\left(p_i^{(c_2)} - x_{i,h}; g_j\right)$$

The column index j' depends on j, c_1, and c_2.

Radial Functions We choose to parameterize our set of random functions as randomly-weighted spherical harmonics with fixed radial functions. The choice of radial functions appears as a design decision in many rotation-invariant machine learning methods. For example, radial functions appear as explicit design choices in Kovács et al. [2021], Christensen et al. [2020], Poulenard et al. [2019] and implicitly in the design of Cohen et al. [2018a].



(a) Radial functions for small-molecule energy regression



(b) Radial functions for 3D shape classification

Figure 2.2: Our rotation-invariant random feature method requires simple user-defined radial functions. Figure 2.2a shows the radial functions used in our small-molecule energy regression experiments. We include $\frac{1}{r^2}$ and the average Carbon-Hydrogen bond length to give context to the horizontal axis. Figure 2.2b shows the radial functions used in our 3D shape classification experiments. The 3D shapes in the benchmark dataset are normalized to fit inside the unit sphere.

This design choice is often paramount to the empirical success of these methods. FCHL19 [Christensen et al., 2020] optimize their radial functions over multiple hyperparameters and carefully balance multiplicative terms including log-normal radial functions, polynomial decay, and soft cut-offs. The ACE models in Kovács et al. [2021] use a set of orthogonal polynomials defined over a carefully-chosen subset of the real line, a physically-motivated spatial transformation, and use extra ad-hoc radial functions that control the behavior of extremely nearby points. For our radial functions, we use two Gaussians, both centered at 1, with width parameters chosen so the full widths at half maximum are 2 and 4 respectively. Our radial functions are shown in Figure 2.2.

QM7 Atomization Energy Regression

The QM7 dataset contains 7,165 small molecules with element types H, C, N, O, S and a maximum of 7 heavy elements. We compare with two lightweight methods, FCHL19 and

Spherical CNNs, that show learning results on the QM7 dataset. We reproduce their original training methods and compare test errors in Table 2.1. Notably, our rotation-invariant random feature method has average errors half of those of Spherical CNNs while being faster to train. Our best-performing model uses 2,000 random features. With the element-type encoding described above, this corresponds to 50,000 trainable parameters.

In this experiment, we use a training dataset of 5,732 samples and a test set of size 1,433. For our method and FCHL19, we use 90% of the training set to train the models and the remaining 10% as a validation set for hyperparameter optimization. To train the FCHL19 models, we use the validation set to optimize over Gaussian kernel widths and L^2 regularization parameters. In our method, we search over the number of random features and L^2 regularization parameters. We solve our ridge regression problem by taking a singular value decomposition of the random feature matrix, and then constructing a solution for each regularization level.

In addition to our ridge regression models, we also experiment using our random features as an input to deep neural networks. We are unable to find deep neural network models with reasonable prediction latencies that outperformed our ridge regression model (Appendix A.4). We also experimented with a model which combined a rotation-invariant PCA alignment [Xiao et al., 2020, Puny et al., 2022] with three-dimensional random features, but the alignments produced by this method were uninformative, and the model failed to meet basic baselines.

The space of small organic drug-like molecules is extremely large. The GDB-13 dataset [Blum and Reymond, 2009] contains nearly one billion reference molecules, even after filtering for molecule size and synthesis prospects. Most of the machine learning methods approximating potential energy surfaces are considered fast approximate replacements for costly quantum mechanical calculations. To screen the extremely large space of small organic drug-like molecules, these methods require high prediction throughput, which can be

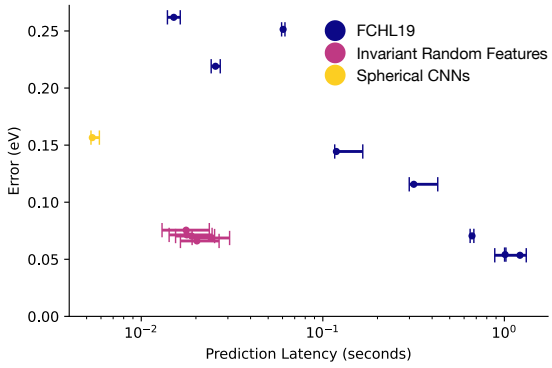
Method	General-Purpose	Mean Absolute Error (eV)	Train Time (s)	Train Device
Spherical CNNs [Cohen et al., 2018a]	✓	0.1565	546.7	single GPU
Random Features (Ours)	✓	0.0660 ± 0.00275	203.4	24 CPU cores
FCHL19 [Christensen et al., 2020]	✗	0.0541	260.7	24 CPU cores

Table 2.1: Test error and training time on the QM7 dataset. We report the mean absolute error on the test set for all methods and the standard error of the mean for our method. We discuss this experiment in Section 2.5.1.

achieved by exploiting parallelism in the models' computational graphs and readily-available multicore CPU architectures. When parallelism inside the model is not available, the set of candidate molecules can be divided and data parallelism can be used to increase screening throughput. However, we consider another setting where experimental samples need to be analyzed in real time to make decisions about ongoing dynamic experiments. In this setting, prediction latency is paramount.

In the low-latency setting, our method can gracefully trade off prediction latency and prediction error by tuning the number of random features used in a given model. Importantly, all training samples are used to train the model. FCHL19 is a kernel method, so the time required to predict a new data point scales linearly with the number of training samples used. Thus, FCHL19 can only improve test latency at the cost of using fewer training samples and incurring higher test errors.

To explore the tradeoff between prediction latency and prediction error, we train random feature models and FCHL19 models of different sizes. We measure their prediction latencies and plot the results in Figure 2.3. We note that at similar error levels, FCHL19 models show prediction latencies that are an order of magnitude slower than ours. Spherical CNNs exhibit low test latency because their method is implemented for a GPU architecture, but their test errors are high, and the neural network approach does not admit obvious ways to



Method	Model Size	Prediction Latency (s)	Train time (s)	Error (eV)
FCHL19	100	0.0151	0.4	0.2620
FCHL19	200	0.0256	0.8	0.2191
FCHL19	400	0.0604	2.3	0.2514
FCHL19	800	0.1188	9.1	0.1444
FCHL19	1600	0.3170	54.0	0.1157
FCHL19	3200	0.6635	258.5	0.0705
FCHL19	5000	1.0096	260.7	0.0541
FCHL19	5732	1.2191	326.1	0.0535
Random Features (Ours)	250	0.0177	61.6	0.0755
Random Features (Ours)	500	0.0178	107.1	0.0714
Random Features (Ours)	750	0.0189	157.8	0.0704
Random Features (Ours)	1000	0.0203	203.4	0.0660
Random Features (Ours)	2000	0.0242	431.4	0.0687
Spherical CNNs	N/A	0.0054	546.7	0.1565

Figure 2.3: Invariant Random Features are faster at prediction time than FCHL19 and more accurate than Spherical CNNs. In this experiment, we compare the prediction latency and prediction error for models of different sizes evaluated on the QM7 dataset. For each method, we measure the latency of predicting each molecule on a held-out test set 5 times. In the figure, we plot the median prediction latencies with error bars spanning the 25^{th} and 75^{th} percentile measurements. The vertical axis of the figure is mean absolute error measured in eV on a held-out test set. The FCHL19 method is evaluated on 24 CPU cores using kernels with sample sizes 100, 200, 400, 800, 1600, 3200, 5000, and 5732. The rotation-invariant random features method is evaluated on 24 CPU cores with model sizes of 250, 500, 750, 1000, and 2000 random features. The Spherical CNNs method is evaluated on a single GPU. We discuss this experiment in Section 2.5.1.

achieve a tradeoff between latency and test error. The prediction latencies of FCHL19 and our random feature models depend on the number of atoms in the molecule, and we show this dependence in Figure A.1.

QM9 Atomization Energy Regression

The QM9 dataset contains 133,885 molecules with up to 9 heavy atoms C, O, N, and F, as well as Hydrogen atoms. This is a larger and more complex dataset than QM7, and because there is more training data, neural network methods perform well on this benchmark. For this dataset, we follow Anderson et al. [2019], Gilmer et al. [2017] by constructing atomization energy as the difference between the internal energy at $0K$ and the thermochemical energy of a molecule's constituent atoms. We train a large-scale random feature model by taking

Method	General-Purpose	Model Type	Mean Absolute Error (eV)	Input Features
Cormorant [Anderson et al., 2019]	✓	General-purpose rotation-invariant architecture	0.022	• Charges • Atomic locations
Random Features (Ours)	✓	Ridge Regression	0.022 ± 4.45e-04	• Charges • Rotation-invariant random features
FCHL19 [Christensen et al., 2020]	✗	Kernel Ridge Regression	0.011	• Charges • Interatomic distances • 3-body angles
SchNet [Schütt et al., 2018]	✗	Neural Network with atomic interaction blocks	0.014	• Charges • Interatomic distances
PhysNet [Unke and Meuwly, 2019]	✗	Neural Network with attention layers	0.008	• Charges • Interatomic distances
OrbNet [Qiao et al., 2020]	✗	Message-Passing GNN	0.005	• Mean-field DFT calculations

Table 2.2: Comparing large-scale models trained on QM9. FCHL19 [Christensen et al., 2020] was trained on 75,000 samples from QM9 and the other methods were trained on 100,000 samples. We report the mean absolute error on the test set for all methods and the standard error of the mean for our method. We discuss this experiment in Section 2.5.1.

100,000 training samples from QM9 and generating 10,000 random features. We select an L^2 regularization parameter *a priori* by observing optimal regularization parameters from smaller scale experiments on a proper subset of the QM9 training data.

We compare the performance of our model with the reported errors of FCHL19 [Christensen et al., 2020] and other neural network methods. All methods considered in this comparison enforce rotational invariance in their predictions. The results of this comparison are shown in Table 2.2. Our rotation-invariant random feature method matches the performance of Cormorant, and it provides a very strong baseline against which we can quantify the effect of expert chemistry knowledge or neural network design. OrbNet [Qiao et al., 2020] uses a graph neural network architecture, and their inputs are carefully chosen from the results of density functional theory calculations in a rotation-invariant basis. PhysNet [Unke and Meuwly, 2019] uses an architecture heavily inspired by that of SCHNet [Schütt et al., 2018], and this iteration in model design resulted in almost a 50% reduction in test error.

Training models on large subsets of the QM9 dataset is difficult. FCHL19 requires 27 hours to construct a complete kernel matrix of size (133,855 $\times$ 133,855) on a compute node with 24 processors. The 27 hours does not include the time required to find the model's linear weights. The neural network methods require long training sequences on GPUs. Both Cormorant and PhysNet report training for 48 hours on a GPU.

When using a large sample size, our method is similarly difficult to train. We are able to construct a matrix of random features of size (100,000 $\times$ 250,000) in 27 minutes on a machine with 24 processors, but the matrix is highly ill-conditioned, and using an iterative method to approximately solve the ridge regression problem requires 76.5 hours. Using an iterative method introduces approximation error which we did not encounter when performing experiments on the other datasets. We attribute some of the performance gap between our method and FCHL19 to this approximation error. Fortunately, solving large, dense,

overdetermined ridge regression problems is an active area of research, and it is quite likely that applying methods from this area of research to our problem would result in a speed-up. We outline some promising approaches in Appendix A.6. We believe the conditioning of our problem is caused by the choice of element-type encoding, and we leave finding a new element-type encoding that produces well-conditioned feature matrices for future work.

2.5.2 Shape Classification

To show that our method is general-purpose, we test our same method on a completely different task, 3D shape classification. In particular, we consider multiclass classification on the ModelNet40 benchmark dataset. The ModelNet40 benchmark dataset Zhirong Wu et al. [2015] is a set of computer-generated 3D models of common shapes, such as mugs, tables, and airplanes. We evaluate the performance of our model in three train/test settings, where the train and test sets either contain rotations about the z axis, or arbitrary rotations in $SO(3)$. There are 9,843 training examples and 2,468 test examples spread across 40 different classes. The shape objects are specified by 3D triangular meshes, which define a (possibly disconnected) object surface. To generate a point cloud from individual objects in this dataset, one must choose a sampling strategy and sample points from the surface of the object. We use the dataset generated by Qi et al. [2017a], which samples 1,024 points on the mesh faces uniformly at random. The point clouds are then centered at the origin and scaled to fit inside the unit sphere.

We solve the multiclass classification problem with multinomial logistic regression. More specifically, we optimize the binary cross entropy loss with L2 regularization to learn a set of linear weights for each of the 40 classes in the dataset. At test time, we evaluate the 40 different linear models and predict by choosing the class with the highest prediction score. We also slightly change the definition of our data function; for a point cloud $p_i = \{x_{i,1}, ..., x_{i,N_i}\}$, we use a normalized data function $p_i(x) = \frac{1}{N_i} \sum_{j=1}^{N_i} \delta(x - x_{i,j})$ to eliminate any dependence

Method	Rotation Invariant	z/z	SO(3)/SO(3)	z/SO(3)
Spherical CNNs [Esteves et al., 2018]	✓	0.889	0.869	0.786
SPHNet [Poulenard et al., 2019]	✓	0.789	0.786	0.779
Vector Neurons [Deng et al., 2021]	✓	0.902	0.895	0.895
Random Features (Ours)	✓	0.693	0.692	0.666
PointNet++ [Qi et al., 2017b]	✗	0.918	0.850	0.284

Table 2.3: Test accuracy on the ModelNet40 shape classification benchmark task. Our method sets a strong baseline to compare against neural network methods, especially in the challenging $z/SO(3)$ train/test regime. We discuss this experiment in Section 2.5.2.

on the number of points sampled from the surface of the shape objects. For this setting, we use radial functions that are three Gaussian bumps with centers at $0, 0.5$, and 1, with width $\sigma = 0.75$.

We compare our method with another rotation-invariant method, SPHNet [Poulenard et al., 2019]. SPHNet is a multilayer point-convolutional neural network that enforces rotational invariance by computing the spherical harmonic power spectrum of the input point cloud. SPHNet's rotation-invariant method also requires a choice of radial functions, and they use two Gaussians with different centers and vary the width of the Gaussians at different layers of their network. The results of our comparison are in Table 2.3. Our method does not achieve near state-of-the-art results, but we conclude it provides a strong baseline on this challenging multiclass problem. We discuss the prediction latency of our method on the ModelNet40 dataset in Appendix A.4.

2.6 Discussion and Future Work

Our method is a simple, flexible, and competitive baseline for rotationally-invariant prediction problems on 3D point clouds. Rotation-invariant random features are simple to explain and implement, and using them in varied prediction tasks requires a minimal amount of

design choices. Our method does not achieve state-of-the-art accuracy on any prediction task, but it provides a competitive baseline that allows us to begin to quantify the effect of neural network models and expert chemistry knowledge in methods that enforce rotational invariance. In particular, we find that for molecular property prediction tasks, the inductive bias of rotation invariance is enough to provide very strong performance with our general-purpose random feature model. For 3D shape classification, we find that the inductive bias of rotation invariance provides a non-trivial baseline, but the gap between our model and neural networks is larger.

Our method shows promise in settings where low prediction latency is desired. In high-throughput experiments, such as the ATLAS experiment at the CERN Large Hadron Collider [Collaboration, 2008], data is generated at such a high velocity that real-time decisions must be made whether to save or discard individual samples. For this type of initial screening task, we imagine our low-latency and flexible prediction method will be an attractive candidate. Another experimental use-case for low-latency models is as replacements for force fields in molecular dynamics simulations [Gilmer et al., 2017]. In these simulations, the energy and forces in a molecular system is repeatedly queried in a serial fashion. Any improvement in the latency of energy or force prediction would translate to a speedup of the overall system. Because our model has an extremely shallow computational graph, we expect it will have much lower latency than deep neural networks once implemented on a GPU.

One can also interpret our work as an initial investigation into the use of random Fourier feature methods for approximating common kernel methods in computational chemistry. Kernel methods are well-studied and highly performant in machine learning for computational chemistry; examples of such methods include Rupp et al. [2012], Montavon et al. [2012], Bartók et al. [2013], Christensen et al. [2020]. It is a natural question to ask whether these methods can benefit from low prediction latency while maintaining high test accuracy when approximated by random features. Our method does not implement an exact approx-

imation to any of the kernel methods above, but our experiments show promise in this area. Our test errors are near those of FCHL19 on the QM7 dataset, and versions of our model using only 10% of the optimal number of parameters have reasonable test errors. However, when training our model on the QM9 dataset, we have seen that the particular element-type encoding we have used creates ill-conditioned feature matrices and makes optimization difficult. To fully realize the potential of random feature methods in accelerating kernel learning for computational chemistry, a new element-type encoding is needed.

Broader Impact Statement

The task of molecular property prediction is important to a wide range of applications, including the development of new pharmaceuticals, materials, and solvents. Some of these applications may be misused.

Acknowledgments

RMW was supported by AFSOR award FA9550-18-1-0166 and NSF DMS-2023109. OJM was supported by an NSF Research Traineeship under grant NSF 2022023.

CHAPTER 3

MULTI-FREQUENCY PROGRESSIVE REFINEMENT FOR LEARNED INVERSE SCATTERING

3.1 Introduction

Wave scattering is an important imaging technology with applications in medical and seismic imaging, sonar and radar detection, and nondestructive testing of materials. In this setting, a known source transmits incident waves through a penetrable medium, and due to an inhomogeneity in the spatial region of interest, the incident waves are scattered. Several receivers measure the scattered wave field at distant locations. We are interested in the inverse wave scattering problem: given a set of scattered wave field measurements, we want to recover the inhomogeneity in the spatial region of interest that produced the measurements. In this chapter, we focus on the inverse wave scattering problem with unknown medium and full-aperture measurements at multiple incident wave frequencies. This problem is characterized by a highly nonlinear forward measurement operator, making the recovery of the scattering potential challenging. We propose a machine learning solution to this problem: given a training set of pairs of scattering potentials and scattered wavefield measurements, we seek to approximate the inversion map with a deep neural network that predicts a scattering potential from scattered wavefield measurements at multiple frequencies. We design a new training method and a new neural network architecture to achieve this goal.

The aforementioned inverse wave scattering problem has been widely studied. While the measurement operator is known to be injective when there are infinitely many sensors positioned in a ring around the scattering potential [Colton and Kress, 2018], computational approaches must always operate in the ill-posed case where finite receivers are present. Thus, past research has focused on optimization approaches to solving the inverse problem. Simple gradient-based optimization approaches to this problem face two major difficulties:

computing a gradient requires solving a partial differential equation (PDE), which can be computationally expensive; additionally, the nonlinearity of the forward model induces a non-convex objective function. Therefore, convergence of local search methods such as gradient descent is not guaranteed without careful initialization.

A classical strategy to alleviate the optimization challenges associated with a nonlinear forward model is to use a linear approximation. This linear approximation allows one to formulate the inverse problem as a linear least squares problem, which is relatively easy to solve. A well-known method inverting this global linear approximation is the so-called *filtered back-projection (FBP) method* [Natterer, 2001]. While this method is relatively easy to implement, it suffers from modeling errors and produces inaccurate reconstructions. To remedy this, many machine learning approaches are aimed at constructing data-driven approximations of the inverse map by designing architectures to imitate FBP [Fan and Ying, 2022, Khoo and Ying, 2019, Li et al., 2022]. At a high level, these works approximate the two key components of FBP – namely, an application of the adjoint of a linearization of the forward operator followed by a filtering step – by suitably chosen neural network blocks. As a result, they suffer from similar drawbacks as the FBP method: in particular, they provide low-quality reconstructions of high-contrast scattering potentials, especially in the presence of unknown inhomogeneous backgrounds or measurement noise. A natural alternative is integrating machine-learning models into other iterative methods, which are computationally demanding relative to FBP but can provide higher quality reconstructions.

A standard approach, which has been successful in the strongly nonlinear scattering regime, is to use data collected at multiple incoming wave frequencies. Recursive linearization methods [Chen, 1995, Bao and Liu, 2003, Borges et al., 2017] use multi-frequency measurements to solve a sequence of sub-problems, starting at the lowest frequency to provide an initial estimate of the scattering potential and refining that estimate at progressively higher frequencies using warm-started local search methods. Algorithms in this family offer two

benefits: first, they alleviate the need for careful initialization since the loss landscape of the lowest frequency sub-problem is typically well-behaved; second, they greatly reduce the number of PDE solves by relying on first-order approximations of the forward model that are relatively inexpensive to invert. However, these methods require measurements at a large number of incident wave frequencies and still involve solving large-scale PDEs and least-squares problems for each frequency. This requires, for example, multiple CPU core-hours to recover a single image, even with a state-of-the-art PDE solver [Borges et al., 2017].

In light of these advances, we propose a new architecture and training method inspired by the recursive linearization algorithm. Our primary approach is based on a residual update architecture and training method that ensures specific network blocks solve specific sub-problems. In addition, we introduce two new methods of extending FBP-inspired neural networks to the multi-frequency setting. We call our networks "Multi-Frequency Inverse Scattering Networks", or "MFISNets". We note that our methods are based on a neural network architecture from Fan and Ying [2022] but could, in principle, use any other neural network architectures designed for the inverse scattering problem in the single-frequency setting.

3.1.1 Contributions & chapter outline

In Section 3.2, we formally define the inverse scattering problem and the machine learning objective. We present standard results about inverse scattering and survey related work in Section 3.3. In Section 3.4, we review the recursive linearization algorithm and introduce our method, MFISNet-Refinement. Finally, we introduce our other two methods, MFISNet-Fused and MFISNet-Parallel, and present a numerical evaluation of our methods in Section 3.5. Our main contributions can be summarized as follows:

1. We introduce "MFISNet-Refinement", short for "Multi-Frequency Inverse Scattering Network with Refinement", a neural network architecture and training method that is

inspired by recursive linearization algorithms [Chen, 1995, Bao and Liu, 2003, Borges et al., 2017]. (Section 3.4)

2. We show that our network achieves lower errors than single-frequency methods [Fan and Ying, 2022] and multi-frequency methods [Li et al., 2022] in the literature as well as the other newly-introduced MFISNets in a high-contrast, noiseless, full-aperture setting. (Section 3.5.3)

3. We demonstrate numerically that our method is robust to moderate measurement noise. (Section 3.5.4)

4. We consider alternative training strategies and find that MFISNet-Refinement is robust to the choice of training method, suggesting the majority of improvement is due to the architecture. (Section 3.5.5)

5. We publicly release our code https://github.com/meliao/mfisnets and dataset https://doi.org/10.5281/zenodo.14514353.

3.2 Problem Setup and Notation

The forward model for our imaging setup is implicitly defined by a PDE problem involving the Helmholtz equation. See Figure 3.1 for a diagram of the geometry of the problem. Let $x \in \mathbb{R}^2$ be the spatial variable. Suppose $u_{\text{in}}(x; s) = e^{ikx \cdot s}$ is an incoming plane wave with direction $s \in \mathbb{S}^1$, wavelength λ, and angular wavenumber $k = 2\pi/\lambda$. We normalize the problem's units so this wave travels at speed $c_0 \equiv 1$ in free space. The incoming wave interacts with a real-valued scattering potential $q(x)$ to produce an additive perturbation, called the scattered wave field $u_{\text{scat}}[q](x; s)$. We define $q(x) = c_0^2/c^2(x) - 1$ where $c(x)$ is the wave speed at x. The total wave field $u[q](x; s) = u_{\text{scat}}[q](x; s) + u_{\text{in}}(x; s)$ solves the following inhomogeneous Helmholtz equation, and the scattered wave field satisfies the Sommerfeld

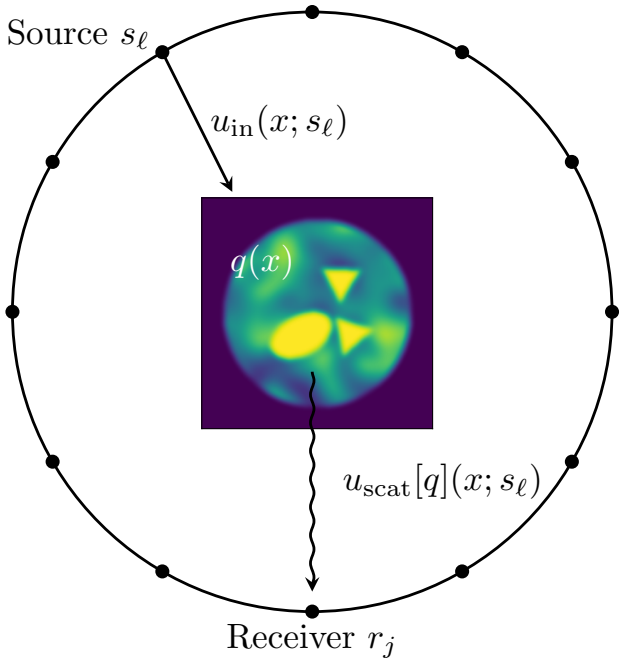


Figure 3.1: Geometry of the inverse scattering problem. An incident plane wave $u_{\text{in}}(x; s_\ell)$ coming from source direction s_ℓ interacts with the scattering potential $q(x)$. The resulting scattered wave field $u_{\text{scat}}[q](x; s_\ell)$ is recorded by a receiver r_j.

radiation boundary condition:

$$\begin{cases} \Delta u[q](x;s) + k^2(1+q(x))u[q](x;s) = 0 & x \in \mathbb{R}^2; \\ \dfrac{\partial u_{\text{scat}}[q](x;s)}{\partial \|x\|_2} - iku_{\text{scat}}[q](x;s) = o(\|x\|_2^{-1/2}), & \text{as } \|x\|_2 \to \infty. \end{cases} \tag{3.1}$$

We assume $q(x)$ is supported on a square domain $\Omega = [-0.5, 0.5]^2$, and we work with $q \in \mathbb{R}^{N_q \times N_q}$, the discretization of $q(x)$ onto a regular grid with (N_q, N_q) grid points. We place the receivers equally spaced around a large ring of radius $R \gg 1$, centered at the origin. We identify individual receivers by their unit-vector directions $r_j \in \mathbb{S}^1$. We compute the solution for the same set of N_r receiver points and N_s incoming source directions, where $N_r = N_s$ and the grid points are equally distributed about the unit circle. This results in a set of (N_r, N_s) observations, $\{u_{\text{scat}}[q](Rr_j; s_\ell)\}_{j,\ell \in [N_r] \times [N_s]}$, which we arrange in a data array $d_k \in \mathbb{C}^{N_r \times N_s}$. We call the mapping from q to d_k the forward model with incoming

wave frequency k:

$$(d_k)_{j,\ell} = \mathcal{F}_k[q]_{j,\ell} \equiv u_{\text{scat}}[q](Rr_j; s_\ell) \tag{3.2}$$

Because we are interested in multi-frequency algorithms, we are interested in observations of the forward model evaluated on the same q but with a set of incoming wave frequencies $[k_1, \ldots, k_{N_k}]$. In particular, our goal is to approximate the following mapping:

$$\left[\mathcal{F}_{k_1}[q], \ldots, \mathcal{F}_{k_{N_k}}[q]\right] \mapsto q. \tag{3.3}$$

Our goal is to train a neural network g_θ with parameters θ to approximate the mapping: $g_\theta(\mathcal{F}_{k_1}[q], \ldots, \mathcal{F}_{k_{N_k}}[q]) \approx q$. Given a distribution $\mathcal{D}$ over scattering potentials q, we draw a training set of n independent samples from $\mathcal{D}$ to generate data to train the neural network. After evaluating the forward model nN_k times, we have a training set

$$\mathcal{D}_n := \left\{(q^{(j)}, \mathcal{F}_{k_1}[q^{(j)}], \ldots, \mathcal{F}_{k_{N_k}}[q^{(j)}])\right\}_{j=1}^{n} \tag{3.4}$$

We evaluate networks in this setting by measuring the relative ℓ_2 error:

$$\text{RelativeL2Error}(g_\theta) = \mathbb{E}_{q\sim\mathcal{D}}\left[\frac{\|g_\theta\left(\mathcal{F}_{k_1}[q], \ldots, \mathcal{F}_{k_{N_k}}[q]\right) - q\|_2}{\|q\|_2}\right] \tag{3.5}$$

In practice, we approximate the expected relative ℓ_2 error in (3.5) by an empirical mean over a held-out test set of 1,000 samples drawn independently from $\mathcal{D}$.

3.3 Background and Related Work

3.3.1 Background

In this section, we review standard results about $\mathcal{F}_k$ relevant to our study. In particular, we focus on a linear approximation of $\mathcal{F}_k$ that gives insights into the inverse scattering problem.

The first result is that $\mathcal{F}_k$ becomes more nonlinear as the magnitude of the scatterer $\|q\|_2$ or the wavenumber k increases. Indeed, the solution to (3.1) can be equivalently defined as the solution to the Lippmann-Schwinger integral equation:

$$u_{\text{scat}}[q](x;s) = k^2 \int_\Omega G_k(\|x-x'\|_2) q(x') (u_{\text{in}}(x';s) + u_{\text{scat}}[q](x';s)) \mathrm{d}x' \tag{3.6}$$

where G_k is the Green's function for the homogeneous Helmholtz operator. This recursive equation provides a nonlinear map from $q(x)$ to $u_{\text{scat}}[q](\cdot;s)$ and therefore $\mathcal{F}_k[q]$.

One way to view this nonlinearity is to interpret the Lippmann-Schwinger equation as a power series in $q(x)$ by iteratively substituting the value of $u_{\text{scat}}[q](x)$ into its appearance on the right-hand side of (3.6). For example, performing this substitution once yields a linear and a quadratic term in $q(x)$ that are independent of $u_{\text{scat}}[q]$, as well as a "remainder" term involving the unknown $u_{\text{scat}}[q]$ that accounts for higher-order terms:

$$\begin{aligned} u_{\text{scat}}[q](x;s) = {} & k^2 \int_\Omega G_k(\|x-x'\|_2) q(x') u_{\text{in}}(x';s) \mathrm{d}x' \\ & + k^4 \int_\Omega G_k(\|x-x'\|_2) q(x') \int_\Omega G_k(\|x'-x''\|_2) q(x'') u_{\text{in}}(x'';s) \mathrm{d}x'' \mathrm{d}x' \\ & + \text{Higher-order terms in } k \text{ and } q(x). \end{aligned} \tag{3.7}$$

This power series does not converge for general $q(x)$, but it helps illustrate which parts of the problem drive the nonlinearity of the operator $\mathcal{F}_k$: as $\|q\|_2$ or k grow, the size of these nonlinear terms will also grow, and as a result $\mathcal{F}_k$ becomes increasingly nonlinear.

The next result is that, under a linear approximation, the far-field measurements are diffraction-limited and can only capture frequency components of q up to $2k$. Equivalently, the measurements depend on q to a spatial resolution of $\lambda/2$. We consider the first-order *Born approximation* [Born et al., 1999], which approximates (3.6) by dropping the $u_{\text{scat}}[q](x';s)$ term from the right-hand side. This is further simplified with an approximation of the Green's function in the far-field limit [Born et al., 1999], yielding

$$d_k(r,s) \approx k^2 \int_\Omega e^{-ik(r-s)\cdot x'} q(x') \mathrm{d}x'. \tag{3.8}$$

We will refer to this linear approximation of the map from $q(x)$ to $d_k(r,s)$ as F_k. Note that $F_k q$ is proportional to the Fourier Transform of q evaluated at frequency vectors of the form $k(r-s)$. Since $r, s \in \mathbb{S}^1$ range over the unit circle, the frequency vectors $k(r-s)$ take on values throughout a disk with radius $2k$ centered at the origin. Thus, evaluations of the linearized forward model, $F_k q$, only contain the low-frequency components of q, while high-frequency components of q are in the kernel of the linearized forward model F_k [Chen, 1995].

Owing to its simplicity, (3.8) is often used as inspiration for the design of neural network architectures approximating the inverse map $d_k \mapsto q$. The networks emulate the FBP method [Natterer, 2001], which produces an estimate $\hat{q}$ of the scattering potential q as

$$\hat{q} = (F_k^* F_k + \mu I)^{-1} F_k^* d_k, \tag{3.9}$$

where F_k^* is the adjoint of F_k and μ is a regularization parameter that stabilizes the inversion of $F_k^* F_k$. The operator $(F_k^* F_k + \mu I)^{-1}$ can be implemented as a two-dimensional spatial convolution [Khoo and Ying, 2019, Fan and Ying, 2022, Li et al., 2022], while novel network architectures have been proposed to emulate $F_k^* \in \mathbb{C}^{N_q^2 \times N_r N_s}$ in a parameter-efficient manner. In particular, Fan and Ying [2022] and Zhang et al. [2023] suggest leveraging the

rotational equivariance of the forward model to emulate F_k^* with one-dimensional convolutions, after applying a far-field scaling and an appropriate coordinate transformation in [Fan and Ying, 2022, Equation 6]). We refer to the network described by Fan and Ying [2022] as FYNet.

3.3.2 Related Work

Deep learning has revolutionized linear inverse problems in imaging, advancing methods for superresolution, inpainting, deblurring, and medical imaging. Many of these advances stem from methods combining deep neural networks with optimization algorithms. For example, the *deep unrolling* paradigm [Monga et al., 2021] performs a fixed number of steps of an iterative algorithm and replaces certain operations with learnable mappings, which are parameterized by neural networks whose weights are learned from data. Components that remain fixed throughout training may reflect prior knowledge of problem parameters, such as explicit knowledge of the forward measurement model. In this setting, the network is usually trained end-to-end by minimizing the Euclidean distance between the network outputs and the true data. Another paradigm, called *plug-and-play denoising* [Venkatakrishnan et al., 2013], suggests that general image denoisers can be used in place of proximal operators for regularization functions, an important subroutine in many optimization routines for linear inverse problems in imaging. In this setting, neural network blocks are often trained to solve a different task, such as denoising corrupted signals, and then used inside the inversion algorithm. Ongie et al. [2020] provides a review of deep learning for inverse problems in imaging.

Several works in the wave scattering literature attempt to solve the inverse scattering problem by augmenting an optimization algorithm with components learned from data. At inference time, these methods require running an iterative optimization algorithm. Kamilov et al. [2017] develops a plug-and-play algorithm for inverse scattering, and show that various

off-the-shelf denoisers can be applied as proximal operators. Zhao et al. [2023] use an encoder network paired with a network emulating the forward model and suggest optimizing a latent representation of the scattering potential using stochastic gradient descent. When only phaseless measurements of the scattered wave field u_{scat} are available, Deshmukh et al. [2022] propose a network unrolling proximal gradient descent, where the proximal operator is a neural network learned from data. For the inverse obstacle scattering problem in two dimensions, Zhou et al. [2023] propose using a fully-connected neural network to warm-start a Gauss-Newton algorithm. Ding et al. [2022] train a neural network to approximately invert a forward scattering process depending on temporal data, and use this approximate inverse as a nonlinear preconditioner for a nonlinear least squares optimization routine. In concurrent work, Zhang et al. [2024] use diffusion sampling to reconstruct scattering potentials and quantify uncertainty of the reconstructions.

Other methods propose to learn the inverse map directly from data. Recently, neural networks that are approximately invariant to discretization size have been proposed as methods of learning maps between general function spaces [Li et al., 2021b, Lu et al., 2021] and these general-purpose networks have been applied to inverse scattering [Ong et al., 2022]. Other networks have been designed to invert the forward scattering model in particular; see Chen et al. [2020] for a broad review of such approaches. In our work, we are particularly interested in FBP-inspired methods. The work of Khoo and Ying [2019] leverages the approximate low-rank structure of scattering operators to design their SwitchNet network. Fan and Ying [2022] propose a data transformation that facilitates emulating the adjoint of the linearized forward model via 1D convolutions. In our work, we consider how to combine multiple such network blocks to invert the multi-frequency forward map in (3.3). One way of combining these blocks is to learn each adjoint operator F_k^* as a separate neural network block and combine data to jointly emulate the learned filtering operators $(F_k^* F_k + \mu I)^{-1}$. This strategy is employed in Zhang et al. [2023], which is similar to our MFISNet-Parallel,

but uses a different parameterization for the layers emulating F_k^* and $(F_k^* F_k + \mu I)^{-1}$. Another strategy is to use the Wide-Band Butterfly Network [Li et al., 2022, 2021a], which hierarchically merges information from different frequencies in the network block emulating F_k^*. We provide more detail and commentary about methods of combining network blocks to form multi-frequency networks in Section 3.5.2.

Finally, we note in passing that the design of our method is inspired by *homotopy methods* [Watson and Haftka, 1989]. These methods solve a sequence of sub-problems of increasing difficulty, gradually transforming a simple (but uninformative) optimization problem to the optimization problem of interest and using solutions to a given sub-problem to warm-start local search methods for subsequent sub-problems. Such a sequence can be constructed explicitly (*e.g.*, by varying regularization levels) or implicitly; for example, *curriculum learning* [Bengio et al., 2009] progressively adjusts the training data distribution from "easy" to "hard" samples and has been used to train physics-informed neural networks in challenging problem settings [Krishnapriyan et al., 2021, Huang et al., 2022].

3.4 Recursive Linearization and Our Method

We propose a neural network that learns to approximate the multi-frequency inversion map from training data. To design the network and training algorithm, we draw inspiration from the recursive linearization method for inverse scattering, which we briefly review below.

3.4.1 Recursive Linearization

Recursive linearization is a classical method for solving the inverse scattering problem, introduced by Chen [1995]. In spite of the nonlinearity of the true forward scattering model described in (3.6), recursive linearization breaks the inverse problem into a series of simpler problems, each of which corresponds to a linear inverse problem. In this section we discuss the intuition behind this strategy.

Recall from our discussion in Section 3.3.1 that the forward map evaluated at low incident wave frequencies k acts approximately like a low-pass filter with cutoff frequency $2k$. At first glance, this suggests that observing $\mathcal{F}_k[q]$ for a high value of k is sufficient for high-resolution recovery of q. However, when viewed from an optimization perspective, it becomes clear that this problem is increasingly challenging for large values of k. For example, one might consider the nonlinear least-squares problem

$$\underset{\hat{q}}{\operatorname{argmin}} \ \|d_k - \mathcal{F}_k[\hat{q}]\|_2^2. \tag{3.10}$$

To illustrate the challenges for the optimization formulation with increasing values of k, we consider a simple example where q is known to be a Gaussian bump with a given spread parameter and unknown amplitude. Given observations $d_k = \mathcal{F}_k[q]$ and a numerical PDE solver to calculate $\mathcal{F}_k[\cdot]$, one could estimate the amplitude of q by solving a minimization problem similar to (3.10), but only searching over the unknown amplitude. In Figure 3.2, we plot this objective as a function of the amplitude of $\hat{q}$ for a range of values of k. For large values of k, the objective function is highly oscillatory and contains many spurious local minima. Typically this suggests that it can be challenging to locate the global minimum unless there is a scheme to initialize estimates close enough to the global minimum to avoid getting stuck elsewhere. However, Figure 3.2 also shows that the objective function oscillates much more slowly at the smallest value of k while sharing the same global minimum. This suggests that the low-frequency observations can be used to get close to the global minimum, even though they are diffraction-limited and cannot resolve high-frequency components.

The recursive linearization algorithm leverages this insight by solving a sequence of inversion problems at increasing wave frequencies k. Crucially, each sub-problem uses the output of the previous sub-problem to initialize a new optimization problem. This method was introduced in Chen [1995] and further developed in Bao and Liu [2003], Borges et al. [2017]. At iteration t, the algorithm uses the previous estimate $\hat{q}_{k_{t-1}}$ along with a new set of obser-

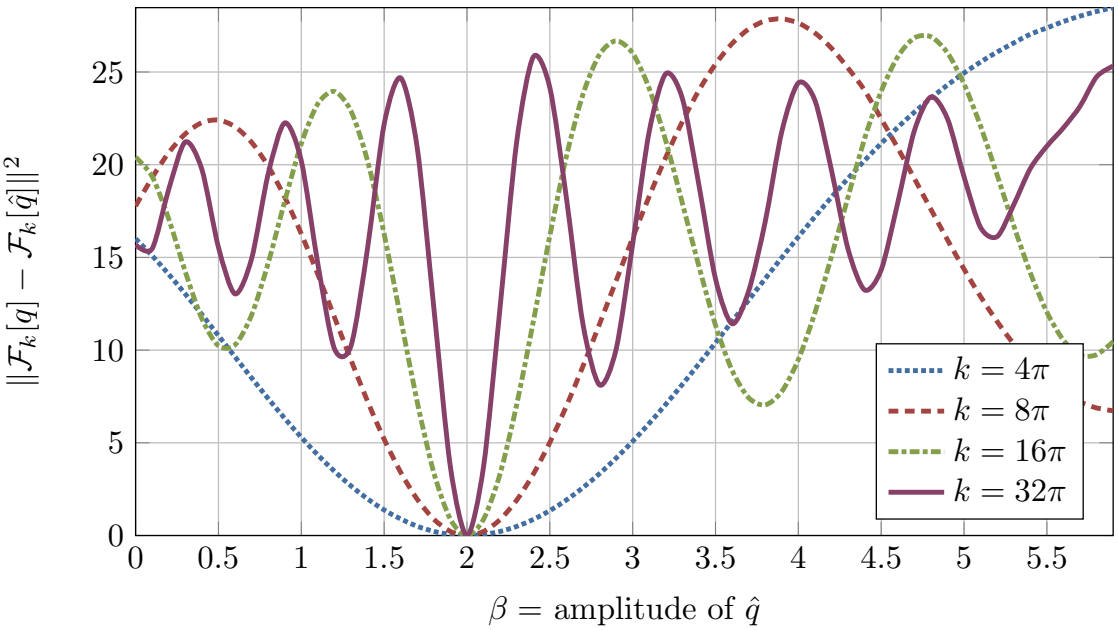


Figure 3.2: Even in a highly stylized setting, accurately and reliably inverting $\mathcal{F}_k$ is difficult for high frequencies k. Suppose the ground-truth scattering potential is $q(x) = \beta \exp(-\frac{\|x\|^2}{2\sigma^2})$, a Gaussian bump with known spread parameter $\sigma = 0.1$ but unknown amplitude β. We show the optimization landscape that arises from searching over different amplitudes. At low incident wave frequencies, this optimization landscape is smooth and has a large basin of attraction. However, as the incident wave frequency increases, the optimization landscape becomes highly oscillatory, requiring a nearly exact initialization to guarantee convergence to the ground truth. In experimental settings, the parameterization of q is often high-dimensional, which requires higher frequency data to resolve high-frequency information in q. This numerical example was inspired by Bao and Liu [2003].

vations d_{k_t}, and it calculates an update δq that minimizes the ℓ_2 distance in measurement space:

$$\underset{\delta q}{\operatorname{argmin}} \ \|d_{k_t} - \mathcal{F}_{k_t}[\hat{q}_{k_{t-1}} + \delta q]\|_2^2 \tag{3.11}$$

The value of $\hat{q}_{k_{t-1}}$ may make it possible to avoid spurious local minima, but this problem is still difficult since an iterative optimizer would require solving a PDE at each of its iterations. However, $\mathcal{F}_{k_t}[\hat{q}_{k_{t-1}} + \delta q]$ is well-approximated by a first-order Taylor expansion about $\hat{q}_{k_{t-1}}$ when δq is small or when it does not contain low-frequency information [Chen, 1995]. This

motivates the following surrogate for the optimization problem in (3.11):

$$\underset{\delta q}{\operatorname{argmin}} \ \| d_{k_t} - \left(\mathcal{F}_{k_t}[\hat{q}_{k_{t-1}}] + D\mathcal{F}_{k_t}[\hat{q}_{k_{t-1}}]\delta q \right) \|_2^2, \tag{3.12}$$

where $D\mathcal{F}_{k_t}[\hat{q}_{k_{t-1}}]$ denotes the Fréchet derivative of the forward model at $\hat{q}_{k_{t-1}}$. The action of $D\mathcal{F}_{k_t}[\hat{q}_{k_{t-1}}]$ and its adjoint, $D\mathcal{F}^*_{k_t}[\hat{q}_{k_{t-1}}]$, can be computed using the adjoint-state method [Bao and Liu, 2003, Borges et al., 2017]. The resulting algorithm is akin to a Gauss-Newton method; critically, each sub-problem of the form shown in (3.12) is a linear least-squares problem. We outline a sketch of the recursive linearization algorithm in Algorithm 1.

The recursive linearization algorithm is very demanding computationally. Each iteration requires solving N_s large-scale PDEs and a high-dimensional least-squares problem, which quickly creates a large computational burden when producing high-resolution solutions. In a classical setting without machine learning, the frequencies should be spaced close to each other for best results. Chen [1997] uses $k = 1, 2, \ldots, 9$ in their numerical experiments, while Borges et al. [2017] uses $k = 1, 1.25, \ldots, 70$, which they report takes around 40-50 hours per sample to produce a single 241×241 pixel image.

Algorithm 1: Recursive Linearization for Inverse Scattering based on Chen [1995, 1997], Bao and Liu [2003], Borges et al. [2017]

Input: Multi-frequency data $\{d_{k_1}, d_{k_2}, \ldots, d_{k_{N_k}}\}$

$\hat{q}_{k_1} \leftarrow \left(F^*_{k_1} F_{k_1} + \mu I \right)^{-1} F^*_{k_1} d_{k_1}$

for $t = 2, \ldots, N_k$ **do**

  Compute $\mathcal{F}_{k_t}[\hat{q}_{k_{t-1}}]$ and $D\mathcal{F}_{k_t}[\hat{q}_{k_{t-1}}]$

  $\delta q_{k_t} \leftarrow \underset{\delta q}{\operatorname{argmin}} \ \| d_{k_t} - \left(\mathcal{F}_{k_t}[\hat{q}_{k_{t-1}}] + D\mathcal{F}_{k_t}[\hat{q}_{k_{t-1}}]\delta q \right) \|_2^2$

  $\hat{q}_{k_t} \leftarrow \hat{q}_{k_{t-1}} + \delta q_{k_t}$

Result: Final estimate $\hat{q}_{k_{N_k}}$

Although recursive linearization is computationally expensive as stated, we believe that one of the key features of recursive linearization is the way that it breaks the recovery process

into multiple steps, each of which refines the estimate from the previous step using data of a higher frequency. We will refer to this step-wise recovery strategy as progressive refinement.

Progressive refinement facilitates the recovery process, since each step is only responsible for a correction to the estimate of the scattering potential within a frequency band. Focusing on this strategy also allows us to look for machine learning methods that do not explicitly emulate $\mathcal{F}_k[\cdot]$ or $D\mathcal{F}_k[\cdot]$, which are expensive to compute.

To this end, we consider a generalization of recursive linearization where we replace lines 3 and 4 in Algorithm 1 by describing the inner loop as a generic refinement step:

$$\delta q_{k_t} = \text{RefinementStep}_{k_t}(\hat{q}_{k_{t-1}}, d_{k_t}) \qquad \text{for } t = 2, \ldots, N_k \tag{3.13}$$

where $\text{RefinementStep}_{k_t}(\hat{q}_{k_{t-1}}, d_{k_t})$ refers to the update calculated for estimate $\hat{q}_{k_t}$ given data d_{k_t} and can be implemented using a neural network. We will propose and discuss a network architecture in the next section.

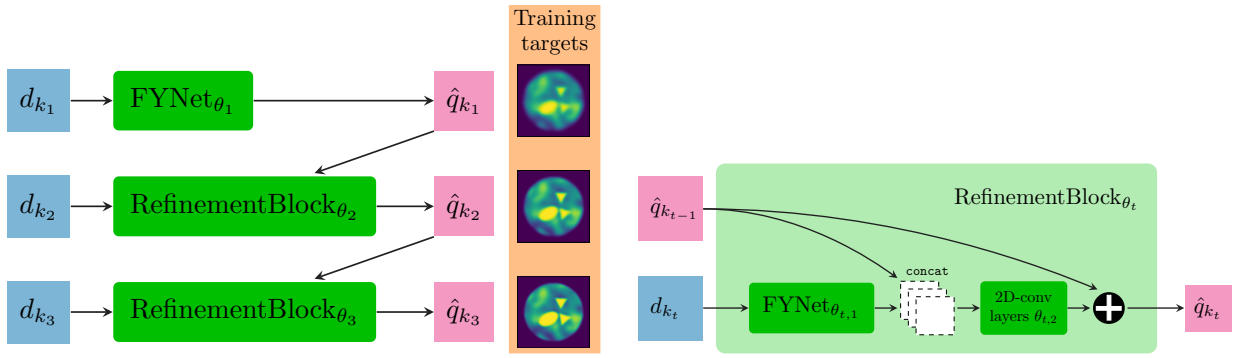
3.4.2 Our Method

We use Algorithm 1 as inspiration for the design of our neural network architecture and training method. In particular, we focus on the following two crucial aspects of Algorithm 1:

Progressive refinement: The algorithm builds intermediate estimates of the scattering potential which are progressively refined with the introduction of new data.

Homotopy through frequency: The iterative refinements from the first step form a homotopy from low to high-frequency measurements. As a result, updates at step t contain high-frequency information relative to k_{t-1}.

To emulate the progressive refinement structure, we propose a network with a residual structure and skip connections. The network comprises multiple blocks, one for each incident wave frequency k_t, $t = 1, \ldots, N_k$. The input to each block is measurement data d_{k_t} collected



(a) MFISNet-Refinement Architecture (b) Refinement Block

Figure 3.3: Our MFISNet-Refinement architecture is designed to emulate the recursive linearization algorithm. Figure 3.3a shows that our network proceeds by making an initial low-frequency reconstruction and then making a series of updates given higher-frequency data and an estimate of the scattering potential. The network is trained to match the intermediate reconstructions to low-pass filtered versions of scattering potentials from the training set. One example collection of such filtered scattering potentials is shown. Figure 3.3b shows that our refinement block is a simple extension of the FYNet architecture from Fan and Ying [2022]. By using a skip connection in this block, we ensure the network only needs to predict an update to the estimated scattering potential.

at a particular incident wave frequency k_t. The input passes through an FYNet block [Fan and Ying, 2022], which approximately inverts the forward model. The output of the FYNet block is then concatenated with the output of the previous block, $\hat{q}_{k_{t-1}}$, and the concatenation is passed to 2D convolutional layers for a second filtering step. Finally, a skip connection adds $\hat{q}_{k_{t-1}}$ to the output of the last convolutional layer of the block, producing the next estimate $\hat{q}_{k_t}$. The network's architecture is shown in Figure 3.3; we call the resulting network "MFISNet-Refinement." Note that, under this construction, the FYNet blocks could be replaced by any other neural network architecture designed for the single-frequency inverse scattering problem.

To emulate the homotopy through frequency, we design a training method to ensure each successive block adds higher-frequency information to the estimate of the scattering potential. Under the Born approximation (3.8), we know d_k contains information about q up to frequency limit $2k$. This suggests that given data d_{k_t}, we should be able to reconstruct

the frequency components of q up to $2k_t$. To reflect this, we train the output of block t with the following loss function:

$$L_t(\hat{q}_{k_t}; q) = \mathbb{E}_{q\sim\mathcal{D}_n}\left[\|\hat{q}_{k_t} - \mathsf{LPF}_{2k_t} q\|^2\right] \tag{3.14}$$

In (3.14), LPF_{2k_t} is a low-pass filter with approximate cutoff frequency $2k_t$, implemented as a Gaussian filter to avoid ringing artifacts; its frequency response is given in the Fourier domain by

$$\widetilde{\mathsf{LPF}}_{f_{\mathsf{cut}}}(f) := \exp\left(-\frac{1}{2}\left(\frac{f}{f_{\mathsf{cut}}/\sqrt{2\log 2}}\right)^2\right), \tag{3.15}$$

given a target cutoff frequency f_{cut}. Note that we shrink the standard deviation by a factor of $\sqrt{2\log 2}$ so that the filter's half-width at half-maximum matches f_{cut}.

To train the network, we first adjust the weights of each block in a sequential fashion and then perform a final training step which fine-tunes all of the blocks jointly; the training procedure is summarized in Algorithm 2.

Algorithm 2: Training Procedure

Input: Randomly-initialized neural network parameters $\{\theta_1, \ldots, \theta_{N_k}\}$;
Training data samples $\mathcal{D}_n := \left\{(q^{(j)}, d^{(j)}_{k_1}, \ldots, d^{(j)}_{k_{N_k}})\right\}_{j=1}^{n}$.

for $t = 1, ..., N_k$ **do**
  Set θ_t as trainable, and freeze all other weights
  if $t < N_k$ **then**
    Train θ_t by optimizing L_t `// Equation (3.14)`
  else
    Train θ_t by optimizing $\|\hat{q}_{k_{N_k}} - q\|_2^2$
Set all weights as trainable
Train all weights by optimizing $\|\hat{q}_{k_{N_k}} - q\|_2^2$

Result: Trained neural network parameters $\{\theta_1, ..., \theta_{N_k}\}$.

3.4.3 Implementation of FYNet

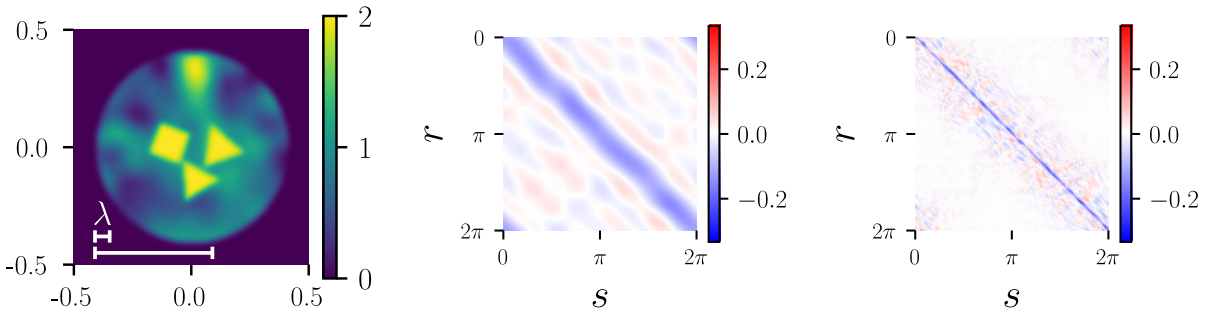
We implement the inversion network described in [Fan and Ying, 2022] and call it FYNet. This method suggests applying the far-field scaling and a transformation from the receiver and source direction coordinates (r, s) to a new set of variables as summarized in [Fan and Ying, 2022, Equation (6)], which we perform using bicubic interpolation. We describe this transformation in Equation (3.16). We split the complex-valued input into real and imaginary parts along a channel dimension. To implement the action of the adjoint operator F_k^*, we use a composition of three 1-dimensional convolutional layers. We implement the convolutional layers as learnable in the Fourier domain because the expression for F_k^* derived in [Fan and Ying, 2022] is local in frequency, but not space. To implement the filtering operator $\left(F_k^* F_k + \mu I\right)^{-1}$, we use a composition of three 2-dimensional convolutional layers. All layers but the last one use ReLU activations. This network outputs an estimate of the scattering potential on a regular polar grid, in coordinates (ρ, ϕ). Following [Fan and Ying, 2022], we train the network by minimizing the difference between predictions and targets on the polar grid. We transform to Cartesian coordinates for visualization and computing final test statistics.

3.5 Experiments

In this section, we describe the setting and results for our numerical investigation of the efficacy of our proposed method.

3.5.1 Dataset and Data Generation

Distribution of Scattering Potentials We define a distribution of scattering potentials $\mathcal{D}$ which has nonzero spatial support on the disk of radius 0.4, with a smoothly varying random background occluded by three randomly placed and randomly sized piecewise-constant



(a) Scattering potential q (b) Measured d_k, $k = 4\pi$ (c) Measured d_k, $k = 32\pi$

Figure 3.4: Figure 4.7a shows a typical example from our distribution of scattering potentials, drawn from the test set. Our distribution of scattering potentials has a random low-frequency background field, occluded by piecewise constant geometric shapes. The bottom-left corner shows the wavelength of two incident waves with frequencies $k = 4\pi$ and $k = 32\pi$. Figures 3.4b and 3.4c show the output of the forward model applied to this scattering potential, using these incident frequencies. The real part of u_{scat} is shown.

shapes. We normalize the scattering potential so the background has minimum and maximum values 0 and 2 respectively, and we normalize the piecewise-constant shapes to have value 2. Figure 4.7a shows one such scattering potential from our distribution.

Notably, the contrast of these scattering potentials $\|q\|_\infty = 2$, which is much larger than the contrast used in distributions to evaluate other machine learning methods in the shape reconstruction regime Fan and Ying [2022], Li et al. [2022]. The high-contrast regime is an important experimental setting because it ensures the nonlinearity of the forward model, which is the difficult and interesting problem setting, is captured. The non-constant background also adds to the difficulty of the problem by increasing $\|q\|_2$, which adds to the nonlinearity of the forward model. It also adds much more entropy to $\mathcal{D}$. We use this model to reflect experimental conditions in imaging tasks, wherein backgrounds are rarely known, constant, or homogeneous.

Implementation of $\mathcal{F}$ To implement the forward model, we implement a numerical PDE solver to compute solutions of (3.1). We implement this by discretizing the scattering domain Ω with a (N_q, N_q) regular grid, with $N_q = 192$. We transform (3.1) into the Lippmann-

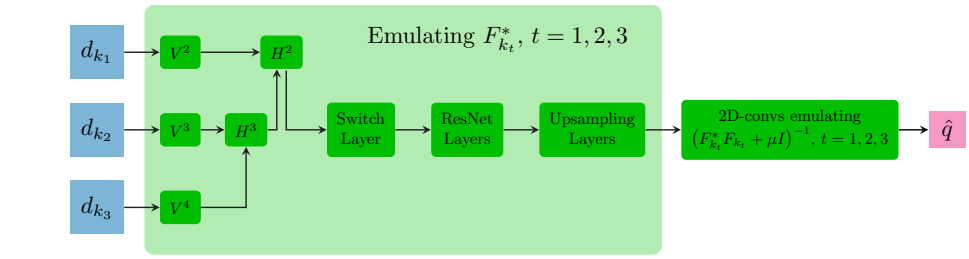
Schwinger integral equation and recast the latter as a sparse linear system, which we accelerate using fast Fourier transforms and hardware acceleration. We solve this linear system with GMRES [Saad and Schultz, 1986] implemented by SciPy [Virtanen et al., 2020] to a relative tolerance of 10^{-2}. This formulation allows us to compute the solution u_{scat} on a large, distant ring placed at radius $R = 100$. We compute the solution at $N_r = 192$ equally-spaced positions on this ring, and we repeat this process for each of the $N_s = 192$ equally-spaced source directions. The sources and receivers are located on the same grid. The runtime for evaluating the full forward model for a given q at the lowest and highest incident wave frequencies considered requires 5 and 220 seconds respectively, using one NVIDIA® A40 GPU. Figures 3.4b and 3.4c show the solution u_{scat} produced by our implementation for low and high incident wave frequencies, respectively.

Fan and Ying [2022] use a coordinate transformation of the far-field scattering data from coordinates (r, s) to (m, h), where

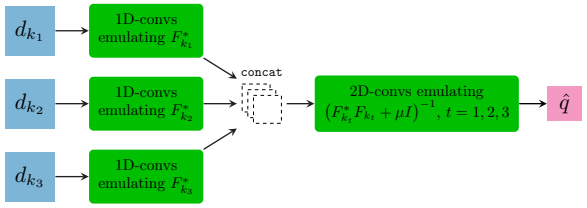
$$m := \frac{r+s}{2} \quad \text{and} \quad h := \frac{r-s}{2} \tag{3.16}$$

as described in [Fan and Ying, 2022, Equation (6)]. The variable m ranges from 0 to 2π, while h ranges from $-\pi/2$ to $\pi/2$. As in Fan and Ying [2022], we choose $N_m = 192$ and $N_h = 96$ to keep the angular sampling frequency fixed for all angular variables considered in the problem. We perform this transformation using bicubic interpolation, and the resulting grid has dimensions $(N_m, N_h) = (192, 96)$.

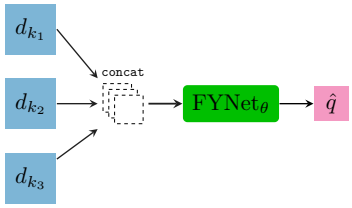
The FYNet blocks reconstruct images on a regular polar grid with $(N_\rho, N_\phi) = (96, 192)$ pixels, and the radial dimension of our polar grid extends to $\rho_{\max} = 0.5$. Finally, we use bicubic interpolation to transform the model's outputs to the (N_q, N_q) Cartesian grid for final visualization and error measurement.



(a) Wide-Band Butterfly Network



(b) MFISNet-Parallel



(c) MFISNet-Fused

Figure 3.5: Alternative neural network architectures for learning the multi-frequency inverse map. All blocks in dark green contain trainable parameters.

3.5.2 Alternative Multi-Frequency Methods

We wish to compare our method, MFISNet-Refinement, with other methods of learning an inverse to the multi-frequency forward map. We design two new methods of extending FBP-inspired single-frequency architectures to the multi-frequency setting. We use the FYNet architecture [Fan and Ying, 2022] to instantiate all three of our MFISNet methods, which allows us to focus on the effects caused by different methods of combining multi-frequency data. We show the architectures in Figure 3.5. We also compare our method with the Wide-Band Butterfly Network [Li et al., 2022, 2021a]. For broader context, we refer to Li et al. [2022] for comparisons between the Wide-Band Butterfly Network and classical multi-frequency methods that do not involve any machine learning such as Full Waveform Inversion (FWI) and a Least-Squares (LS) scheme. The authors show that the Wide-Band Butterfly Network achieves better accuracy than either FWI or LS with much faster inference times with minimal frequency or hyperparameter tuning.

MFISNet-Fused In MFISNet-Fused, we use an FYNet architecture that is constrained to learn all of the adjoint operators $F^*_{k_1}, \ldots, F^*_{k_{N_k}}$ jointly. We concatenate the inputs $[d_{k_1}, \ldots, d_{k_{N_k}}]$ along a new channel dimension, so the input array has shape $(N_m, N_h, N_k, 2)$. The concatenated input is then passed into a standard FYNet architecture, with the first layer having slightly wider convolutional channels. The shape of the weights in the 2D convolutional layers is constant in the problem dimensions, so the number of parameters in this network is dominated by the 1D convolutional weights and scales as $O(N_k^2 N_h)$.

MFISNet-Parallel In MFISNet-Parallel, we use an extension of FYNet that allows each adjoint operator $F^*_{k_1}, \ldots, F^*_{k_{N_k}}$ to be learned individually. Each input d_{k_t} is input to a unique 1D CNN which emulates $F^*_{k_t}$. After the adjoint operators are learned separately, the results are concatenated along a channel dimension, and the filtering operators $\left(F^*_{k_t} F_{k_t} + \mu I\right)^{-1}$ are emulated jointly by 2D CNN layers. Again, the number of 2D CNN weights does not scale with the problem dimensions, so the number of parameters in this network is dominated by the N_k 1D CNN blocks and scales as $O(N_k N_h)$.

Wide-Band Butterfly Network The Wide-Band Butterfly Network is introduced and defined in Li et al. [2022, 2021a]. Similar to MFISNet-Fused, this architecture also jointly parameterizes the adjoint operators $F^*_{k_1}, \ldots, F^*_{k_{N_k}}$, but it leverages the complementary low-rank property of $F^*_{k_t}$ [Khoo and Ying, 2019] to hierarchically merge the data using a butterfly network. For this network, we use code provided by the authors.[1] The reference implementation is limited to using data at three incident frequencies, so we only present results in this setting.

1. https://github.com/borongzhang/ISP_baseline

3.5.3 Stabilizing Reconstruction by Adding Frequencies

First, we test whether the intuition built in Section 3.4 is true in a machine learning context. We test whether machine learning methods that operate on data with multiple incoming wave frequencies are more accurate and stable than single-frequency machine learning methods. To make this comparison fair, we create a sequence of training datasets with number of incident wave frequencies $N_k \in \{1, \dots, 5\}$ and keep the amount of training data, nN_k, constant for each dataset by suitably adjusting the number of training samples n.

For the $N_k = 1$ dataset, we train an FYNet model, and for the other datasets, we train our three models: MFISNet-Fused, MFISNet-Parallel, and MFISNet-Refinement. For each model, we train for a fixed number of epochs and choose the model weights at the epoch at which a validation set of size $n/10$ is minimized. We also use the validation set to search over various hyperparameters, such as the size of 1D and 2D convolutional kernels, the number of channels in the convolutional layers, and optimization hyperparameters, such as step size and weight decay. For the Wide-Band Butterfly Network, we search over the rank of the butterfly factorization, as well as optimization hyperparameters, such as initial learning rate, batch size, and learning rate decay (Appendix B.2).

We present the results of this experiment in Table 3.1 and Figures 3.6 and 3.7. See also Appendix B.4 for additional empirical results, which include training and testing runtimes, visualizations of predictions on more held-out test samples, and visualizations of predictions of MFISNet-Parallel and MFISNet-Fused. The relatively poor performance of FYNet confirms our belief that we are in a challenging nonlinear problem regime. As more frequencies are added, the multi-frequency methods improve. The performances of MFISNet-Fused and MFISNet-Parallel are comparable, indicating that the distinction between learning adjoint operators separately or jointly does not have a large effect in this setting. For $N_k \geq 3$, the tested methods are uniformly outperformed by our method, MFISNet-Refinement. As we keep increasing N_k, the performance of MFISNet-Refinement plateaus. We hypothesize

that our method could be improved by choosing a different set of incident wave frequencies, possibly linearly-spaced in a smaller frequency band. The optimal choice of frequencies could also be learned from data in a reinforcement learning setting Jiang et al. [2024].

In the case of $N_k = 3$, the Wide-Band Butterfly Network underperforms the other methods. We hypothesize that the weaker performance of the Wide-Band Butterfly Network may be attributed to several factors: on the one hand, the butterfly factorization was inspired by analysis in a weak (linear) scattering regime, but our experiments are in a strong (nonlinear) scattering regime. Also, the Wide-Band Butterfly Network was previously tested in low-contrast settings with sub-wavelength scatterers and a known background, while we are in an experimental setting with high contrast and an unknown, inhomogeneous background.

Performance Comparison (Noiseless)

N_k	$[k_1, k_2, ...]$	n	**Method Name**	**Relative ℓ_2 Error**
1	$[32\pi]$	$10,000$	FYNet	0.159 ± 0.033
2	$[16\pi, 32\pi]$	$5,000$	MFISNet-Fused	0.158 ± 0.030
			MFISNet-Parallel	$\mathbf{0.144 \pm 0.029}$
			MFISNet-Refinement (Ours)	0.152 ± 0.032
3	$[8\pi, 16\pi, 32\pi]$	$3,333$	Wide-Band Butterfly Network	0.156 ± 0.037
			MFISNet-Fused	0.121 ± 0.024
			MFISNet-Parallel	0.107 ± 0.021
			MFISNet-Refinement (Ours)	$\mathbf{0.094 \pm 0.018}$
4	$[4\pi, 8\pi, 16\pi, 32\pi]$	$2,500$	MFISNet-Fused	0.103 ± 0.020
			MFISNet-Parallel	0.106 ± 0.021
			MFISNet-Refinement (Ours)	$\mathbf{0.082 \pm 0.019}$
5	$[2\pi, 4\pi, 8\pi, 16\pi, 32\pi]$	$2,000$	MFISNet-Fused	0.115 ± 0.022
			MFISNet-Parallel	0.109 ± 0.021
			MFISNet-Refinement (Ours)	$\mathbf{0.087 \pm 0.018}$

Table 3.1: When holding the number of forward model evaluations $= nN_k$ constant, methods trained on more frequencies outperform methods with fewer frequencies. The final column reports the relative ℓ_2 error mean $\pm$ one standard deviation computed over $1,000$ held-out test samples. The lowest mean for each incident frequency set is marked in boldface font.

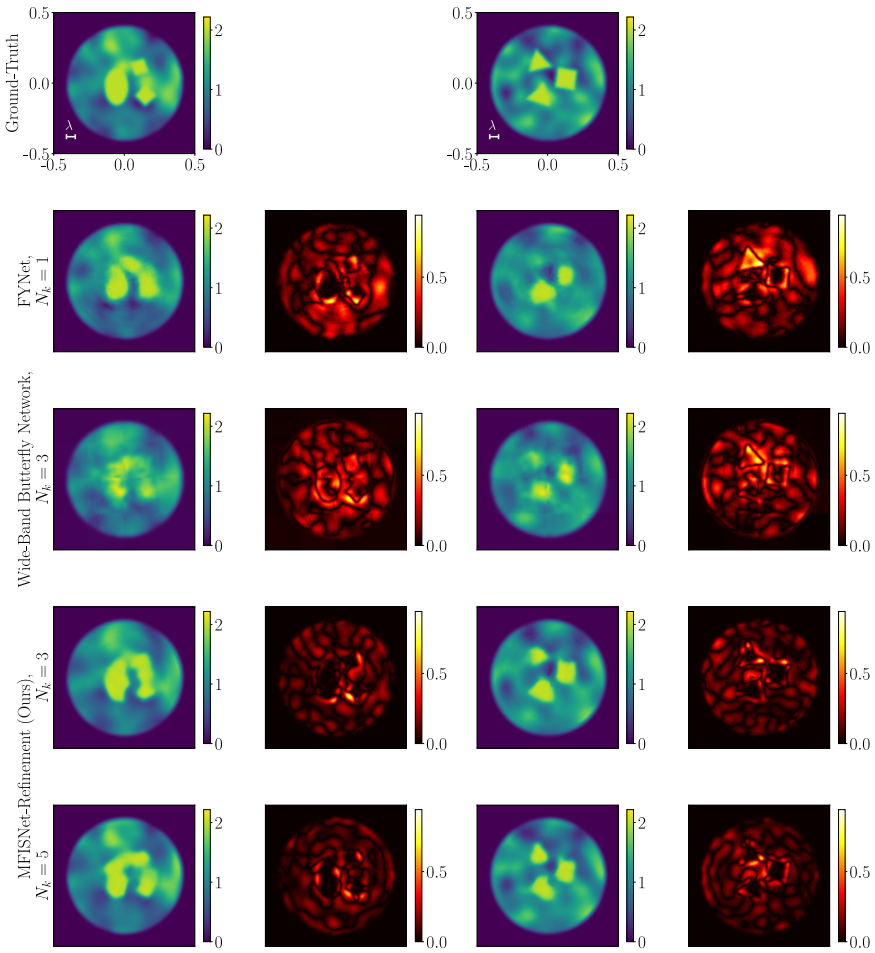


Figure 3.6: Sample predictions from models trained on different datasets. Predictions and errors on two held-out test samples are shown. The first row shows the ground-truth scattering potential; in this plot we show the wavelength corresponding to the maximum frequency $k = 32\pi$. The remaining rows show predictions and errors for FYNet, Wide-Band Butterfly Network, MFISNet-Refinement ($N_k = 3$), and MFISNet-Refinement ($N_k = 5$). See Appendix B.4 for additional samples and outputs from MFISNet-Parallel and MFISNet-Fused.

3.5.4 Measurement Noise

We now turn to the question of robustness against measurement noise. We repeat the experiment above but train and test the models using noisy inputs. We assume an additive noise model similar to Borges et al. [2017]. Given a clean input $d_k \in \mathbb{C}^{N_m \times N_h}$ and a desired

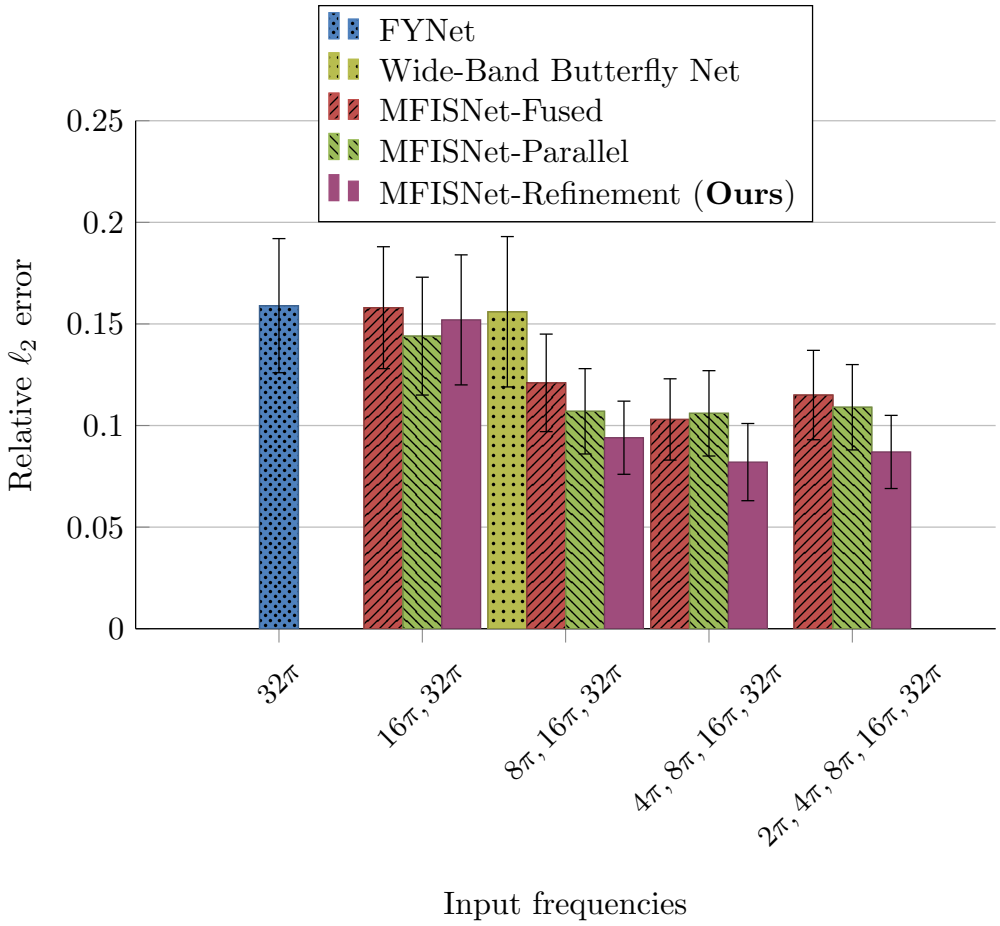


Figure 3.7: Illustration of the results of Table 3.1. The advantage of the MFISNet-Refinement model grows as more low-frequency data is used, even as the total volume of training data is held constant regardless of the number of frequencies.

noise-to-signal ratio δ, we define a noisy input $\tilde{d}_k$ as

$$\begin{aligned} &\tilde{d}_k = d_k + \sigma(Z_1 + iZ_2), \qquad (3.17)\\ \text{where} \quad &\sigma = \delta \frac{\|d_k\|_2}{\sqrt{2N_m N_h}};\\ \text{and} \quad &[Z_j]_{m,h} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0,1) \text{ for } j = 1,2 \text{ and } (m,h) \in [N_m] \times [N_h]. \end{aligned}$$

Under this noise model, the expected noise-to-signal ratio is $\mathbb{E}_{Z\sim\mathcal{N}(0,I)}[\|\tilde{d}_k - d_k\|/\|d_k\|] \approx \delta$. We do not alter the scattering potentials q_i in any way. We train and test all models using noisy inputs, and report the results of this experiment in Table 3.3. For the Wide-Band

N_k	n	**Method Name**	**Training Time (seconds)**	**Testing Time (seconds)**
1	$10,000$	FYNet	870.3	27.3
3	$3,333$	Wide-Band Butterfly Network	$6,453.6$	29.9
		MFISNet-Fused	326.2	27.8
		MFISNet-Parallel	256.1	27.8
		MFISNet-Refinement (Ours)	$2,033.0$	28.2
5	$2,000$	MFISNet-Fused	306.3	27.6
		MFISNet-Parallel	182.2	27.8
		MFISNet-Refinement (Ours)	$2,095.5$	28.8

Table 3.2: The time required to train and evaluate machine learning models. In different settings of N_k, we report the training time for each method. We also report the time required to make predictions on the entire set of $1,000$ held-out test samples, performed in ten batches of size 100. As we increase N_k, the number of samples n decreases, so MFISNet-Fused and MFISNet-Parallel see an increase in training speed. For MFISNet-Refinement, as we increase the number of frequencies, we also increase the number of training epochs. The two effects, smaller dataset and more epochs, approximately negate each other, so the training time for MFISNet-Refinement remains approximately constant for different N_k. The testing times for the FYNet and MFISNet models are all approximately equal because they are dominated by a final polar-to-Cartesian coordinate transformation step, which is performed on a single CPU core but can in principle be accelerated.

Butterfly Network, which takes d_k inputs in the original (r, s) coordinates, we add noise to the input in its original coordinates and divide by $\sqrt{2N_rN_s}$ instead. This results in an equivalent noise profile thanks to the normalization that is grid-invariant. We note that the models lose between 1% - 2% accuracy on test set, suggesting the methods are relatively robust to the presence of measurement noise. Again, our method, MFISNet-Refinement, outperforms all other methods at most values N_k. We present the results of this experiment in Table 3.3 and visualize the results in Figure 3.8.

3.5.5 Investigating the Training Method

Our method unfolds the reconstruction problem into a sequence of simpler frequency-dependent refinement steps. This emulates the sequential, frequency-dependent structure

Performance Comparison (Noisy, $\delta = 0.1$)

N_k	$[k_1, k_2, ...]$	n	**Method Name**	**Relative L2 Error**
1	$[32\pi]$	$10,000$	FYNet	0.163 ± 0.031
2	$[16\pi, 32\pi]$	$5,000$	MFISNet-Fused	0.164 ± 0.035
			MFISNet-Parallel	$\mathbf{0.148 \pm 0.029}$
			MFISNet-Refinement (Ours)	0.151 ± 0.031
3	$[8\pi, 16\pi, 32\pi]$	$3,333$	Wide-Band Butterfly Network	0.156 ± 0.037
			MFISNet-Fused	0.125 ± 0.025
			MFISNet-Parallel	0.110 ± 0.023
			MFISNet-Refinement (Ours)	$\mathbf{0.097 \pm 0.019}$
4	$[4\pi, 8\pi, 16\pi, 32\pi]$	$2,500$	MFISNet-Fused	0.102 ± 0.021
			MFISNet-Parallel	0.108 ± 0.019
			MFISNet-Refinement (Ours)	$\mathbf{0.086 \pm 0.019}$
5	$[2\pi, 4\pi, 8\pi, 16\pi, 32\pi]$	$2,000$	MFISNet-Fused	0.112 ± 0.022
			MFISNet-Parallel	0.103 ± 0.020
			MFISNet-Refinement (Ours)	$\mathbf{0.090 \pm 0.017}$

Table 3.3: Model evaluation on noisy train and test inputs with $\delta = 0.1$ (see (3.17) for a description of the noise model). The final column reports the relative ℓ_2 error mean $\pm$ one standard deviation computed over $1,000$ held-out test samples. The lowest mean for each incident frequency set is marked in boldface font.

of the recursive linearization method. Compared to existing methods, our method introduces both a new residual architecture and a new sequential training procedure, so a natural question would be how important each of these factors is to our method's success. To test this question, we investigate two different changes to our training method which remove the progressive refinement structure. We describe each adjustment below. The results are presented in Table 3.4. In this experiment, we find the accuracy of our method is relatively robust to perturbations of the training method presented in Algorithm 2. This indicates the advantage of our approach is primarily driven by the residual architecture.

No Homotopy through Frequency Rather than sequentially training each network block after the previous blocks have been optimized, we jointly train all of the blocks by

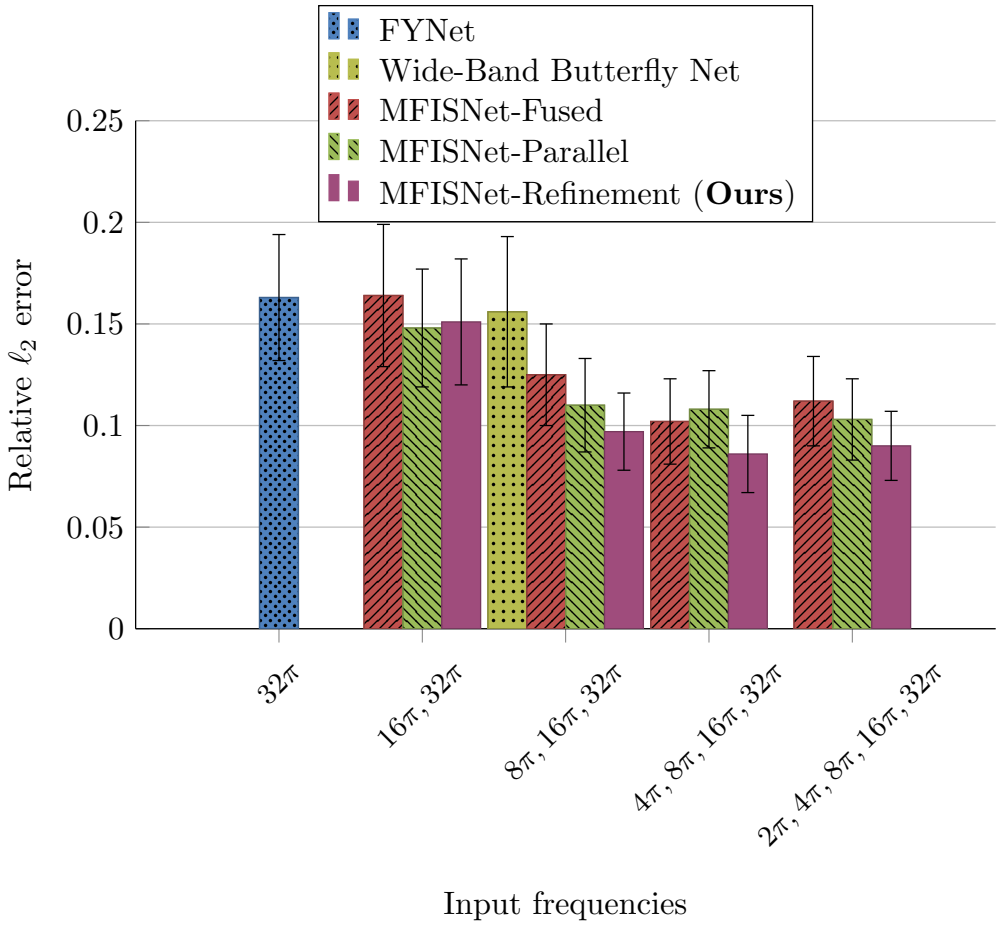


Figure 3.8: Illustration of results in the noisy measurement regime (Table 3.3). The relative performance of the models remains the same as in the noiseless case.

optimizing the loss function

$$\|\hat{q}_{k_{N_k}} - q\|_2^2 + \sum_{t=1}^{N_k-1} \gamma^{N_k-t} \|\hat{q}_{k_t} - \mathsf{LPF}_{2k_t} q\|_2^2 \tag{3.18}$$

Here γ is a hyperparameter which controls the relative importance of the different loss terms. We tuned over a few choices of γ; see Table B.6 for details.

No Progressive Refinement Instead of using the intermediate loss terms $\|\hat{q}_{k_t} - \mathsf{LPF}_{2k_t} q\|_2^2$, designed to promote specific network blocks learning different parts of the reconstruction, we train the network by only optimizing the final loss term $\|\hat{q}_{k_{N_k}} - q\|_2^2$. This is the standard

training loss used in most other works, including Khoo and Ying [2019], Fan and Ying [2022], Li et al. [2022].

Training Method	$[k_1, k_2, ...]$	n	**Relative L2 Error**
No Progressive Refinement	$[16\pi, 32\pi]$	$5,000$	$\mathbf{0.152 \pm 0.030}$
No Homotopy through Frequency			$\mathbf{0.152 \pm 0.030}$
Algorithm 2			$\mathbf{0.152 \pm 0.032}$
No Progressive Refinement	$[8\pi, 16\pi, 32\pi]$	$3,333$	$\mathbf{0.094 \pm 0.019}$
No Homotopy through Frequency			0.097 ± 0.018
Algorithm 2			$\mathbf{0.094 \pm 0.018}$
No Progressive Refinement	$[4\pi, 8\pi, 16\pi, 32\pi]$	$2,500$	0.091 ± 0.020
No Homotopy through Frequency			0.086 ± 0.018
Algorithm 2			$\mathbf{0.082 \pm 0.019}$
No Progressive Refinement	$[2\pi, 4\pi, 8\pi, 16\pi, 32\pi]$	$2,000$	$\mathbf{0.082 \pm 0.018}$
No Homotopy through Frequency			0.087 ± 0.016
Algorithm 2			0.087 ± 0.018

Table 3.4: Alternative training methods, designed to remove parts of the recursive linearization structure, produce models of similar accuracy to the training procedure discussed in Algorithm 2. This indicates the advantage of our MFISNet-Refinement method is driven primarily by its residual architecture. We describe the alternate training methods in Section 3.5.5. The optimal hyperparameters for these models are listed in Appendix B.2.

3.5.6 Investigating the Speed of Training Convergence

While Tables 3.1 and 3.3 show that MFISNet-Refinement is more accurate than other networks in most settings, Table 3.2 shows that training our network is slower than the baseline MFISNet-Parallel and MFISNet-Fused architectures. This is because our MFISNet-Refinement architecture is more complex than MFISNet-Parallel or MFISNet-Fused, and our training procedure (Algorithm 2) takes a block-wise approach which requires many training epochs. In this section, we conduct experiments to investigate whether we can reduce the time spent training MFISNet-Refinement. Because the results presented in Section 3.5.5 suggest that all three training methods considered, Algorithm 2, "No Homotopy through Frequency", and "No Progressive Refinement", produce models which are approximately similar

in accuracy, we consider training with all of these methods.

We introduce another training method, designed to mimic Algorithm 2 but converge in fewer training epochs. This method, which we call "Algorithm 2 + Warm-Start Initialization", uses the previous block's parameters θ_{t-1} to initialize the weights of the next block θ_t. This "warm start" is inspired by the fact that neighboring blocks in our MFISNet-Refinement architecture are designed to learn similar functions; each block is meant to increase the resolution of the estimated scattering potential by a small amount. Thus, the weights learned by the previous block may be a good starting point for the optimization of the next block. We give full pseudocode for this algorithm in Appendix B.3.

We set a early stopping criterion, which terminates training if the relative ℓ_2 error has not decreased by at least 10^{-3} over the past 15 epochs. We use this early stopping condition to terminate the "No Homotopy through Frequency" and "No Progressive Refinement" training methods. Similarly, we use this early stopping criterion to terminate each block in Algorithm 2. In Table 3.5, we measure the number of epochs used and runtime of each training method. We observe that the warm-start initialization decreases the number of epochs and training time of Algorithm 2. For $N_k = 3$ frequencies, this method converges faster than the other three methods. In both cases, using the warm start initialization incurs a slight decrease in accuracy while greatly increasing the speed of training convergence. Different early stopping criteria could achieve different tradeoffs between training speed and accuracy, and we leave the investigation of such criteria to future work.

The time required to complete an epoch of training for Algorithm 2 with and without warm-starting is much shorter than that of the "No Progressive Refinement" and "No Homotopy through Frequency" methods. This is because for the majority of epochs in Algorithm 2, only one block is updated at a time, making the backpropogation of gradients much faster. In the "No Progressive Refinement" and "No Homotopy through Frequency" methods, all trainable parameters are updated in each epoch.

Training Method	$[k_1, k_2, ...]$	n	Number of Epochs	Training Time (seconds)	Relative ℓ_2 Error
No Progressive Refinement	$[8\pi, 16\pi, 32\pi]$	3,333	90	1,200.6	0.096 ± 0.020
No Homotopy through Frequency			55	920.3	0.105 ± 0.020
Algorithm 2			215	1,226.4	0.094 ± 0.019
Algorithm 2 + Warm-Start Init.			195	802.8	0.098 ± 0.019
No Progressive Refinement	$[2\pi, 4\pi, 8\pi, 16\pi, 32\pi]$	2,000	45	811.6	0.095 ± 0.019
No Homotopy through Frequency			50	997.8	0.091 ± 0.017
Algorithm 2			260	1,417.5	0.087 ± 0.017
Algorithm 2 + Warm-Start Init.			210	1041.9	0.105 ± 0.020

Table 3.5: Using warm-starting in Algorithm 2 reduces the training runtime between 20 and 35%. In this table, we report time and epochs required to train MFISNet-Refinement models with different training methods, and the relative ℓ_2 errors of each fully-trained model. The warm-start initialization method and the convergence criterion are described in Section 3.5.6.

3.6 Conclusion

This chapter investigates the use of multi-frequency data in deep learning approaches to the inverse medium scattering problem in a highly nonlinear, full-aperture regime. We review standard optimization results for this problem, identify recursive linearization as an algorithm particularly well-suited for this problem, and use this insight to design a neural network architecture and training method. We experimentally evaluate our proposed approach, comparing against novel and previously-published methods for combining multi-frequency data. In these comparisons, we find our method outperforms the other methods across a wide range of data settings, including with and without measurement noise.

This work also leaves open important questions about machine learning in different multi-frequency data settings. We leave the investigation of these important problems to future work. The first setting is seismic imaging, where sources and receivers are located on one side of the scattering potential, resulting in limited-aperture measurements. Another setting to consider is full-aperture measurements, with a small fixed or frequency-dependent number of source and receiver directions, as N_s and N_r drive real-world costs when implementing an imaging system. Real-world imaging systems often encounter noise that is not well-approximated by an additive zero-mean Gaussian noise model; characterizing the robustness

of deep learning methods in more realistic noise settings is important future work. Finally, we suggest a distribution of scattering potentials with an unknown, smoothly varying background occluded by strongly scattering shapes. We note that an important open problem is to learn to *segment* the reconstruction into disjoint regions, containing only background or only the strong scatterers.

Author Contributions

- **Owen Melia:** Conceptualization; Data curation; Methodology; Software; Writing - original draft.
- **Olivia Tsang:** Conceptualization; Data curation; Methodology; Software; Writing - original draft.
- **Vasileios Charisopoulos:** Conceptualization; Methodology; Supervision; Writing - review & editing.
- **Yuehaw Khoo:** Conceptualization; Methodology; Supervision; Writing - review & editing.
- **Jeremy Hoskins:** Conceptualization; Methodology; Supervision; Writing - review & editing.
- **Rebecca Willett:** Conceptualization; Methodology; Supervision; Funding acquisition; Project administration; Writing - review & editing.

Data Availability

Our publicly-available GitHub repository https://github.com/meliao/mfisnets contains the following:

- Code for defining and training MFISNet-Refinement, MFISNet-Parallel, and MFISNet-Fused.
- Code for generating the dataset used in our experiments.

Our publicly-available dataset of $10,000$ training samples, $1,000$ validation samples, and $1,000$ test samples can be downloaded from https://doi.org/10.5281/zenodo.14514353.

Acknowledgements

The authors would like to thank Borong Zhang for sharing his implementation of the Wide-Band Butterfly Network. OM, OT, VC, and RW gratefully acknowledge the support of AFOSR FA9550-18-1-0166 and NSF DMS-2023109. RW and YK gratefully acknowledge the support of DOE DE-SC0022232. YK gratefully acknowledges the support of NSF DMS-2339439. The team gratefully acknowledges the support of the Margot and Tom Pritzker Foundation.

CHAPTER 4

HARDWARE ACCELERATION FOR HPS ALGORITHMS IN TWO AND THREE DIMENSIONS

4.1 Introduction

Many problems in scientific computing require solving systems of linear, elliptic partial differential equations (PDEs). Such PDEs can accurately model a variety of physics, such as wave propagation, electrostatics, and diffusion phenomena. Because analytical solutions of these equations are often unknown, the task of computing numerical solutions has been an area of active research for hundreds of years. Today, there are a myriad of numerical solution methods available, and many are tailored to particular classes of equations or to specific use cases. We are most interested in designing methods for settings such as inverse or control problems, where the PDE implicitly defines some functional which, along with its gradient, is evaluated sequentially hundreds or thousands of times in the inner loop of an iterative algorithm.

In these settings, fast direct solvers [Martinsson, 2019] are a compelling choice. These solvers are able to rapidly compute a high-accuracy solution operator and can rapidly evaluate the solution given new data by applying the solution operator, often at the cost of a few matrix-vector multiplications. Fast direct solvers are also preferable for certain PDEs with oscillatory solutions [Gillman et al., 2015], especially ones modeling wave propagation, as they do not incur a data-dependent iteration complexity cost associated with iterative solvers, which can be quite large [Ernst and Gander, 2012].

Recently, scientific computing has undergone a paradigm shift with the advent of general-purpose hardware accelerators, such as GPUs. These hardware accelerators allow for massively parallel computation—they have thousands of processor cores on a single chip—but have strict memory constraints, a resource profile very different from standard multicore

CPU architectures. This chapter explores hardware acceleration of fast direct solvers and introduces new methods to facilitate this acceleration.

In particular, we focus on the hierarchical Poincaré–Steklov family of algorithms [Martinsson, 2013, Gillman and Martinsson, 2014, Gillman et al., 2015], a class of direct solution methods for variable-coefficient elliptic PDEs. These methods are characterized by a nested dissection approach combined with a high-order composite spectral discretization. We identify the algorithmic structure of these algorithms which makes them amenable to GPU acceleration and introduce new techniques for reducing the memory footprint of these methods. For two-dimensional problems, we introduce a novel recomputation strategy that minimizes data transfer between the GPU and host memory. In three dimensions, we use an adaptive discretization method, which greatly reduces the algorithm's peak memory complexity. Our numerical examples show these ideas are useful in challenging applied settings such as wave propagation, inverse problems, and molecular biology simulations.

We focus on solving linear, elliptic partial differential equations of the form

$$\mathcal{L}u(x) = f(x), \qquad x \in \Omega, \tag{4.1}$$

$$u(x) = g(x), \qquad x \in \partial\Omega. \tag{4.2}$$

In Equation (4.1), $\mathcal{L}$ is a linear, elliptic, second-order partial differential operator with spatially-varying coefficient functions, and Ω is a square $\subset \mathbb{R}^2$ or cube $\subset \mathbb{R}^3$. Equation (4.2) specifies Dirichlet boundary data, but our methods can also solve problems with Robin or Neumann boundary data. In these problems, we assume we can evaluate the differential operator $\mathcal{L}$, the source f, and the boundary data g at a set of discretization points of our choosing. We represent the solution u by its restriction to the same set of discretization points and rely on high-order polynomial interpolation to evaluate u away from the discretization points. In this chapter, we refer to vectors with bold lowercase symbols such as $\boldsymbol{f}$ and matrices with bold uppercase symbols such as $\boldsymbol{A}$. We use x for the spatial variable, and

when we want to indicate Cartesian coordinates, we use $(x_1, x_2) \in \mathbb{R}^2$ and $(x_1, x_2, x_3) \in \mathbb{R}^3$. We use the subscript u_n to denote the outward-pointing boundary normal derivative of a function, and we use Δ to denote the Laplace operator, the sum of second derivatives in each dimension.

4.1.1 Chapter outline and contributions

In Section 4.2, we discuss related work, including algorithmic development for fast direct solvers and GPU-specific optimizations. In Section 4.3, we give an overview of the hierarchical Poincaré–Steklov method and discuss the potential for massively parallel implementations of the algorithm. In the rest of the chapter, we make the following contributions:

- We optimize data transfer patterns to accelerate our method applied to two-dimensional problems (Section 4.4.1).
- To alleviate peak memory requirements in three-dimensional problems, we extend the two-dimensional adaptive method of Geldermans and Gillman [2019] to three dimensions, develop the first adaptive 3D GPU-compatible HPS implementation, and provide numerical examples to demonstrate memory and accuracy tradeoffs (Section 4.4.2).
- We provide a range of numerical examples illustrating the application of our method, focusing on two settings: high-wavenumber scattering problems and the linearized Poisson–Boltzmann equation (Sections 4.5 and 4.6).
- We show our proposed algorithm and implementation can be combined easily with standard automatic differentiation software, which makes it particularly amenable to application in optimization, inverse problems, and machine learning contexts (Section 4.5).
- We make our JAX-based implementation publicly available at `https://github.com/meliao/ha-hps`.

4.2 Related work

HPS algorithms are built on two conceptual building blocks: composite high-order spectral collocation methods [Kopriva, 1998, Pfeiffer et al., 2003, Yang and Hesthaven, 2000], and nested dissection of the computational domain [George, 1973]. Composite spectral collocation methods are those which separate the computational domain into a set of disjoint elements, and use a high-order spectral collocation scheme to represent the problem and solution separately on each element. Nested dissection methods break the original problem into a series of subproblems defined on a hierarchy of subdomains. The careful ordering of subproblems reduces the overall computational complexity by leveraging knowledge about properties of the solution, i.e. continuity of the solution and its derivative. Composite spectral collocation and hierarchical matrix decomposition ideas were combined in integral equation methods [Ho and Greengard, 2012] for constant-coefficient PDEs.

Martinsson [2013] first proposed combining these elements in a fast direct solver for variable-coefficient linear elliptic PDEs. The proposed scheme discretizes and merges Dirichlet-to-Neumann (DtN) operators. Gillman and Martinsson [2014] proposed a compression scheme that leverages the structure of these DtN operators to build a solver with $O(n)$ computational complexity for n elements. To alleviate the instabilities observed when merging DtN operators for Helmholtz problems, Gillman et al. [2015] proposed a scheme that merges impedance-to-impedance (ItI) operators instead. Further analysis for this scheme was provided in Beck et al. [2022]. Modifications for three dimensions have been proposed, including Lucero Lorca et al. [2024], Hao and Martinsson [2016], Kump et al. [2025], which all build solvers for three-dimensional Helmholtz problems. To alleviate memory and computational complexity, Lucero Lorca et al. [2024] use an iterative method at the highest-level subproblems. In concurrent work to our own, Kump et al. [2025] approach three-dimensional problems with uniform discretizations using a hybrid GPU-CPU approach by combining the composite spectral collocation method with a two-level sparse direct solver. Fortunato et al.

[2021] use the ultraspherical spectral method to discretize triangular or quadrilateral mesh elements and compute solutions over polygonal domains by merging DtN operators. Fortunato [2024] develops a variant of the HPS method which merges DtN and ItI operators to solve PDEs on unstructured meshes of smooth two-dimensional surfaces.

There has been significant interest in the GPU acceleration of (low-order) iterative PDE solvers [Georgescu et al., 2013]. Accelerating these algorithms often requires the rapid application of an extremely sparse system matrix. Applying GPU acceleration to direct solvers [Abdelfattah et al., 2022, Ghysels and Synk, 2022, Li and Demmel, 2003] requires different techniques; the literature has mostly focused on sparse direct solvers which do not employ a nested dissection method. These sparse direct solvers often have much higher peak memory requirements and heterogeneous computation profiles when compared with iterative PDE solvers.

Other solvers have been designed directly for GPU acceleration. Yesypenko and Martinsson [2024a,b] designed a composite high-order spectral collocation method and associated sparse direct solver with highly heterogeneous computation patterns, which eases GPU acceleration. This method can solve variable-coefficient 2D problems very quickly; our method solves similar problems, but can also handle three-dimensional problems and interface with automatic differentiation. Developing a GPU-compatible implementation of the solver in Yesypenko and Martinsson [2024b], as well as the implementation of our method, has been greatly eased by the advent of high-performance hardware-accelerated linear algebra frameworks popularized by deep learning, such as PyTorch [Ansel et al., 2024] and JAX [Bradbury et al., 2018]. These frameworks are highly efficient for batched linear algebra tasks and implement automatic differentiation capabilities. Both are high-level packages that sit on top of the XLA compiler [Leary and Wang, 2017], which compiles and launches optimized kernels that execute on general-purpose GPUs. There has also been work creating automatic differentiation compatible PDE solvers in JAX, tailored for problems such as synchrotron

simulation [Diao et al., 2024], computational mechanics [Xue et al., 2023], and ordinary differential equations [Kidger, 2021].

4.3 Introduction to HPS methods

In this section, we provide an overview of the HPS algorithms used in this chapter, with a particular eye on their computational structure and the possibilities for GPU acceleration. Full algorithms are available in Appendices C.1 to C.3. We use different variants of this algorithm for merging different types of Poincaré–Steklov operators. A Poincaré–Steklov operator $T : g \mapsto h$ maps from one type of boundary data to another. Take, for example, a Dirichlet-to-Neumann (DtN) operator, which maps from Dirichlet data g on the boundary of Ω to Neumann data h on the same boundary:

$$g = u|_{\partial\Omega},$$
$$h = u_n|_{\partial\Omega},$$

where u satisfies Equation (4.1). Another example commonly used is an impedance-to-impedance (ItI) operator, which maps "incoming" to "outgoing" impedance data [Gillman et al., 2015]:

$$g = u_n + i\eta u|_{\partial\Omega},$$
$$h = u_n - i\eta u|_{\partial\Omega},$$

where u satisfies Equation (4.1). These Poincaré–Steklov operators are linear operators, and we work with their discretization $\boldsymbol{T}$. Throughout the algorithm, we also work with $\boldsymbol{g}$ and $\boldsymbol{h}$, vectors of incoming and outgoing boundary data evaluated at a set of discretization points.

It is important to remember that because we are solving a linear partial differential equation, we can decompose the solution $u(x)$ into a particular solution $v(x)$ and homogeneous solution $w(x)$ where $u(x) = v(x) + w(x)$. The particular solution $v(x)$ satisfies

$$\begin{cases} \mathcal{L}v(x) = f(x), & x \in \Omega, \\ v(x) = 0, & x \in \partial\Omega, \end{cases}$$

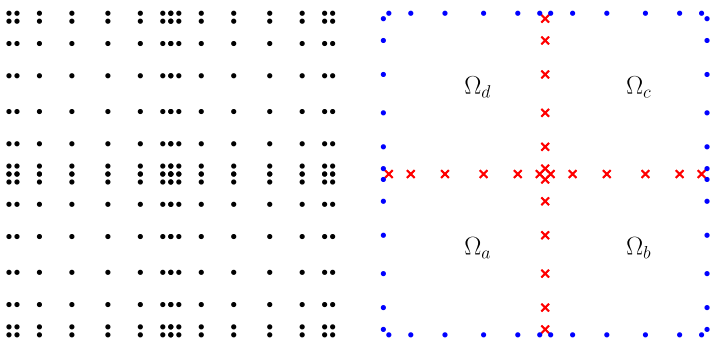
and the homogeneous solution $w(x)$ satisfies

$$\begin{cases} \mathcal{L}w(x) = 0, & x \in \Omega, \\ w(x) = g(x), & x \in \partial\Omega. \end{cases}$$

4.3.1 Discretization via composite high-order spectral collocation

To numerically solve Equation (4.1), we need to discretize $\mathcal{L}$, f, and g, and represent these objects in some finite-dimensional basis. We do this in two steps: a recursive partition of the domain Ω and a high-order spectral collocation scheme. The first step is to recursively partition Ω using a quadtree or octree structure down to a user-specified maximum depth L. We use n_{leaves} to denote the number of patches at the finest level of the spatial partition. In this section, we consider uniform discretization trees, so each tree with depth L will have $n_{\text{leaves}} = 2^{dL}$ elements at the lowest level in dimension $d = 2, 3$. In Section 4.4.2, we consider more general discretization trees. At times, it will be useful to describe the progress of the algorithm using language to describe the trees representing the spatial partition. To that end, we will sometimes refer to the elements as *nodes* and elements at the lowest level of the tree as *leaves*. The element at the highest level of the tree, which represents the entire computational domain, is sometimes called the *root*.

We discretize each leaf using a tensor product of Chebyshev–Lobatto points, with user-specified order p. This requires p^d points per leaf in dimension $d = 2, 3$. To represent the



(a) (b)

Figure 4.1: Visualizing the high-order composite spectral collocation scheme for a simple two-dimensional problem. Figure 4.1a shows the Chebyshev points on a two-dimensional problem with polynomial order $p = 8$ and $L = 1$ level of refinement. Figure 4.1b shows the Gauss–Lobatto points discretizing the boundaries of the leaves using order $q = 6$. When merging nodes together, we distinguish between the *exterior* boundary points, drawn with blue dots, and the *interior* boundary points, drawn with red $\times$'s.

boundary of each leaf, we use order-q Gauss–Legendre points. For simplicity and stability, we always use $q = p - 2$ [Gillman et al., 2015]. In two and three dimensions, there are $(2d)q^{d-1}$ boundary discretization points per leaf. We show the interior and boundary points in Figure 4.1. The resulting discretization has $N = n_{\text{leaves}}p^d = (2^L p)^d$ interior discretization points.

4.3.2 Local solve stage

At each leaf, we discretize the differential operator restricted to that leaf on the p^d Chebyshev discretization points. We call the resulting matrix $\boldsymbol{L}^{(i)}$ for leaf i. This matrix is a combination of Chebyshev spectral differentiation matrices [Trefethen, 2000] and evaluations of the spatially varying coefficient functions. We discretize the source function f on the same points and call the resulting vector $\boldsymbol{f}^{(i)}$. At this point in the algorithm, we solve a local boundary-value problem on each patch. We solve this problem using a "boundary bordering" technique which enforces the differential operator on the Chebyshev nodes interior to the leaf, and enforces a Dirichlet or impedance boundary condition on the Chebyshev

nodes on the leaf's boundary. At this point in the algorithm, we do not know the correct boundary values to enforce at each leaf, so we use $\boldsymbol{L}^{(i)}$ and $\boldsymbol{f}^{(i)}$ to precompute a solution map for the local problem. To precompute this local solution map, we construct a matrix $\boldsymbol{Y}^{(i)}$ which maps from any boundary data $\boldsymbol{g}^{(i)}$ to the corresponding homogeneous solution on the Chebyshev nodes. We also solve for $\boldsymbol{v}^{(i)}$, a vector evaluating the particular solution on the Chebyshev nodes. For each leaf, we also construct a Poincaré–Steklov matrix $\boldsymbol{T}^{(i)}$ and a vector of outgoing data $\boldsymbol{h}^{(i)}$. Computing $\boldsymbol{T}^{(i)}$ and $\boldsymbol{h}^{(i)}$ only requires multiplying $\boldsymbol{Y}^{(i)}$ and $\boldsymbol{v}^{(i)}$ with a fixed, precomputed operator which composes interpolation matrices from Chebyshev to Gauss–Legendre discretization points and spectral Chebyshev differentiation matrices.

Algorithm 3 shows that this stage of the algorithm is a long loop over linear algebra operations. The size of these operations is controlled by the polynomial order p, and we find these operations are efficient for the orders $p \leq 16$ considered in this work. The units of work inside the loop are *embarrassingly parallel*, meaning that one iteration does not depend on the output of any other iterations. Furthermore, because we hold p constant on all leaves, all of the linear algebra operations are homogeneous, meaning they all operate on the same sizes of matrices. This computational structure facilitates GPU acceleration by batching and parallelizing local solves. We give full details describing the local solve stage in Algorithms 12 and 13 in the Appendix.

4.3.3 Merge stage

After computing $\boldsymbol{T}^{(i)}$ and $\boldsymbol{h}^{(i)}$ for each leaf in the local solve stage, we begin merging nodes of the tree together. This process creates a hierarchy of solution operators which will later be used to propagate the boundary data on $\partial\Omega$ to the boundary of each leaf. For 2D problems, we merge nodes four at a time, and for 3D problems, we merge nodes eight at a time. Suppose we are merging a set of nodes $\{a, b, \ldots\}$ which all share parent node j. We have access to

Algorithm 3: Local solve stage. Full details are available in Algorithms 12 and 13.

Input: Discretized differential operators $\{\boldsymbol{L}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$; discretized source functions $\{\boldsymbol{f}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$; precomputed interpolation and differentiation matrices

for Leaf $i = 1, \ldots, n_{\text{leaves}}$ **do**
    Perform boundary bordering to $\boldsymbol{L}^{(i)}$
    Invert the resulting matrix `// Main computational work`
    Construct $\boldsymbol{Y}^{(i)}$, the interior solution matrix
    Construct $\boldsymbol{T}^{(i)}$, the Poincaré–Steklov matrix
    Construct $\boldsymbol{v}^{(i)}$, the leaf-level particular solution
    Construct $\boldsymbol{h}^{(i)}$, the outgoing boundary data

Result: Poincaré–Steklov matrices $\{\boldsymbol{T}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$; outgoing boundary data $\{\boldsymbol{h}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$; interior solution matrices $\{\boldsymbol{Y}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$; leaf-level particular solutions $\{\boldsymbol{v}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$

the following data:

- $\{\boldsymbol{T}^{(a)}, \boldsymbol{T}^{(b)}, \ldots\}$, the Poincaré–Steklov matrices of the nodes being merged.
- $\{\boldsymbol{h}^{(a)}, \boldsymbol{h}^{(b)}, \ldots\}$, the outgoing boundary data due to the particular solution of the nodes being merged.

At this point, it is helpful to distinguish between vectors that are defined along the *exterior* of the patches being merged and vectors that are defined along the *interior* of the patches being merged. We indicate these vectors with subscripts *ext* and *int*, respectively. See Figure 4.1b for a diagram of the interior and exterior points in a 2D merge operation. During a merge operation, we wish to precompute a solution operator which propagates the information from the exterior boundary points to the interior boundary points. This solution operator takes the form $\boldsymbol{g}_{\text{ext}}^{(j)} \mapsto \boldsymbol{S}^{(j)}\boldsymbol{g}_{\text{ext}}^{(j)} + \tilde{\boldsymbol{g}}^{(j)}$. In this equation, $\boldsymbol{S}^{(j)}\boldsymbol{g}_{\text{ext}}^{(j)}$ evaluates the homogeneous solution on the interior boundary points, and $\tilde{\boldsymbol{g}}^{(j)}$ evaluates the particular solution on the interior boundary points. As in Section 4.3.2, the boundary data $\boldsymbol{g}_{\text{ext}}^{(j)}$ is not available at this stage of the algorithm, but we can precompute the other parts of the solution operator. To that end, we want to compute the following objects in each merge

operation:

- $\boldsymbol{S}^{(j)}$, the propagation operator for node j, which maps incoming homogeneous boundary data from the exterior boundary points to the interior boundary points.
- $\tilde{\boldsymbol{g}}^{(j)}$, the incoming boundary data due to the particular solution evaluated at the interior boundary points.
- $\boldsymbol{T}^{(j)}$, the Poincaré–Steklov matrix for node j.
- $\boldsymbol{h}^{(j)}$, the outgoing boundary data for node j.

The first two outputs will be used in the final stage of the HPS method when applying the precomputed solution operator, and the final two outputs will be used in a future merge operation.

To compute the merge outputs, we set up a system of equations for a given $\boldsymbol{g}_{\text{ext}}^{(j)}$ and unknown $\boldsymbol{g}_{\text{int}}^{(j)}$ and solve using a Schur complement approach.[1] The constraints in this system come from our knowledge that the solution and its derivative will be continuous across merge interfaces. The resulting system of constraints is a linear system and can be written in a blockwise fashion:

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{g}_{\text{ext}}^{(j)} \\ \boldsymbol{g}_{\text{int}}^{(j)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{u}_{\text{ext}}^{(j)} - \boldsymbol{h}_{\text{ext}}^{(\text{child})} \\ -\boldsymbol{h}_{\text{int}}^{(\text{child})} \end{bmatrix}. \tag{4.3}$$

We build the matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ blockwise from the child Poincaré–Steklov matrices $\{\boldsymbol{T}^{(a)}, \boldsymbol{T}^{(b)}, \ldots\}$, build $\boldsymbol{h}_{\text{ext}}^{(\text{child})}$ and $\boldsymbol{h}_{\text{int}}^{(\text{child})}$ from the child outgoing boundary data $\{\boldsymbol{h}^{(a)}, \boldsymbol{h}^{(b)}, \ldots\}$, and use $\boldsymbol{u}_{\text{ext}}^{(j)}$ to represent the unknown solution evaluated on the exterior boundary points. See Appendices C.1 to C.3 for the exact definitions of these objects.

1. To the best of our knowledge, the presentation of the merge stage as a block linear system involving unknown outgoing boundary values first appeared in Chipman et al. [2024] for an algorithm merging 2D DtN matrices four at a time. We generalize this presentation to encompass 3D problems and merging ItI matrices as well.

At this point in the algorithm, we do not know $\boldsymbol{u}_{\text{ext}}^{(j)}$, which means we can not directly invert the linear system to solve for $\boldsymbol{g}_{\text{ext}}^{(j)}$ or $\boldsymbol{g}_{\text{int}}^{(j)}$. However, we can use a Schur complement approach to partially solve the system:

$$\begin{bmatrix} \boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C} & \boldsymbol{0} \\ \boldsymbol{D}^{-1}\boldsymbol{C} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{g}_{\text{ext}}^{(j)} \\ \boldsymbol{g}_{\text{int}}^{(j)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{u}_{\text{ext}}^{(j)} - \boldsymbol{h}_{\text{ext}}^{(\text{child})} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{h}_{\text{int}}^{(\text{child})} \\ -\boldsymbol{D}^{-1}\boldsymbol{h}_{\text{int}}^{(\text{child})} \end{bmatrix}.$$

Interpreting the rows of this linear system gives us the desired outputs:

$$\boldsymbol{u}_{\text{ext}}^{(j)} = \underbrace{\left(\boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C}\right)}_{\boldsymbol{T}^{(j)}} \boldsymbol{g}_{\text{ext}}^{(j)} + \underbrace{\boldsymbol{h}_{\text{ext}}^{(\text{child})} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{h}_{\text{int}}^{(\text{child})}}_{\boldsymbol{h}^{(j)}}, \tag{4.4}$$

$$\boldsymbol{g}_{\text{int}}^{(j)} = \underbrace{-\boldsymbol{D}^{-1}\boldsymbol{C}}_{\boldsymbol{S}^{(j)}} \boldsymbol{g}_{\text{ext}}^{(j)} + \underbrace{-\boldsymbol{D}^{-1}\boldsymbol{h}_{\text{int}}^{(\text{child})}}_{\tilde{\boldsymbol{g}}^{(j)}}. \tag{4.5}$$

Algorithm 4 gives pseudocode for the merge stage of the HPS algorithm. The majority of the computational work for each merge is inverting $\boldsymbol{D}$, which has size proportional to the number of discretization points along the merge interfaces. This matrix is quite small at the lowest levels of the discretization tree and grows as the algorithm proceeds to higher nodes in the tree. Similar to the local solve stage, each merge operation is dominated by linear algebra work, and the units of work inside the inner loop are embarrassingly parallel. This means we can easily use GPU acceleration to parallelize the inner loop of Algorithm 4. The outer loop is iterating over different levels, which means the computation during outer loop iteration ℓ depends on the outputs of the previous iteration. Because we only use a moderate number of refinement levels $L < 10$, we find Algorithm 4 executes very quickly on the GPU despite this dependency structure. We note that our choice to merge nodes four-to-one and eight-to-one (rather than the standard two-to-one) decreases the length of the outer loop by factors of two and three, respectively.

Algorithm 4: Merge stage. Full details are available in Appendices C.1.2, C.2.2 and C.3.2.

Input: Leaf-level Poincaré–Steklov matrices $\{\boldsymbol{T}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$; Leaf-level outgoing boundary data $\{\boldsymbol{h}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$

for Merge level $\ell = L-1, \ldots, 0$ **do**
  for Node j in level ℓ **do**
    Let $a, b, \ldots$ be the children of node j
    Use $\{\boldsymbol{T}^{(a)}, \boldsymbol{T}^{(b)}, \ldots\}$ to build blocks $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$, and $\boldsymbol{D}$
    Use $\{\boldsymbol{h}^{(a)}, \boldsymbol{h}^{(b)}, \ldots\}$ to build $\boldsymbol{h}_{\text{ext}}^{(\text{child})}$ and $\boldsymbol{h}_{\text{int}}^{(\text{child})}$
    Invert $\boldsymbol{D}$ `//Main computational work`
    Evaluate $\boldsymbol{T}^{(j)}$, $\boldsymbol{h}^{(j)}$, $\boldsymbol{S}^{(j)}$, $\tilde{\boldsymbol{g}}^{(j)}$ `//Equations (4.4) and (4.5)`

Result: Poincaré–Steklov matrices $\boldsymbol{T}^{(j)}$ for each node; outgoing boundary data $\boldsymbol{h}^{(j)}$ for each node; propagation operators $\boldsymbol{S}^{(j)}$ for each node; incoming particular solution data $\tilde{\boldsymbol{g}}^{(j)}$ for each node

4.3.4 Downward pass

In the final stage of the HPS method, all parts of the structured solution operators mapping $g \mapsto u$ have been computed. Now, we evaluate this structured solution operator by propagating information down the discretization tree from the boundary of the root to the interior of the leaves. The $\boldsymbol{S}^{(j)}$ matrices propagate the homogeneous boundary data to the merge interfaces, and the $\tilde{\boldsymbol{g}}^{(j)}$ vectors add back in the particular solution. This part of the HPS method is extremely fast, as it only involves matrix-vector products. As with the structure of the merge stage, the iterations of the inner loop are embarrassingly parallel and can be batched on the GPU. We show the pseudocode for this stage in Algorithm 5.

4.4 Hardware acceleration for HPS methods

While the algorithms presented in Section 4.3 have attractive computational complexity ($O(p^6 n_{\text{leaves}} + p^3 n_{\text{leaves}}^{3/2})$ in 2D and $O(p^9 n_{\text{leaves}} + p^6 n_{\text{leaves}}^2)$ in 3D) and possibilities for parallel execution, they also incur large memory footprints. At each step of the algorithm, dense solution matrices are precomputed and must be stored for future use. This poses a significant

Algorithm 5: Downward pass.

Input: Boundary data $\boldsymbol{g}$; propagation operators $\boldsymbol{S}^{(j)}$ for each node; incoming particular solution data $\tilde{\boldsymbol{g}}^{(j)}$ for each node; leaf-level interior solution matrices $\{\boldsymbol{Y}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$; leaf-level particular solutions $\{\boldsymbol{v}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$

for Merge level $\ell = 0, ..., L-1$ **do**
  for Node j in level ℓ **do**
    Look up $\boldsymbol{S}^{(j)}, \boldsymbol{g}^{(j)}$, and $\tilde{\boldsymbol{g}}^{(j)}$
    $\boldsymbol{g}_{\text{int}} = \boldsymbol{S}^{(j)}\boldsymbol{g}^{(j)} + \tilde{\boldsymbol{g}}^{(j)}$
    Let $a, b, \ldots$ be the children of node j
    Concatenate $\boldsymbol{g}_{\text{int}}$ and $\boldsymbol{g}^{(j)}$ to form $\{\boldsymbol{g}^{(a)}, \boldsymbol{g}^{(b)}, \ldots\}$
for Leaf $i = 1, \ldots, n_{\text{leaves}}$ **do**
  $\boldsymbol{u}^{(i)} = \boldsymbol{Y}^{(i)}\boldsymbol{g}^{(i)} + \boldsymbol{v}^{(i)}$

Result: Leaf-level solutions on the Chebyshev discretization points $\{\boldsymbol{u}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$

challenge for GPU acceleration as general-purpose GPU architectures have significantly more processor cores per unit of memory than standard multicore CPU nodes. While standard multicore compute nodes may have 1TB of available random-access memory (host RAM), high-end GPUs have only 80GB of on-device memory, with slow interconnects between the GPU and host RAM. This means that batched linear algebra operations are extremely fast on the GPU, but the overall algorithm is slowed by steps transferring data between the GPU and the host. Thus, to efficiently accelerate HPS algorithms on the GPU, one must devote significant thought to reducing the memory footprint of these algorithms. In this section, we introduce two ideas to reduce this memory footprint in two and three-dimensional problems.

4.4.1 Recomputation strategies to minimize communication costs

CPU-bound implementations of HPS methods in 2D often spend an order of magnitude longer in the local solve stage than in the merge or downward pass stages [Fortunato, 2024]. This suggests that HPS schemes can be greatly accelerated by placing the local solve stage computation on the GPU alone. Indeed, such savings have been observed in Yesypenko and Martinsson [2024a]. We observe that for the lowest levels of the merge stage, each unit of

computational work is similarly small, suggesting that the GPU can efficiently accelerate this part of the algorithm, too, provided the algorithm is correctly expressed to leverage its inherent parallelism. In many GPUs, the interconnect between the host device and the GPU's on-device memory is very slow in relation to the speed at which the thousands of processor cores can process data. This means that for large problem sizes, operations transferring precomputed solution operators to and from the GPU are a major impediment to fast execution as they incur a large latency and are often "blocking" operations, which require all parallel threads to complete before executing.

For large problem sizes in 2D, it is advantageous to delete some data computed in the early stages of the algorithm and recompute it later when necessary. This strategy increases the number of floating point operations on the GPU but minimizes costly data transfers. A leaf-level recomputation strategy for implementing HPS algorithms on a GPU is presented in Algorithm 6; a similar method is presented in Yesypenko and Martinsson [2024a]. The leaf recomputation strategy avoids transferring the $\{\boldsymbol{Y}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$ and $\{\boldsymbol{v}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$ by performing the local solve stage again at the end of the algorithm. Under this recomputation strategy, all of the leaf-level Poincaré–Steklov matrices must be transferred to RAM during the local solve stage, and then back to the GPU during the merge stage.

We find that it is advantageous to push the idea of reducing data transfers at the cost of more floating-point operations further. In our proposed recomputation strategy (Algorithm 7), we delete and recompute the products of the local solve stage and multiple levels of the merge stage. To implement this, we operate in batches defined by "complete subtrees", which are subtrees containing all of the descendants of a particular node j. We break the lowest levels of the discretization tree into the largest complete subtrees where the computations in Algorithms 3 and 4 can all fit into a GPU's on-device memory. The size of these maximal complete subtrees varies depending on GPU memory, polynomial order p, and floating-point datatype; in our experiments, these maximal complete subtrees usually have depth 6 or 7.

For each such complete subtree, we perform all of the local solve and merge operations, after which we only save the top-level Poincaré–Steklov matrix and outgoing boundary data vector. Because this is a small amount of data, we can store it on the GPU, and do not need to move the outputs to host RAM. After processing all of the subtrees, the final merge stages are performed on the GPU. The downward pass is evaluated sequentially on the different subtrees, at which point the local solve stage and low-level merges must be recomputed. This recomputation method was inspired by optimizations for contemporary deep learning architectures [Dao et al., 2022, Gu and Dao, 2024], which suggest kernel fusion, a technique that performs multiple steps of a sequential computation at once to keep the necessary data near the processor cores. We visualize the different recomputation methods in Figure 4.2.

In Figure 4.3, we compare the performance of our method across computer architectures and recomputation strategies. We consider two different architectures, a multicore Intel Xeon node with a 64-core processor, and a GPU architecture using a single Nvidia H100 GPU. Evaluating our method on the GPU gives us significant speedups over the multicore CPU architecture, even when using an implementation with no recomputation strategy, which transfers all precomputed matrices to host RAM after each algorithm step. The two recomputation strategies begin to diverge for problem sizes over 10^7 discretization points, at which point the precomputed matrices cannot all fit on the GPU. We also estimate the percentage of peak double-precision floating point operations per second (FLOPS) achieved by the different recomputation strategies. Our proposed recomputation strategy uses the most floating-point operations and has the fastest runtime, which means it reaches a higher percentage of peak FLOPS than the other implementations.

4.4.2 An adaptive discretization strategy to reduce memory complexity in 3D

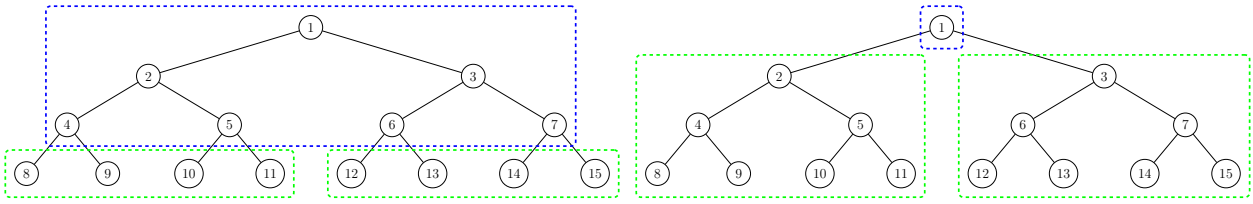
When extending from two to three dimensions, we face different computational challenges. A large part of the difficulty involves the size of the matrices arising in the merge stage

Algorithm 6: Leaf recomputation strategy.

Input: Differential operators $\{\boldsymbol{L}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$; source functions $\{\boldsymbol{f}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$; boundary data $\boldsymbol{g}$

Let b be the maximum batch size that can fit on the GPU
Split the indices $\{1, 2, \ldots, n_{\text{leaves}}\}$ into batches $I_1, I_2, \ldots, I_{\lceil n_{\text{leaves}}/b \rceil}$
for Batch j **do**
    Move $\{\boldsymbol{L}^{(i)}\}_{i \in I_j}$ and $\{\boldsymbol{f}^{(i)}\}_{i \in I_j}$ to the GPU
    Perform the local solve stage for this batch of leaves
    Delete $\{\boldsymbol{Y}^{(i)}\}_{i \in I_j}$ and $\{\boldsymbol{v}^{(i)}\}_{i \in I_j}$
    Transfer $\{\boldsymbol{T}^{(i)}\}_{i \in I_j}$ and $\{\boldsymbol{h}^{(i)}\}_{i \in I_j}$ to host RAM
Concatenate $\{\boldsymbol{T}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$ and $\{\boldsymbol{h}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$ and transfer to GPU
Perform all merge operations on the GPU and transfer all $\boldsymbol{S}^{(i)}$ and $\tilde{\boldsymbol{g}}^{(i)}$ to host RAM
Propagate boundary data to the leaves
Transfer leaf-level boundary data $\{\boldsymbol{g}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$ to host RAM
for Batch j **do**
    Move $\{\boldsymbol{L}^{(i)}\}_{i \in I_j}$, $\{\boldsymbol{f}^{(i)}\}_{i \in I_j}$, and $\{\boldsymbol{g}^{(i)}\}_{i \in I_j}$ to the GPU
    Compute local solutions $\boldsymbol{u}_{i \in I_j}^{(i)}$
    Transfer $\{\boldsymbol{u}^{(i)}\}_{i \in I_j}$ to host RAM

Result: Solutions on the Chebyshev discretization points $\{\boldsymbol{u}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$



(a) Leaf recomputation strategy (b) Subtree recomputation strategy

Figure 4.2: Comparing the batching patterns of the two different recomputation strategies. The leaf recomputation strategy in Figure 4.2a performs the local solve stage operations in large batches and then performs all of the merge stages in a separate batch. Our proposed subtree recomputation strategy (Figure 4.2b) performs local solves and multiple levels of the merge stage for a complete subtree of the discretization tree structure.

discussed in Section 4.3.3. For each merge operation, the matrix $\boldsymbol{D}$ must be inverted. This matrix has a number of rows and columns proportional to the number of discretization points that lie along the interfaces being merged. In two dimensions, this quantity is $4q$ at the leaf level and increases by a factor of 2 for each subsequent merge level. In three dimensions, the

Algorithm 7: Subtree recomputation strategy.

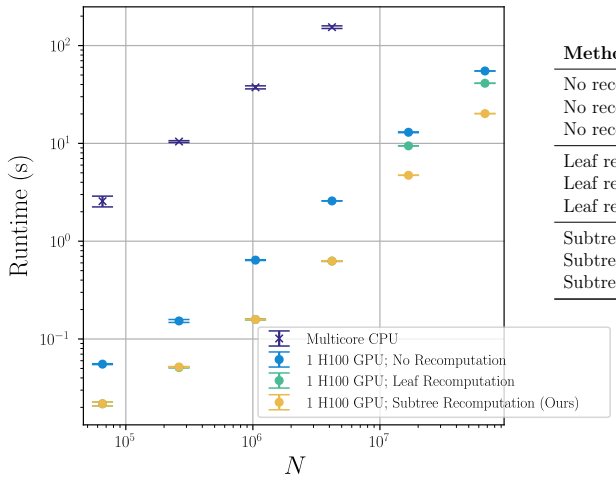
Input: Differential operators $\{\boldsymbol{L}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$; source functions $\{\boldsymbol{f}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$; boundary data $\boldsymbol{g}$

Let $M = m_1, m_2, ...$ be the roots of the maximal subtrees
for Subtree rooted at m_j **do**
    Let I_j be the set of leaves of the subtree
    Move $\{\boldsymbol{L}^{(i)}\}_{i\in I_j}$ and $\{\boldsymbol{f}^{(i)}\}_{i\in I_j}$ to the GPU
    Perform the local solve stage for this subtree
    Delete $\{\boldsymbol{Y}^{(i)}\}_{i\in I_j}$ and $\{\boldsymbol{v}^{(i)}\}_{i\in I_j}$
    Merge the leaves to the top of subtree m_j, deleting all outputs except $\boldsymbol{T}^{(m_j)}$ and $\boldsymbol{h}^{(m_j)}$
    Keep $\boldsymbol{T}^{(m_j)}$ and $\boldsymbol{h}^{(m_j)}$ on GPU
Perform final merge operations on the GPU
Propagate boundary data to the roots of the maximal subtrees
Transfer boundary data to host RAM
for Subtree rooted at m_j **do**
    Let I_j be the set of leaves of the subtree
    Move $\{\boldsymbol{L}^{(i)}\}_{i\in I_j}$, $\{\boldsymbol{f}^{(i)}\}_{i\in I_j}$ and $\boldsymbol{g}^{(m_j)}$ to the GPU
    Perform the local solve stage for this subtree
    Merge the leaves to the top of subtree j
    Propagate boundary information down the tree to the leaves
    $\boldsymbol{u}^{(i)} \leftarrow \boldsymbol{Y}^{(i)}\boldsymbol{g}^{(i)} + \boldsymbol{v}^{(i)}$
    Transfer $\{\boldsymbol{u}^{(i)}\}_{i\in I_j}$ to host RAM

Result: Solutions on the Chebyshev discretization points $\{\boldsymbol{u}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$

size of this matrix is $12q^2$ at the leaf level and increases by a factor of 4 for each subsequent merge level. Because this quantity grows very quickly as we increase the tree depth L, we quickly run out of memory required to store and invert the matrices on the GPU at the highest level of the merge stage. Performing the top-level merge operation is when the instantaneous memory footprint peaks, as space for $\boldsymbol{D}$, $\boldsymbol{D}^{-1}$, and various buffers must all be allocated on the GPU simultaneously. In contrast with other parts of the algorithm, this peak memory footprint is not reducible by strategies such as batching or data transfer.

To reduce the size of $\boldsymbol{D}$ at the final merge step, we propose to extend the adaptive HPS method presented for 2D problems in Geldermans and Gillman [2019] to three dimensions.



Method	N	Runtime (s)	% of Peak FLOPS
No recomputation	4,194,304	2.58	1.56%
No recomputation	16,777,216	12.99	2.30%
No recomputation	67,108,864	54.99	4.17%
Leaf recomputation	4,194,304	0.63	6.45%
Leaf recomputation	16,777,216	9.43	3.27%
Leaf recomputation	67,108,864	41.18	5.66%
Subtree recomputation (Ours)	4,194,304	0.63	6.45%
Subtree recomputation (Ours)	16,777,216	4.72	12.65%
Subtree recomputation (Ours)	67,108,864	20.16	17.29%

Figure 4.3: Even with a naïve implementation of the HPS algorithm which does not perform any recomputation, using a single GPU achieves large speedups over a multicore CPU system. When we use our proposed subtree recomputation strategy, the speedup increases by another factor of two. (Left) We generate different problem sizes by holding $p = 16$ fixed and varying $L = 4, \ldots, 9$. We measure the total runtime of our 2D method merging DtN matrices. Vertical error bars show $\pm$ 1 standard error computed over five trials. (Right) For each of the GPU implementations, we compute the total number of FLOPS and report this as a percentage of the GPU's peak FLOPS, estimated by the manufacturer to be 34×10^{12}.

This method adaptively refines element sizes in a data-dependent manner, in an effort to concentrate discretization points in the regions of the domain where the coefficient and source functions have high local variation. In Table 4.1, we show that the size of $\boldsymbol{D}$ generated using our adaptive refinement technique is much smaller than that of the uniform refinement with no loss in accuracy.

Developing a version of the HPS methods presented in Section 4.3 that is compatible with an adaptive discretization requires slight modification of the algorithms presented in the previous section. The major changes are the introduction of a method for adaptively refining our octree and a method for merging nodes with different levels of refinement.

Method	p	Relative ℓ_∞ Error	# Leaves	Size of $\boldsymbol{D}$
Uniform	8	1.48×10^{-4}	512	6,912
Adaptive	8	1.45×10^{-4}	190	2,700
Uniform	12	3.62×10^{-7}	512	19,200
Adaptive	12	2.04×10^{-7}	442	7,500
Uniform	16	4.20×10^{-6}	64	9,408
Adaptive	16	1.41×10^{-6}	57	4,116

Table 4.1: Adaptive discretization methods can greatly reduce the peak memory requirements of HPS methods in three dimensions. We present the size of the discretization tree and final merge steps in our 3D "wavefront" example (Section 4.6.1). We compare adaptive and uniform discretizations that have similar errors and observe the adaptive discretization strategy can greatly reduce the size of the final $\boldsymbol{D}$ matrix, a proxy for peak memory usage.

Criterion for adaptive refinement

For a given leaf of the discretization tree, let $\boldsymbol{x}_0$ be the set of p^3 Chebyshev points discretizing the leaf. Let $\boldsymbol{x}_1$ be the set of $8p^3$ discretization points found by breaking the leaf into eight children and creating a Chebyshev grid on each child. Let $\boldsymbol{L}_{8f1}$ be an interpolation matrix mapping from $\boldsymbol{x}_0$ to $\boldsymbol{x}_1$. We evaluate whether a function is sufficiently refined on a leaf by checking whether we can use polynomial interpolation to accurately map from evaluations on $\boldsymbol{x}_0$ to evaluations on $\boldsymbol{x}_1$, relative to the global L_∞ norm of the function. We specify a tolerance parameter ϵ, and for each leaf in our tree, we check the following condition:

$$\frac{\|f(\boldsymbol{x}_1) - \boldsymbol{L}_{8f1} f(\boldsymbol{x}_0)\|_\infty}{\|f\|_\infty} < \epsilon. \tag{4.6}$$

If this condition is met, we say the leaf is sufficiently refined. Otherwise, we split the leaf into eight children and check each child. We form a final discretization tree by refining each coefficient function in our differential operator, as well as the source term, and taking the union of the resulting trees. Additionally, we refine a few extra leaves to enforce a "level restriction" criterion, which specifies that no leaf can have a side length greater than twice that of its neighbors. This method is an extension of the method for two-dimensional

problems presented in Geldermans and Gillman [2019], which uses a similar relative L_2 convergence criterion and level restriction criterion.

Local solve stage

The local solve stage for adaptively refined discretization trees is the same as in the uniform refinement case. Although they are defined over leaves with different volumes, each local boundary value problem has the same number of interior and boundary discretization points, and the local problems are still embarrassingly parallel. Thus, we can use batched linear algebra to accelerate this part of the algorithm.

Merging nodes with different discretization levels

The nonuniform merge stage is different from the uniform merge stage because neighboring nodes may have different refinement levels, which means the discretization points along either side of the merge interface may not exactly align. Recall the block linear system (Equation (4.3)) arising during the merge stage:

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{g}_{\text{ext}}^{(j)} \\ \boldsymbol{g}_{\text{int}}^{(j)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{u}_{\text{ext}}^{(j)} - \boldsymbol{h}_{\text{ext}}^{(\text{child})} \\ -\boldsymbol{h}_{\text{int}}^{(\text{child})} \end{bmatrix}.$$

When neighboring volume elements have different refinement levels, there will be a mismatch between the interior boundary discretization points on either side of the merge interface. We need to decide how to represent $\boldsymbol{g}_{\text{int}}^{(j)}$, $\boldsymbol{h}_{\text{int}}^{(\text{child})}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ in Equation (4.3). We choose to discretize these objects using the coarser of the two sets of discretization points along the merge interface; the discretization points along the exterior boundary elements are inherited from the child nodes. To assemble the blocks in Equation (4.3), this requires projecting some rows and columns of the Poincaré–Steklov matrices using precomputed interpolation operators which map between one and four 2D Gauss–Legendre panels. The "level restriction"

constraint greatly simplifies this compression because the resulting projection operations are guaranteed to be four-to-one.

Downward pass

To propagate the boundary information to the leaf nodes, we follow the general structure of Algorithm 5. However, we must undo the projection along merge interfaces that occurs during the nonuniform merge stage. This is accomplished by applying the precomputed interpolation operators to the boundary data $\boldsymbol{g}^{(i)}$.

4.5 Numerical examples in two dimensions

In this section, we present numerical results on problems with two spatial dimensions. All experiments in this section were conducted using one Nvidia H100 GPU and a host memory space with 100GB of RAM. In all of the experiments, we use the novel subtree recomputation strategy introduced in Section 4.4.1.

4.5.1 High-order convergence on variable-coefficient problems with known solutions

There are two main ways to increase the accuracy of our composite spectral colocation scheme: refine each leaf patch into four children or increase the polynomial order of the representation of the solution on each patch. Empirically, the error is controlled by the polynomial order p and the side length of each leaf h. In our implementation, p is specified by the user and h is controlled by L, the user-specified depth of the discretization tree. The tradeoff between these two parameters is a widely-studied topic in numerical analysis and goes by the name of "hp-adaptivity". We study the hp-adaptivity properties of our solver using two problems with variable-coefficient differential operators and known solutions. The

first problem is a variant of Poisson's equation with spatially-varying coefficients:

$$\begin{cases} \Delta u(x) - \cos(5x_2) u_{x_1}(x) + \sin(5x_2) u_{x_2}(x) = f(x), & x \in [-1,1]^2, \\ u(x) = g(x), & x \in \partial[-1,1]^2, \end{cases} \tag{4.7}$$

where u_{x_1} and u_{x_2} are the partial derivatives of u in directions x_1 and x_2, respectively. We manufacture the source f and the Dirichlet data g so the solution to this problem is

$$u(x) = u(x_1, x_2) = e^{5x_1} \sin(5x_2) + \sin(10\pi x_1) \sin(\pi x_2).$$

We also study an inhomogeneous Helmholtz problem with a Robin boundary condition:

$$\begin{cases} \Delta u(x) + (1 + e^{-50\|x\|^2}) u(x) = f(x), & x \in [-1,1]^2, \\ u_n(x) + iu(x) = g(x), & x \in \partial[-1,1]^2. \end{cases} \tag{4.8}$$

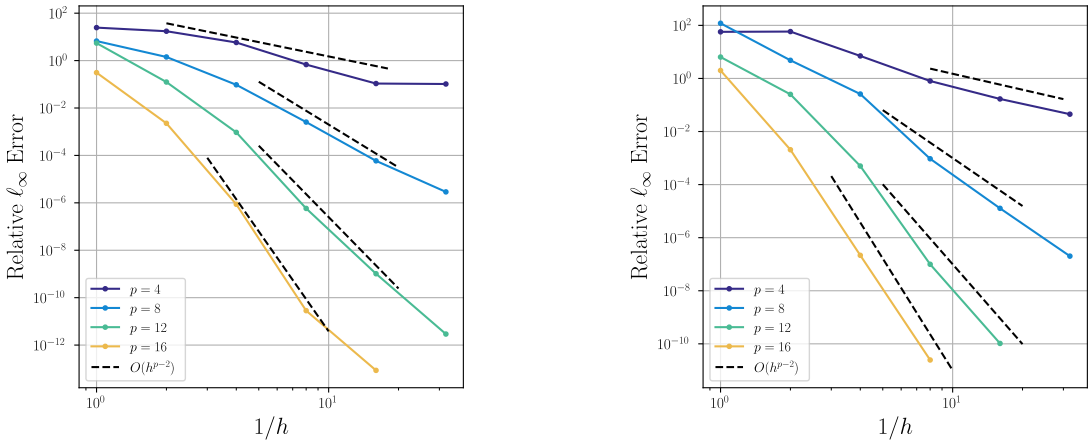
We manufacture the source f and the Robin data g so solution to this problem is

$$u(x) = u(x_1, x_2) = e^{i20x_1} + e^{i30x_2}.$$

Figure 4.4 shows the convergence of our method on these problems. We measure the relative error of our computed solution $\boldsymbol{u}$ by computing $\|\boldsymbol{u} - \boldsymbol{u}_{\text{true}}\|_\infty / \|\boldsymbol{u}_{\text{true}}\|_\infty$. The ℓ_∞ norms are estimated by taking the maximum over all interior discretization points. In Figure 4.4, we see the errors of both the DtN and ItI versions of the method converging at rate $O(h^{p-2})$, even for high polynomial orders.

4.5.2 High-frequency forward wave scattering problems

In this example, we solve a variable-coefficient Helmholtz equation coupled with a Sommerfeld radiation condition. The system is excited by a plane wave with direction $\hat{s} = [1, 0]^\top$



(a) DtN (b) ItI

Figure 4.4: Using p Chebyshev points per dimension on each leaf, and leaves of size h, the relative ℓ_∞ errors of our method converge at rate $O(h^{p-2})$. Figure 4.4a shows the convergence of the HPS method using DtN matrices applied to Equation (4.7) and Figure 4.4b shows the convergence of the HPS method using ItI matrices applied to Equation (4.8).

and frequency k:

$$\begin{cases} \Delta u(x) + k^2(1 + q(x))u(x) = -k^2 q(x) e^{ik\langle \hat{s}, x\rangle}, & x \in [-1, 1]^2, \\ \sqrt{r}\left(\dfrac{\partial u}{\partial r} - iku\right) \to 0, & r = \|x\|_2 \to \infty. \end{cases} \tag{4.9}$$

Equation (4.9) models time-harmonic wave scattering in many imaging modalities, such as sonar or radar imaging, geophysical sensing, and nondestructive testing of materials [Borges et al., 2017]. As such, forward wave scattering solvers are often used inside an inner loop of optimization routines for solving these inverse problems. These forward solvers must be highly optimized as they are evaluated hundreds or thousands of times over the course of an algorithm.

To solve Equation (4.9), we use the ItI variant of our implementation, and at the top level of the merge stage, we solve a boundary integral equation which enforces the Sommerfeld radiation condition [Gillman et al., 2015]. Discretizing the boundary integral equation requires a high-order Nyström method to generate single- and double-layer potential matri-

ces, which we perform in MATLAB using the `chunkIE` package [Askham et al., 2024]. For each discretization level and frequency, generating these matrices takes a few seconds on a standard laptop. Because these matrices can be precomputed once for a given discretization level and frequency, we do not include the time required to generate these matrices in our runtime measurements. The solution of this boundary integral equation specifies incoming impedance data, which is propagated down the tree to form the interior solution. While we solve Equation (4.9) for one source direction $\hat{s}$, this scheme can compute solutions for multiple different sources in parallel at the cost of a few extra matrix-vector multiplications.

In Figures 4.6 and 4.8, we measure the runtime and accuracy of our GPU-accelerated solver. Because analytical solutions for Equation (4.9) are unavailable for general scattering potentials $q(x)$, we measure error relative to an overrefined reference solution $\boldsymbol{u}_{\text{over}}$ with approximately 2,800 discretization points in both dimensions. For each computed solution $\boldsymbol{u}$, we compute the relative ℓ_∞ error $\|\boldsymbol{u} - \boldsymbol{u}_{\text{over}}\|_\infty / \|\boldsymbol{u}_{\text{over}}\|_\infty$. The ℓ_∞ norm is estimated by taking the maximum over a grid of 500×500 regularly-spaced grid points. We repeat this experiment for two different choices of the scattering potential $q(x)$. We first choose a single Gaussian bump, which loosely focuses the incoming wave:

$$q(x) = 1.5e^{-160\|x\|^2}.$$

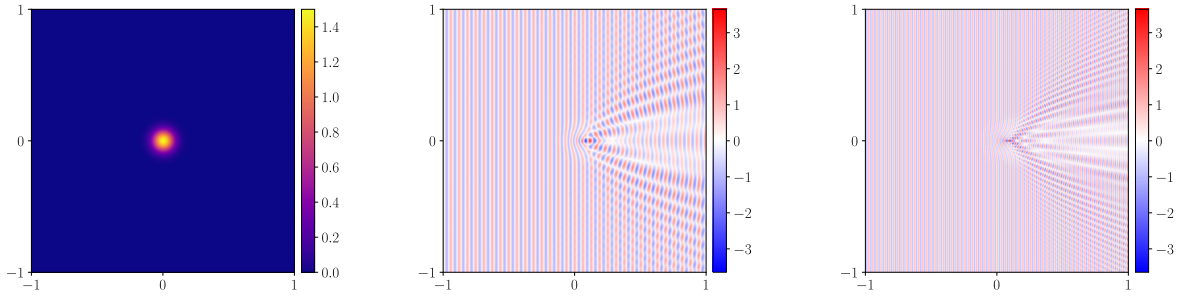
We also consider a collection of randomly placed Gaussian bumps with centers $\{z^{(i)}\}_{i=1}^{10}$, which causes multiple scattering effects at high wavenumbers:

$$q(x) = \sum_{i=1}^{10} e^{-50\|x - z^{(i)}\|^2}.$$

In Figures 4.5 and 4.7, we show these scattering potentials as well as the resulting total wave field $u(x) + e^{ik\langle \hat{s}, x\rangle}$ for different choices of k.

Our GPU-accelerated method with our proposed recomputation strategy is very fast. It

is able to compute high-accuracy solutions to these challenging scattering problems in a few seconds. We note that because this method does not rely on any iterative algorithms, the runtime for a fixed discretization level does not depend on the frequency k or the scattering potential $q(x)$. However, finer discretizations are required to achieve a fixed error tolerance as k increases. Figures 4.6 and 4.8 show that polynomial order is $p = 16$ dominates the lower-order methods for all values of k and scattering potentials considered.



(a) Scattering potential $q(x)$ (b) Total wave field; $k = 100$ (c) Total wave field; $k = 200$

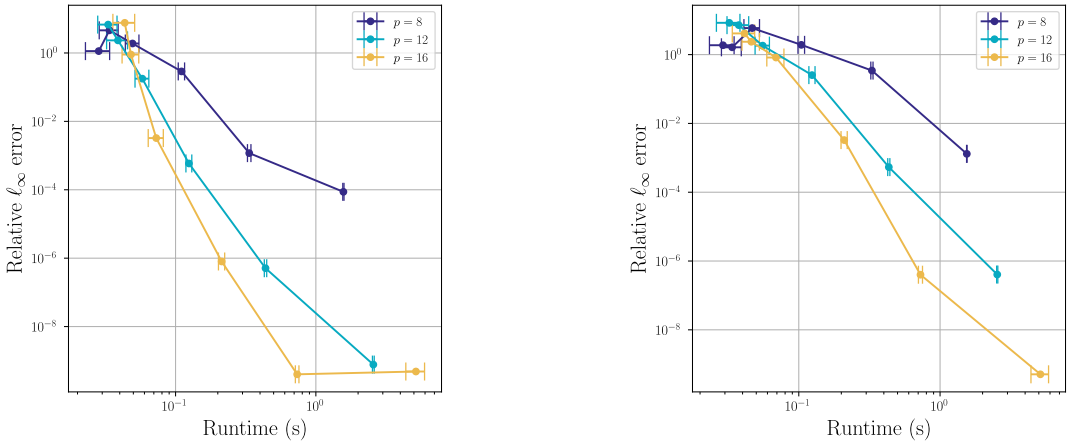
Figure 4.5: Visualizing the solutions of forward scattering problems for the single Gaussian bump scattering potential. Figures 4.5b and 4.5c show the real part of the total wave field.

4.5.3 Solving inverse scattering problems with automatic differentiation

Our solver is compatible with the JAX automatic differentiation framework, which allows us to very easily implement gradient-based optimization algorithms using our accelerated HPS solver as a forward model. We consider an inverse scattering task to recover the location of a compactly supported scattering potential with a known shape and amplitude, parameterized by $\theta = [\theta^{(1)}, \theta^{(2)}, \theta^{(3)}, \theta^{(4)}]^\top$, the centers of four Gaussian bumps:

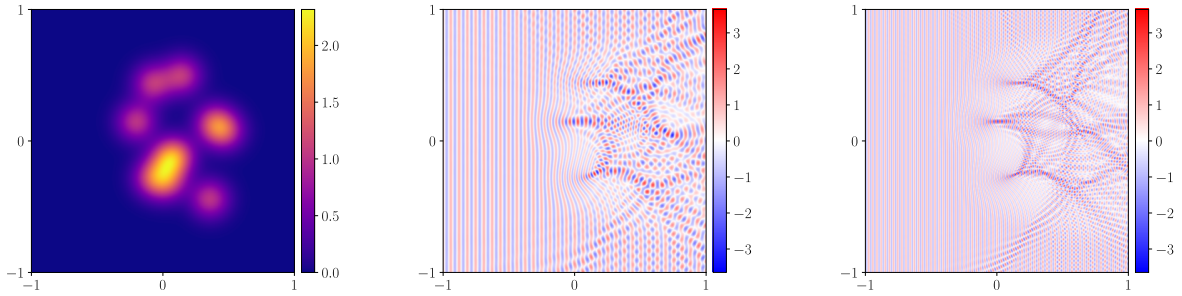
$$q_\theta(x) = 0.5 \sum_{j=1}^{4} e^{-\|x-\theta^{(j)}\|^2/(0.15)^2}. \tag{4.10}$$

The goal of this inverse scattering task is to estimate θ. Equation (4.9) describes how $q_\theta(x)$ affects the scattered wave field $u_\theta(x)$. Our forward model $\mathcal{F}$ maps $\theta \in \mathbb{R}^8$ to scattered wave



(a) Runtime and errors when $k = 100$ (b) Runtime and errors when $k = 200$

Figure 4.6: Error-runtime study on the single Gaussian bump scattering potential. Using GPU acceleration, our method can rapidly converge to high-accuracy solutions in high-frequency wave scattering problems. The runtime measurements include the runtime of the entire HPS algorithm and the setup and solution of the boundary integral equation enforcing the radiation condition. Horizontal error bars show ± 1 standard error computed over five trials.



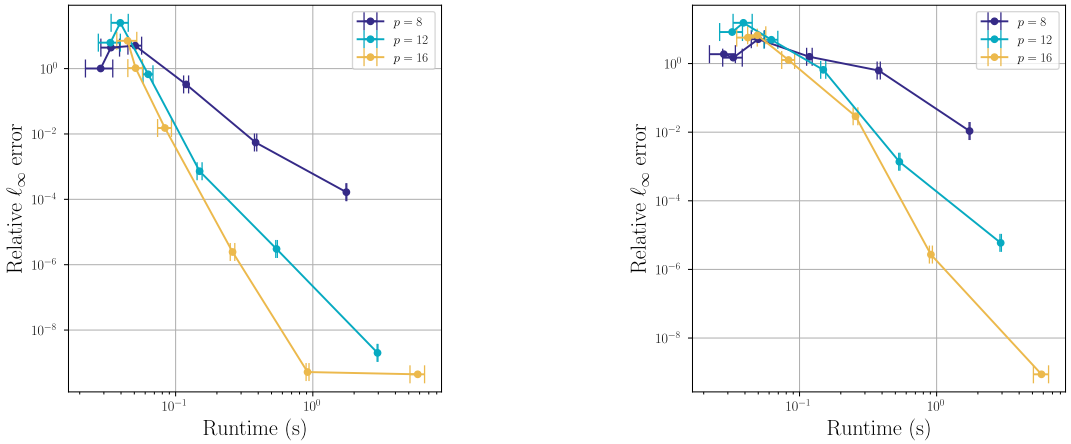
(a) Scattering potential $q(x)$ (b) Total wave field; $k = 100$ (c) Total wave field; $k = 200$

Figure 4.7: Visualizing the solutions of forward scattering problems for the sum of randomly placed Gaussian bumps scattering potential. Figures 4.7b and 4.7c show the real part of the total wave field.

field data evaluated at points $x^{(j)}$ away from the support of the scattering potential.

$$\mathcal{F}[\theta]_j = u_\theta\big(x^{(j)}\big) \tag{4.11}$$

We choose a ground-truth $\theta^* = [\theta^{(1)^*}, \theta^{(2)^*}, \theta^{(3)^*}, \theta^{(4)^*}]^\top$ and generate data $\mathcal{F}[\theta^*]$. Figure 4.9a shows the real part of the ground-truth wave field and the sample points used by



(a) Runtime and errors when $k = 100$

(b) Runtime and errors when $k = 200$

Figure 4.8: Error-runtime study on the sum of randomly placed Gaussian bumps scattering potential.

our forward model. The sample points can be arbitrary; we choose a subset of the HPS discretization points for convenience. Given this data and our knowledge of the forward model, we wish to recover an estimate of θ^*. We can phrase this problem in an optimization framework:

$$\underset{\theta \in [-0.5,0.5]^8}{\operatorname{argmin}} \|\mathcal{F}[\theta] - \mathcal{F}[\theta^*]\|_2^2. \tag{4.12}$$

We evaluate $\mathcal{F}$ using our GPU-accelerated fast direct solver. Because our solver is compatible with automatic differentiation, we can solve this optimization problem using gradient-based methods. In particular, we use Gauss–Newton iterations for nonlinear least squares problems. This algorithm requires access to the matrix $J[\theta]$, which is the Jacobian matrix of $\mathcal{F}$ evaluated at θ. Because the dense Jacobian is expensive to compute and represent, JAX instead exposes subroutines to automatically compute vector-Jacobian products $v^\top J[\theta]$ and Jacobian-vector products $J[\theta]v$ for arbitrary vectors v and estimates θ without requiring the user to compute any partial derivatives by hand. We pair these subroutines with a sparse linear algebra least-squares solver [Paige and Saunders, 1982a] from the SciPy library [Virtanen et al., 2020] to

implement the Gauss–Newton algorithm presented in Algorithm 8.

Algorithm 8: Gauss–Newton iterations for nonlinear least squares problems

Input: Data $\mathcal{F}[\theta^*]$; Initial estimate θ_0; $v^\top J[\theta]$ subroutine; $J[\theta]v$ subroutine

$t \leftarrow 0$

while not converged **do**

  Compose automatic differentiation and our fast direct solver to compile the function $v \mapsto v^\top J[\theta_t]$

  Compose automatic differentiation and our fast direct solver to compile the function $v \mapsto J[\theta_t]v$

  Define a linear operator $J[\theta_t]$ using subroutines $v^\top J[\theta_t]$ and $J[\theta_t]v$

  Use a least-squares solver to compute $\delta \leftarrow \underset{\delta}{\operatorname{argmin}} \ \|\mathcal{F}[\theta^*] - (\mathcal{F}[\theta_t] + J[\theta_t]\delta)\|_2^2$

  $\theta_{t+1} \leftarrow \theta_t + \delta$

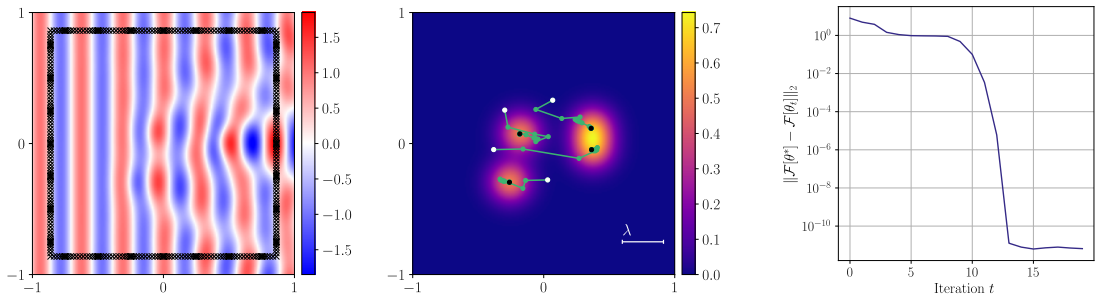
  $t \leftarrow t + 1$

Result: Final estimate θ_t

We randomly initialize the optimization at $\theta_0 \in [-0.5, 0.5]^8$ and run 20 iterations of the Gauss–Newton algorithm. Figure 4.9 shows the optimization variables take some iterations to approach θ^* and then converge superlinearly to machine precision. Calculating each Gauss–Newton update is fast because of the low-dimensional parameterization and GPU acceleration of the forward model, $J[\theta_t]v$, and $v^\top J[\theta_t]$. The entire experiment runs in 75 seconds using one H100 GPU.

4.6 Numerical examples in three dimensions

In the three-dimensional examples, we focus on problems with localized regions of high variation. Our adaptive discretization method described in Section 4.4 is designed for such problems. In these experiments, we use a single Nvidia H100 GPU with 80GB of on-device memory and 200GB of host RAM.



(a) (b) (c)

Figure 4.9: Our GPU-accelerated PDE solver can interface with automatic differentiation to rapidly solve inverse problems. In Figure 4.9a, we show the real part of the ground-truth total wave field and the sample locations (in black) used by our forward model $\mathcal{F}$. In Figure 4.9b, we show the ground-truth scattering potential $q_{\theta^*}(x)$ and the iterates (in green) taken by the Gauss–Newton algorithm. The initial estimate θ_0 is shown in white, and the final estimate θ_{19} is shown in black. We also show the free-space wavelength of the incident plane wave for scale. Figure 4.9c shows the objective value of the optimization problem, which reaches machine precision in 15 iterations.

4.6.1 Adaptive refinement on a problem with known solution

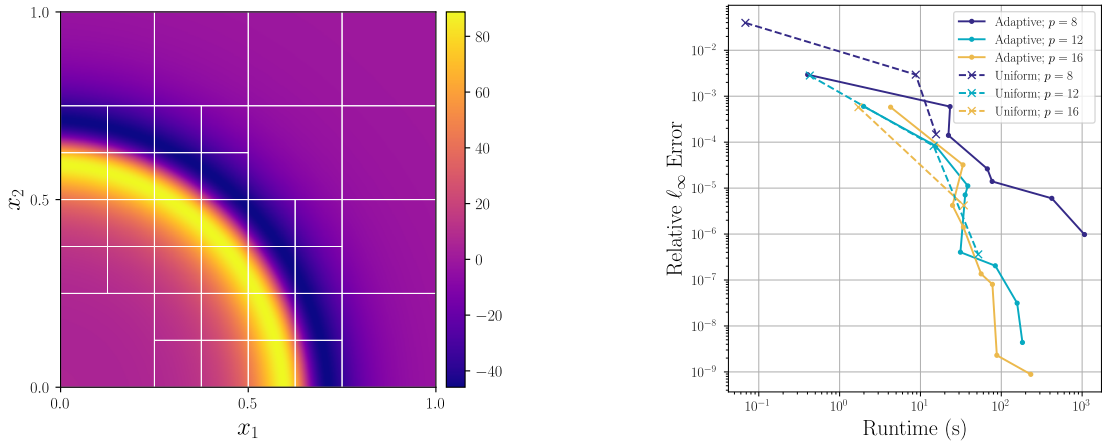
In this example, we study the convergence of our method on a three-dimensional problem with a known analytical solution. We build a problem that is solved by a "wavefront" located along a three-dimensional curved surface. The problem is given by

$$\begin{cases} \Delta u(x) = f(x), & x \in [0,1]^3, \\ u(x) = g(x), & x \in \partial[0,1]^3. \end{cases} \tag{4.13}$$

We manufacture a source term $f(x)$ so the solution takes the form

$$u(x) = u(x_1, x_2, x_3) = \arctan\left(10\sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2 + (x_3 - 0.5)^2} - 0.7\right) \tag{4.14}$$

and use samples of this function along the boundary to create our boundary data g. Figure 4.10a shows that f has a localized region of high variation. A uniform discretization strategy cannot adapt to the locality of this problem, but our adaptive discretization strat-



(a) (b)

Figure 4.10: Adaptive refinement allows for more accuracy before encountering memory bottlenecks. Figure 4.10a shows the source function restricted to the $x_3 = 0$ plane and the adaptive mesh formed with tolerance 1×10^{-7} and Chebyshev parameter $p = 16$. In Figure 4.10b, we study the runtime and error of different refinement strategies applied to Equation (4.13). For each step of the uniform refinement curve, we refined the uniform grid by one more level. For the adaptive refinement curve, we decreased the adaptive refinement tolerance by a factor of 10. For all methods, we refined until running out of memory on the GPU during the build stage.

egy can adaptively refine the octree to place a higher density of discretization points in this region. The adaptive discretization also uses larger leaves, where possible, on the parts of the domain with a very smooth source function.

In Figure 4.10b, we show accuracy versus runtime for both a uniform and adaptive discretization applied to this problem. For all methods, we increased the number of discretization points until saturating the GPU's memory limit when inverting the highest-level $\boldsymbol{D}$ matrix. While the uniform methods are fast on this problem, they are not highly accurate because they cannot use more than $L = 3$ levels of uniform refinement. Our adaptive method computes solutions with much higher accuracy before saturating the GPU's memory limit by adaptively placing the discretization points in regions where the source and solution have high variation. Table 4.1 shows the size of the highest-level merge matrix $\boldsymbol{D}$ for selected points on this graph.

4.6.2 Linearized Poisson–Boltzmann equation

An example application where the data and solution have local regions of high variation is the linearized Poisson–Boltzmann equation, a model of the electrostatic properties of a molecule in a solution. This can, for example, be used to compute the stability of a given molecular configuration in a solution. A standard model, developed in Grant et al. [2001], starts with atoms represented by point charges $\{z^{(i)}\}_{i=1}^{N_z}, z^{(i)} \in \mathbb{R}^3$. These atoms give rise to a charge distribution $\rho(x)$,

$$\rho(x) = \sum_{i=1}^{N_z} e^{-\delta \|x - z^{(i)}\|^2}, \tag{4.15}$$

and a spatially-varying permittivity function $\varepsilon(x)$,

$$\varepsilon(x) = \epsilon_0 + (\epsilon_\infty - \epsilon_0) e^{-A\rho(x)}. \tag{4.16}$$

We use parameters $N_z = 50$, $\delta = 45$, $\epsilon_0 = 16$, $\epsilon_\infty = 100$, and $A = 10$ [Grant et al., 2001]. The atomic centers $\{z^{(i)}\}_{i=1}^{N_z}$ are drawn uniformly from the box $[-0.5, 0.5]^3$. We can now model the electrostatic potential $u(x)$ which is implicitly defined by the linearized Poisson–Boltzmann equation:

$$\begin{cases} \nabla \cdot (\varepsilon(x) \cdot \nabla u(x)) = -\rho(x), & x \in [-1, 1]^3, \\ u(x) = 0, & x \in \partial[-1, 1]^3. \end{cases} \tag{4.17}$$

Existing approaches, such as a fast integral equation method [Vico et al., 2016] and finite difference schemes [Nicholls and Honig, 1991, Colmenares et al., 2014] solve a simplified version of Equation (4.17), where the permittivity function is replaced by one derived from

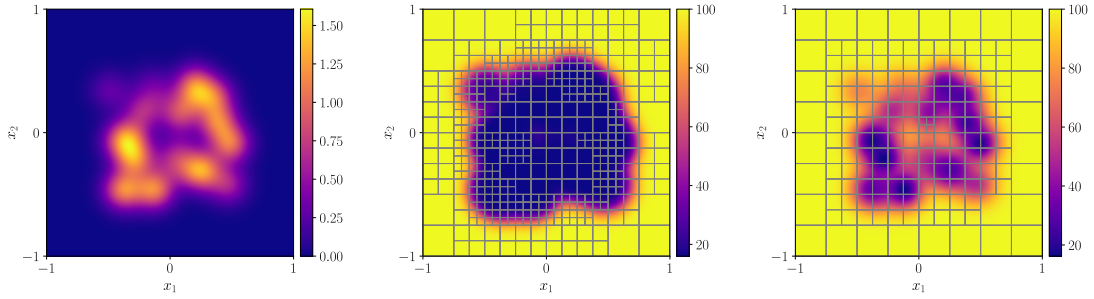
van der Waals surfaces:

$$\varepsilon_{\text{vdW}}(x) = q(x)\epsilon_0 + (1 - q(x))(\epsilon_\infty - \epsilon_0), \tag{4.18}$$

$$q(x) = 1 - \prod_{i=1}^{N_z} \left[1 - e^{-\delta \|x - z^{(i)}\|^2}\right]. \tag{4.19}$$

$\varepsilon_{\text{vdW}}(x)$ is an easier function to resolve to high accuracy as it lacks the steep gradients observed in $\varepsilon(x)$. However, Grant et al. [2001] reports "experimentation with a number of dielectric mapping functions using $[\varepsilon_{\text{vdW}}]$ produced dielectric functions that increase toward solvent values far too rapidly with distance from atomic centers," and "$[\varepsilon_{\text{vdW}}]$ also produced undesired patches of high dielectric inside proteins." We use our GPU-accelerated adaptive HPS method to solve Equation (4.17) with both permittivity models.

To form an adaptive discretization for this problem, we refine a discretization tree given the charge distribution ρ, the permittivity ε, and the components of $\nabla\varepsilon$. Figure 4.11 shows a 2D slice of the charge distribution and permittivity, along with the discretization found by this refinement process. Table 4.2 gives statistics about the discretization and runtime for a range of tolerances and Chebyshev parameters p.



(a) Source $\rho(x)$ (b) Permittivity $\varepsilon(x)$ (c) Permittivity $\varepsilon_{\text{vdW}}(x)$

Figure 4.11: Visualizing the variable coefficients and source term of our problem. These plots show the source function $\rho(x)$ and permittivity functions $\varepsilon(x)$ and $\varepsilon_{\text{vdW}}(x)$ restricted to the plane $x_3 = 0$. The adaptive discretizations formed using $p = 8$ and tolerance 1×10^{-4} are shown. This adaptive discretization is found by forming the union of meshes adaptively refined on the source, the permittivity, and components of the gradient of the permittivity.

p	Tolerance	n_{leaves}	N	Max Depth	Runtime (s)
8	10^{-1}	358	183,296	3	48.4
8	10^{-2}	1,436	735,232	4	177.8
8	10^{-3}	1,884	964,608	5	218.9
8	10^{-4}	7,659	3,921,408	5	1,523.0
10	10^{-1}	253	253,000	3	52.5
10	10^{-2}	449	449,000	4	76.3
10	10^{-3}	1,625	1,625,000	4	243.4
10	10^{-4}	1,982	1,982,000	5	319.1
12	10^{-1}	99	171,072	3	41.5
12	10^{-2}	386	667,008	3	91.0
12	10^{-3}	715	1,235,520	4	190.6
12	10^{-4}	1,695	2,928,960	4	OOM
16	10^{-1}	64	262,144	2	53.9
16	10^{-2}	204	835,584	3	136.2
16	10^{-3}	365	1,495,040	3	OOM
16	10^{-4}	470	1,925,120	4	OOM

p	Tolerance	n_{leaves}	N	Max Depth	Runtime (s)
8	1×10^{-3}	1,093	559,616	4	154.4
8	1×10^{-4}	1,737	889,344	5	227.2
8	1×10^{-5}	5,111	2,616,832	5	869.0
8	1×10^{-6}	9,423	4,824,576	5	OOM
10	1×10^{-3}	435	435,000	4	94.5
10	1×10^{-4}	1,016	1,016,000	4	193.3
10	1×10^{-5}	1,485	1,485,000	4	241.7
10	1×10^{-6}	2,605	2,605,000	5	483.0
12	1×10^{-3}	274	473,472	3	93.7
12	1×10^{-4}	477	824,256	4	145.8
12	1×10^{-5}	974	1,683,072	4	275.2
12	1×10^{-6}	1,366	2,360,448	4	OOM
16	1×10^{-3}	127	520,192	3	119.7
16	1×10^{-4}	239	978,944	3	172.2
16	1×10^{-5}	323	1,323,008	3	227.0
16	1×10^{-6}	456	1,867,776	4	OOM

Table 4.2: Resource usage statistics for using our 3D adaptive HPS method applied to the linearized Poisson–Boltzmann equation (Equation (4.17)) using permittivity $\varepsilon(x)$ (left) and simplified permittivity $\varepsilon_{\text{vdW}}(x)$ (right). We use "OOM" to indicate which discretizations caused out of memory errors when invering the final $\boldsymbol{D}$ matrix.

4.7 Conclusion

This chapter presents methods for efficiently accelerating HPS algorithms using general-purpose GPUs. Because there is a large amount of inherent parallelism in the structure of the HPS algorithms, they are a natural target for GPU acceleration once adjustments are made to reduce memory complexity. We introduce methods for reducing the memory footprint of HPS algorithms for problems in two and three dimensions.

This work leaves open important questions and avenues for improvement. While our method can efficiently interface with automatic differentiation, we could, in principle, gain much more efficiency by implementing custom automatic differentiation rules to reuse the precomputed solution operators, like those derived in Borges et al. [2017]. Our methods of reducing memory complexity could be pushed further by using a hybrid approach, i.e., by performing a few levels of merging via dense linear algebra and then relying on a sparse direct solver such as Yesypenko and Martinsson [2024b] or Kump et al. [2025] for higher-level merges. We hypothesize such a hybrid approach would greatly reduce runtimes for very large problems by reducing the ranks needed to accurately resolve the sparse system matrix. Finally, our methods could be extended to unstructured meshes and surfaces, a setting where

nonuniform merge operations make parallel implementations challenging.

Acknowledgment

The authors would like to thank Manas Rachh, Leslie Greengard, and Olivia Tsang for many useful discussions. The authors are grateful to the Flatiron Institute for providing the computational resources used to conduct the experiments in this work. OM and RW gratefully acknowledge the support of NSF DMS-2023109, DOE DE-SC0022232, the Physics Frontier Center for Living Systems funded by the National Science Foundation (PHY-2317138), and the support of the Margot and Tom Pritzker Foundation. The Flatiron Institute is a division of the Simons Foundation.

CHAPTER 5

JAXHPS: AN ELLIPTIC PDE SOLVER BUILT WITH MACHINE LEARNING IN MIND

This is a short manuscript under review at the Journal of Open Source Software, which accepts short papers that accompany open-source research software repositories. The manuscript is accompanied by our software repository and documentation.

5.1 Summary

Elliptic partial differential equations (PDEs) can model many physical phenomena, such as electrostatics, acoustics, wave propagation, and diffusion. In scientific machine learning settings, a high-throughput PDE solver may be required to generate a training dataset, run in the inner loop of an iterative algorithm, or interface directly with a deep neural network. To provide value to machine learning users, such a PDE solver must be compatible with standard automatic differentiation frameworks, scale efficiently when run on graphics processing units (GPUs), and maintain high accuracy for a large range of input parameters. We design the `jaxhps` package with these use-cases in mind by implementing a highly efficient and accurate solver for elliptic problems with native hardware acceleration and automatic differentiation support. This is achieved by expressing a highly-efficient solution method for elliptic PDEs in JAX [Bradbury et al., 2018]. This software implements algorithms specifically designed for fast GPU execution of a family of elliptic PDE solvers, which are described in full in Melia et al. [2025a].

Our Python package can numerically compute solutions $u(x)$ to problems of the form:

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

$$u(x) = g(x), \qquad x \in \partial\Omega. \tag{5.2}$$

In our setting, $\mathcal{L}$ is a linear, elliptic, second-order partial differential operator with spatially varying coefficient functions. The spatial domain, Ω, can be a 2D square or 3D cube.

5.2 Statement of need

While there is a vast array of PDE solvers implemented in JAX, we make a distinct contribution by implementing methods from the hierarchical Poincaré–Steklov (HPS) family of algorithms [Martinsson, 2013, Gillman and Martinsson, 2014, Gillman et al., 2015]. These methods use modern numerical analysis tools to resolve physical phenomena that are challenging for simpler tools, such as finite difference or finite element methods. One example of such a physical phenomenon is the oscillatory behavior of time-harmonic wave propagation simulations, which HPS methods resolve accurately and finite element methods do not [Babuška and Sauter, 1997, Yesypenko and Martinsson, 2024b].

While open-source implementations of HPS methods exist for users of MATLAB [Fortunato, 2024] and C++ [Chipman, 2024], these packages do not offer native hardware acceleration or automatic differentiation capabilities. In addition, these packages do not offer support for three-dimensional problems. Yesypenko [2024] is a Python implementation of the hardware-accelerated HPS-like method described in Yesypenko and Martinsson [2024b], but it is designed for performance on extremely large 2D systems, which requires different design choices than the machine learning-focused optimizations we include in `jaxhps`.

5.3 Software overview

The software is designed with two goals:

- Allow users to interact with a simple interface that abstracts the complex HPS algorithms.
- Organize the flow of data to allow the user to reuse computations where possible.

A typical user of `jaxhps` will wish compute a solution $u(x)$ to Equations (5.1) and (5.2). The user will first specify Ω by creating `DiscretizationNode` and `Domain` objects. These objects automatically compute a high-order composite spectral discretization of Ω. The `Domain` class exposes utilities for interpolating between the composite spectral discretization and a regular discretization specified by the user. If f or $\mathcal{L}$ have local regions of high curvature, the `Domain` object's discretization can also be computed in an adaptive way that assigns more discretization points to those parts of Ω. Additional utilities for interacting with the composite spectral discretization can be found in the `quadrature` module.

After the `Domain` is initialized, a `PDEProblem` object is created by the user from data specifying $\mathcal{L}$ and (optionally) f. The `PDEProblem` object also stores pre-computed interpolation and differentiation operators that can be reused during repeated calls to the solver.

The user can then execute the HPS algorithm by calling the `build-solver` method, specifying f and g, and finally calling the `solve` method. During the `build-solver` and `solve` methods, pointers to various solution operators are stored in the `PDEProblem` object. If the problem size is large, and these solution operators can not all be stored simultaneously on a GPU, care must be taken to organize the computation and data transfer between the GPU and CPU memory. To facilitate this, we provide `solve-subtree`, `upward-pass-subtree`, and `downward-pass-subtree`, newly developed algorithms designed to minimize data transfer costs. A full description of these algorithms can be found in [Melia et al., 2025a]. After computing the solution, the user can use automatic differentiation to compute the gradient of the solution with respect to input parameters by calling `jax.jvp` or `jax.vjp`. Multiple examples showing these capabilities are included in the source repository and the documentation.

Finally, some researchers may want to design new algorithms by operating on the outputs of various subroutines underlying these HPS methods. To facilitate this, we expose a collection of these subroutines in the `local-solve`, `merge`, `up-pass`, and `down-pass` modules.

CHAPTER 6

OPTIMIZATION AND REGULARIZATION FOR PDE-BASED INVERSE PROBLEMS WITH FAST DIRECT SOLVERS

6.1 Introduction

Linear elliptic partial differential equations (PDEs) can model a broad range of physical phenomena, including wave propagation, electrostatics, and diffusion. Many common imaging modalities can be described by linear elliptic PDEs; examples include ultrasound imaging with time-harmonic measurements, electrical impedance tomography, and optical tomography. In these imaging modalities, the image is probed by some source wave, charge, or magnetic field. The probe is modulated by a spatially varying medium, such as a spatially varying wave speed or photon absorption rate. The goal of these imaging modalities are to reconstruct an image of the spatially varying medium from data provided by a set of sensors. Finally, the measured data is used to reconstruct the image. This chapter is concerned with this final step in the case when the forward model is defined by a linear elliptic PDE.

Image reconstruction algorithms in this setting face multiple challenges. The most apparent difficulty is the computational expense required to evaluate the forward model. Because the PDE forward model does not define an explicit mapping, any reconstruction algorithm must solve a new PDE for each estimated image. Another source of difficulty is the nonlinearity of the forward model; although the PDEs considered in this chapter are linear, the mapping from image to measurements is, in general, nonlinear. The nonlinearity leads to nonconvex optimization formulations of the inverse problem, which can create many challenges for reconstruction algorithms, including spurious local minima. In practice, these nonconvex optimization formulations are often poorly conditioned, which limits the rate of progress we can expect from iterates of the optimization algorithm and can render reconstructions sensitive to measurement noise. In this chapter, we design reconstruction algorithms that

alleviate these problems by combining efficient PDE solvers with optimization algorithms designed for stable image recovery.

In particular, we focus on regularized image reconstruction algorithms which minimize the sum of two terms, a data-fit functional and a regularization function g:

$$\underset{q}{\operatorname{argmin}} \ \frac{1}{2}\|\mathcal{F}[q] - y\|_2^2 + \lambda g(q). \tag{6.1}$$

In Equation (6.1), $\mathcal{F}$ is the forward model implicitly defined by a PDE, y is the measured data, and $\lambda > 0$ is a regularization parameter. The regularization term allows us to inject prior information about the image being reconstructed, such as a small ℓ_2 norm, or a preference toward low frequency content. The regularization term can also improve the quality of image reconstruction by reducing sensitivity to noise in the measurements y. While there are many potential benefits to including the regularization term, this new term can be difficult to handle in commonly-used algorithms for PDE-based inverse problems, as we discuss in Section 6.3.

We consider two model problems in this chapter, inverse medium scattering and continuous-wave diffuse optical tomography. Both problems have broad applications in imaging sciences; they have been used to model medical imaging [Boas et al., 2001], seismic imaging [Fichtner, 2011], and nondestructive testing [Kumar and Arnold, 2022]. While uniqueness and stability results are known for the inverse scattering problem [Colton and Kress, 2013, Hähner and Hohage, 2001] and diffuse optical tomography inverse problems [Meftahi, 2021], the quantitative recovery of images using these modalities is often challenging. The two forward models exhibit qualitatively different behavior. Inverse medium scattering problems model wave-type behavior, which leads to highly oscillatory optimization problems with many spurious local minima [Bao and Liu, 2003]. Continuous-wave diffuse optical tomography exhibits diffusion and absorption phenomena, resulting in a forward model with a nontrivial kernel containing high-frequency parts of the image. This often causes

reconstruction algorithms to output over-smoothed image estimates [Konecky et al., 2008, Okawa and Hoshi, 2023]. The inclusion of the regularization term in Equation (6.1) can potentially mitigate these problems by changing the optimization landscape or injecting *a priori* knowledge about the imaging targets.

In addition to stable recovery, we are particularly interested in decreasing the runtime of the reconstruction algorithms considered, as many imaging applications require fast reconstructions. We are able to accurately reconstruct images much faster than previous methods through algorithmic innovation and the use of hardware acceleration using general-purpose graphics processing units (GPUs). We leverage recent advances in direct solvers for elliptic PDEs using GPU acceleration ([Melia et al., 2025a] and Chapter 5) to greatly reduce the per-iteration time cost of reconstruction algorithms in this setting.

Section 6.2 introduces the necessary background and notation. Section 6.3 reviews optimization approaches with and without regularization for nonlinear inverse problems. Section 6.4 introduces the inverse medium scattering problem, prior work on this problem, and gives experimental results. Section 6.5 introduces the diffuse optical tomography problem and discusses related work. Section 6.6 concludes.

6.2 Background

This section introduces the technical background necessary to discuss the inverse problems and algorithms considered in this chapter. Throughout this chapter, we use x for the spatial variable, and when we want to indicate Cartesian coordinates, we use $(x_1, x_2) \in \mathbb{R}^2$. We use the subscript u_n to denote the outward-pointing boundary normal derivative of a function, and we use Δ to denote the Laplace operator, the sum of second derivatives in each dimension. We use A^* to denote the adjoint of a complex-valued matrix A: $A^*_{i,j} = \overline{A}_{j,i}$. In this section, we sometimes consider functions $q : \Omega \to \mathbb{R}$ in an infinite-dimensional vector space. These functions are often the spatially varying media that we hope to recover. In practice, we

recover a finite-dimensional approximation of q, sampled on a $d \times d$ regular discretization of some domain Ω. To distinguish between the two, we use bold symbols to denote finite dimensional vectors; $\boldsymbol{q} \in \mathbb{R}^{d^2}$ represents the discretization of q. We use $\text{mat}(\boldsymbol{q})$ to denote the rearrangement of this vector into a square image $\in \mathbb{R}^{d \times d}$.

6.2.1 Inverse Problem and Optimization Formulation

We specify a set of n measurement points $\{x^{(i)}\}_{i=1,\ldots,n}$ and define the forward model $\mathcal{F}$:

$$\mathcal{F}[q]_i = u(x^{(i)}), \qquad u \text{ solves Equations (6.6) and (6.7).} \tag{6.2}$$

The function u is the solution to a PDE parameterized by q; we discuss the class of PDEs considered and the relationship between q and u in Section 6.2.2. The measured data is the evaluation of $\mathcal{F}$ at the ground-truth image q_*, with additive zero-mean noise:

$$\boldsymbol{y} = \mathcal{F}[q_*] + \xi. \tag{6.3}$$

The exact specification of the noise model will depend on the application. Given $\boldsymbol{y}$ and knowledge of $\mathcal{F}$, our goal is to form an estimate $\hat{\boldsymbol{q}} \approx \boldsymbol{q}_*$. We formulate this as a regularized optimization problem:

$$\hat{\boldsymbol{q}} = \underset{\boldsymbol{q} \in \mathbb{R}^{d^2}}{\text{argmin}} \underbrace{\frac{1}{2}\|\mathcal{F}[\boldsymbol{q}] - \boldsymbol{y}\|_2^2}_{:= f(\boldsymbol{q})} + \lambda g(\boldsymbol{q}). \tag{6.4}$$

We will refer to the first term as the "data term" and the second term as the "regularization term". Our regularization function is the anisotropic total variation norm:

$$\|\boldsymbol{q}\|_{\text{TV}} = \sum_{i,j=1}^{d} \sqrt{\left(\text{mat}(\boldsymbol{q})_{i+1,j} - \text{mat}(\boldsymbol{q})_{i,j}\right)^2 + \left(\text{mat}(\boldsymbol{q})_{i,j+1} - \text{mat}(\boldsymbol{q})_{i,j}\right)^2}. \tag{6.5}$$

The total variation norm induces sparsity in the image gradient. This is a standard regularization used in image processing, which avoids over-smoothed estimates obtained using Tikhonov regularization.

6.2.2 Linear Elliptic Partial Differential Equations

We focus on solving linear, elliptic partial differential equations of the form

$$\mathcal{L}^{(q)}u(x) = f^{(q)}(x), \qquad x \in \Omega, \tag{6.6}$$

$$\mathcal{B}^{(q)}u(x) = g^{(q)}(x), \qquad x \in \partial\Omega. \tag{6.7}$$

In Equation (6.6), $\mathcal{L}^{(q)}$ is a linear, elliptic, second-order partial differential operator with spatially varying coefficient functions. $\mathcal{B}^{(q)}$ is a linear boundary operator, $f^{(q)}$ is the source data, and $g^{(q)}$ is the boundary data. The superscript notation is meant to indicate that all of these objects may depend on the image $q : \Omega \to \mathbb{R}$. Ω is a square $\subset \mathbb{R}^2$. For the problems considered in this chapter, the dependence of $\mathcal{L}^{(q)}$ and $f^{(q)}$ on q is simple and known in closed form. $\mathcal{B}^{(q)}$ and $g^{(q)}$ often depend on q in a complicated way, but given q we can numerically compute $\mathcal{B}^{(q)}$ and $g^{(q)}$.

6.2.3 GPU-Accelerated Direct PDE Solvers

Optimization algorithms for Equation (6.4) typically require the ability to evaluate $\mathcal{F}[\hat{\boldsymbol{q}}]$ for arbitrary estimates $\hat{\boldsymbol{q}}$, and access to $\nabla\mathcal{F}[\hat{\boldsymbol{q}}]$, the Jacobian matrix of $\mathcal{F}$ centered at $\hat{\boldsymbol{q}}$. Each of these primitives is evaluated in each iteration of a traditional optimization algorithm, so they must be fast and accurate.

We use the GPU-accelerated Hierarchical Poincaré–Steklov (HPS) method from Chapter 5 [Melia et al., 2025a]. This allows fast and accurate access to $\mathcal{F}[\hat{\boldsymbol{q}}]$ and $\nabla\mathcal{F}[\hat{\boldsymbol{q}}]$. At a high level, the Hierarchical Poincaré–Steklov method computes a solution u to Equations (6.6)

and (6.7) by:

- Forming a high-order composite spectral discretization by placing a quadtree on Ω with a user-specified depth and representing each leaf of the tree with a high-order spectral discretization.
- Solving a local problem on each leaf of the tree.
- Hierarchically merging patches to construct a global solution operator by enforcing continuity conditions across patch interfaces.
- Applying the solution operator to data $g^{(q)}$ and $f^{(q)}$ to rapidly compute the solution u.

The high-order spectral discretization and exact hierarchical merge steps afford this method a high level of accuracy for a fixed computational effort, especially in challenging wavelike equations. For the problems considered in this paper, all of these steps can be executed completely on the GPU with a high amount of parallelization, so computing $\mathcal{F}$ is very fast. Once u is computed on the high-order spectral discretization points, it can be interpolated to the measurement points $\{x^{(i)}\}_{i=1,\ldots,n}$. Jacobian-vector and vector-Jacobian products involving $\nabla\mathcal{F}[\hat{\boldsymbol{q}}]$ can be evaluated using standard automatic differentiation software; the cost of evaluating $\nabla\mathcal{F}[\hat{\boldsymbol{q}}]\boldsymbol{v}$ or $\boldsymbol{v}^*\nabla\mathcal{F}[\hat{\boldsymbol{q}}]$ for an arbitrary vector $\boldsymbol{v}$ is approximately equal to the cost of computing $\mathcal{F}[\hat{\boldsymbol{q}}]$. The application of the automatic differentiation software to this direct PDE solver allows us to implement and experiment with optimization algorithms without deriving adjoint equations by hand.

6.3 Optimization for Nonlinear Inverse Problems

We consider different iterative algorithms for optimizing Equation (6.4). Because we are interested in minimizing the runtime of these algorithms, we must consider two quantities:

the cost to compute each iterate, and the number of iterates required to reach convergence. The first algorithm we consider has a low cost per iterate but can only be expected to converge linearly to a fixed point [Beck and Teboulle, 2009]. The second algorithm has a much higher cost per iteration but can be expected to converge superlinearly [Lee et al., 2014].

The algorithms considered in this chapter are "proximal methods" which interact with the regularization term through its proximal operator:

$$\text{prox}_{\lambda g}(\boldsymbol{q}) = \underset{\boldsymbol{z}\in\mathbb{R}^{d^2}}{\text{argmin}} \ \frac{1}{2}\|\boldsymbol{z}-\boldsymbol{q}\|_2^2 + \lambda g(\boldsymbol{z}). \tag{6.8}$$

In iterative algorithms, proximal operators are often used to apply regularization in an iterative algorithm. Suppose an estimate q_t approximately minimizes the data term in Equation (6.4). Intuitively, we would want a reconstruction algorithm to find an estimate near $\boldsymbol{q}_t$ which also approximately minimizes the regularizer $g(\cdot)$. Applying $\text{prox}_{\lambda g}$ to $\boldsymbol{q}_t$ gives us a new estimate $\boldsymbol{z}_t = \text{prox}_{\lambda g}(\boldsymbol{q}_t)$ which satisfies these desiderata. The algorithms considered in this section work similarly: they alternate between steps which make progress on the data term in Equation (6.4), and steps which apply proximal operators to make progress on the regularization term.

Computing Equation (6.8) for some choices of g requires an iterative approach to approximately solve the minimization problem. To implement the proximal operator of the total variation norm, we use the Douglas-Rachford splitting method developed in Charisopoulos and Willett [2025]. The splitting method operates by viewing the total variation norm as the composition of a sparse linear operator and a group norm. The iterations alternate between projecting onto the graph of this sparse linear operator and projecting the groups onto the unit norm ball; both operations can be implemented very efficiently. Because $\text{prox}_{\lambda\|\cdot\|_{\text{TV}}}$ is efficient, the major computational cost of the algorithms considered comes from the evaluation of $\mathcal{F}[\boldsymbol{q}]$ and $\nabla\mathcal{F}[\boldsymbol{q}]$.

6.3.1 FISTA

The first algorithm we consider is a variant of the Fast Iterative Shrinking and Thresholding Algorithm (FISTA), introduced by Beck and Teboulle [2009]. The original FISTA method applied regularization using the proximal operator of the ℓ_1 norm; the variant we consider applies regularization using the proximal operator of the total variation norm. FISTA has been applied to many nonlinear inverse problems, such as inverse scattering problems [Liu et al., 2018, Kamilov et al., 2017], electrical impedance tomography [Nguyen Diep et al., 2025], and time-of-flight diffuse optical tomography [Zhao et al., 2021]. FISTA was initially designed to minimize Equation (6.4) when g is the ℓ_1 norm, but can be extended to other non-smooth regularization functions.

Our implementation of FISTA is outlined in Algorithm 9. The most computationally intensive step is Line 6, which requires one forward model evaluation to compute $\mathcal{F}[\boldsymbol{q}_{t-1}]$ and one vector-Jacobian product to compute the update direction. The forward model evaluation uses the GPU-accelerated HPS method described in Section 6.2; the vector-Jacobian product is computed using automatic differentiation software applied to the GPU-accelerated HPS code.

Algorithm 9: FISTA with Total Variation Regularization. (HPS + FISTA)

Input: Initial estimate $\boldsymbol{q}_0$; data $\boldsymbol{y}$; regularization strength λ; step size γ.

1 $\boldsymbol{s}_0 \leftarrow \boldsymbol{0}$
2 $\alpha_0 \leftarrow 0$
3 $t \leftarrow 0$
4 **while** Not converged **do**
5 $\quad t \leftarrow t + 1$
6 $\quad \boldsymbol{z}_t \leftarrow \boldsymbol{q}_{t-1} - \gamma \nabla \mathcal{F}[\boldsymbol{q}_{t-1}]^* \left(\mathcal{F}[\boldsymbol{q}_{t-1}] - \boldsymbol{y}\right)$ `// Main computational work`
7 $\quad \boldsymbol{s}_t \leftarrow \operatorname{prox}_{\lambda \|\cdot\|_{\mathrm{TV}}}(\boldsymbol{z}_t)$
8 $\quad \alpha_t \leftarrow \frac{1}{2}\left(1 + \sqrt{1 + 4\alpha_{t-1}^2}\right)$
9 $\quad \boldsymbol{q}_t \leftarrow \boldsymbol{s}_t + \left(\frac{\alpha_{t-1} - 1}{\alpha_t}\right)(\boldsymbol{s}_t - \boldsymbol{s}_{t-1})$

Result: Final estimate $\hat{\boldsymbol{q}} = \boldsymbol{q}_t$.

6.3.2 *Proximal Quasi-Newton Methods*

Quasi-Newton methods are a popular algorithm for PDE-based inverse problems; they have been successfully applied to inverse scattering problems [Borges et al., 2017, Borges and Biros, 2018] and diffuse optical tomography [Okawa and Hoshi, 2023]. These methods proceed by minimizing an approximate second-order model of the data term at each iteration. At iteration t, the second-order model function f_t is

$$f_t(\boldsymbol{q}) = f(\boldsymbol{q}_t) + \nabla f(\boldsymbol{q}_t)^*(\boldsymbol{q} - \boldsymbol{q}_t) + \frac{1}{2}(\boldsymbol{q} - \boldsymbol{q}_t)^* H_t(\boldsymbol{q} - \boldsymbol{q}_t). \tag{6.9}$$

H_t is the Hessian approximation for iteration t. The Gauss-Newton method specifies a particular Hessian approximation constructed from first-order function information:

$$H_t = \nabla f(\boldsymbol{q}_t)^* \nabla f(\boldsymbol{q}_t) = \nabla \mathcal{F}[\boldsymbol{q}_t]^* \nabla \mathcal{F}[\boldsymbol{q}_t]. \tag{6.10}$$

The Gauss-Newton method computes an iterate $\boldsymbol{q}_{t+1}$ by minimizing Equation (6.9). When combined with the definition of f from Equation (6.4), one can derive a closed-form expression for this update step:

$$\boldsymbol{q}_{t+1} = \left(\nabla \mathcal{F}[\boldsymbol{q}_t]^* \nabla \mathcal{F}[\boldsymbol{q}_t]\right)^{-1} \nabla \mathcal{F}[\boldsymbol{q}_t]^* \left(\mathcal{F}[\boldsymbol{q}_t] - \boldsymbol{y}\right)$$

Proximal quasi-Newton methods seek to add a regularization term to the second-order model:

$$h_t(\boldsymbol{q}) = f_t(\boldsymbol{q}) + \lambda g(\boldsymbol{q}) = f(\boldsymbol{q}_t) + \nabla f(\boldsymbol{q}_t)^*(\boldsymbol{q} - \boldsymbol{q}_t) + \frac{1}{2}(\boldsymbol{q} - \boldsymbol{q}_t)^* H_t(\boldsymbol{q} - \boldsymbol{q}_t) + \lambda g(\boldsymbol{q}). \tag{6.11}$$

Because of the regularization term, there is no closed-form expression for the minimizer of h_t, so operator splitting methods [Ryu and Yin, 2022] are employed to compute a minimizer.

Different splitting methods have been proposed, including a forward-backward envelope splitting [Tan et al., 2024] and a splitting which introduces a new "rescaled proximal operator" [Lee et al., 2014]. To avoid the exact Hessian evaluation required by the forward-backward envelope splitting and the poorly-conditioned rescaled proximal operator, we instead use a simple method to compute a minimizer of Equation (6.11) based on Douglas-Rachford splitting:

$$\boldsymbol{z}_{n+1} = \operatorname{prox}_{f_t}(\bar{\boldsymbol{z}}_n), \tag{6.12}$$

$$\bar{\boldsymbol{z}}_{n+1} = \bar{\boldsymbol{z}}_n + \operatorname{prox}_{\lambda g}(2\boldsymbol{z}_{n+1} - \bar{\boldsymbol{z}}_n) - \boldsymbol{z}_{n+1}, \tag{6.13}$$

which converges to the minimizer of Equation (6.9) as $n \to \infty$. Under this splitting, $\operatorname{prox}_{f_t}$ is the solution of a linear system with linear operator $I + H_t$. In particular, the solution $z_* = \operatorname{prox}_{f_t}(\boldsymbol{z})$ solves

$$(H_t + I)\, \boldsymbol{z}_* = \nabla \mathcal{F}[\boldsymbol{q}_t]^*(\boldsymbol{y} - \mathcal{F}[\boldsymbol{q}_t]) + \boldsymbol{z} + H_t \boldsymbol{q}_t \tag{6.14}$$

In our implementation, we use the Gauss-Newton Hessian approximation (Equation (6.10)), so the operator $I + H_t$ can be applied using one Jacobian-vector product and one vector-Jacobian product; we compute both using automatic differentiation. We are now ready to present our proximal Gauss-Newton method in Algorithm 10. In our implementation, Line 7 is the most computationally expensive step. The conjugate gradient method requires one vector-Jacobian and one Jacobian-vector product per iteration, and the number of iterations is controlled by the condition number of $H_t + I$. The splitting method we use is designed so that this condition number is relatively benign. However, this expensive iteration is inside an inner iteration, meaning it is evaluated many times per outer iteration of the algorithm.

Algorithm 10: Proximal Gauss-Newton with Total Variation Regularization. (HPS + GN)

Input: Initial estimate q_0; data y; regularization strength λ; step size γ.

$t \leftarrow 0$
while Not converged **do**
  $t \leftarrow t + 1$
  $\bar{\boldsymbol{z}}_0 \leftarrow \boldsymbol{q}_t$
  while Not converged **do**
    $n \leftarrow n + 1$
    Use the conjugate gradient method to compute $z_n = \text{prox}_{f_t}(\bar{\boldsymbol{z}}_{n-1})$
    $\bar{\boldsymbol{z}}_n \leftarrow \bar{\boldsymbol{z}}_{n-1} + \text{prox}_{\lambda\|\cdot\|_{\text{TV}}}(2\boldsymbol{z}_n - \bar{\boldsymbol{z}}_{n-1}) - z_n$
  $\boldsymbol{q}_t \leftarrow \boldsymbol{z}_n$

Result: Final estimate $\hat{\boldsymbol{q}} = \boldsymbol{q}_t$.

6.4 Inverse Medium Scattering

In the inverse medium scattering problem, the image q is a compactly supported spatially varying wave speed profile. This profile is probed by incoming plane waves $u_{\text{in}}(x) = e^{ik\langle x,s\rangle}$ with a known frequency k and direction s. The resulting scattered wave field is measured at distant receiver points. The forward model is given by the inhomogeneous Helmholtz equation with the Sommerfeld radiation condition:

$$\Delta u(x) + k^2(1 + q(x))u(x) = 0, \qquad x \in \Omega, \tag{6.15}$$

$$\sqrt{r}\left(\frac{\partial u_{\text{scat}}}{\partial r} - iku_{\text{scat}}\right) \to 0, \qquad r = \|x\|_2 \to \infty. \tag{6.16}$$

Where the total wave field u is the sum of the known incident wave u_{in} and the unknown scattered wave u_{scat}. The forward model $\mathcal{F}[q]$ evaluates u_{scat} at a set of receiver points. We use the HPS method coupled with a boundary integral equation [Gillman et al., 2015] to compute u_{scat}. The resulting boundary operator has a complicated dependence on q.

Standard linearization analysis of the forward model suggests that $\mathcal{F}$ can only resolve q up to frequency content $\leq 2k$ [Chen, 1995, 1997, Colton and Kress, 2013]. To reflect this, when evaluating the forward model, we project $\hat{\boldsymbol{q}}$ onto a bandlimited basis from Borges et al.

[2017] . For a square domain Ω centered at the origin with side length L, this basis is

$$B_k = \left\{ \frac{2}{L} \sin\left(m_1 \frac{\pi}{L} \left(x_1 - \frac{L}{2} \right) \right) \sin\left(m_2 \frac{\pi}{L} \left(x_2 - \frac{L}{2} \right) \right) \middle| \sqrt{m_1^2 + m_2^2} \leq \left\lfloor \frac{2kL}{\pi} \right\rfloor \right\}. \quad (6.17)$$

B_k is an orthonormal basis that spans bandlimited functions vanishing on $\partial\Omega$. We benchmark reconstructions against $\boldsymbol{q}_{\|}$, the projection of the ground-truth $\boldsymbol{q}_*$ onto B_k. We measure the relative ℓ_2 reconstruction error:

$$\text{Relative } \ell_2 \text{ Error}(\boldsymbol{q}) = \frac{\|\boldsymbol{q} - \boldsymbol{q}_*\|_2}{\|\boldsymbol{q}_*\|_2}, \quad (6.18)$$

and call the reconstruction error incurred by $\boldsymbol{q}_{\|}$ the "projection error". The linearization analysis suggests this amount of error is unavoidable given the frequency of the incident wave.

6.4.1 Related Work

The FISTA algorithm with total variation regularization has previously been applied to inverse scattering data [Liu et al., 2018]. This algorithm has also been used in the plug-and-play setting to apply general denoisers as priors for inverse scattering data [Kamilov et al., 2017]. The most recent version of this line of work is the CISOR algorithm [Ma et al., 2018], which uses a relaxed FISTA method with total variation and nonnegativity regularization. All of these approaches use a Born series as an approximation to $\mathcal{F}$, which is chosen because the Born approximation and its gradient can be quickly computed. The Born approximation is accurate only in the case of low frequency k and small scattering potential q_*; the approximation breaks down as these quantities increase.

Quasi-Newton methods are also popular for optimizing inverse scattering problems. Recursive linearization algorithms [Chen, 1995, Bao and Liu, 2003, Borges et al., 2017, Borges and Biros, 2018] use quasi-Newton methods to solve inverse scattering problems when data

at a range of frequencies k is present. These algorithms take a continuation in frequency approach by solving the inversion problem at the lowest frequency, and using the estimated scattering potential to "warm-start" the optimization at the next frequency. Typically, one iteration of Gauss-Newton is used to solve the sub-problem at each frequency.

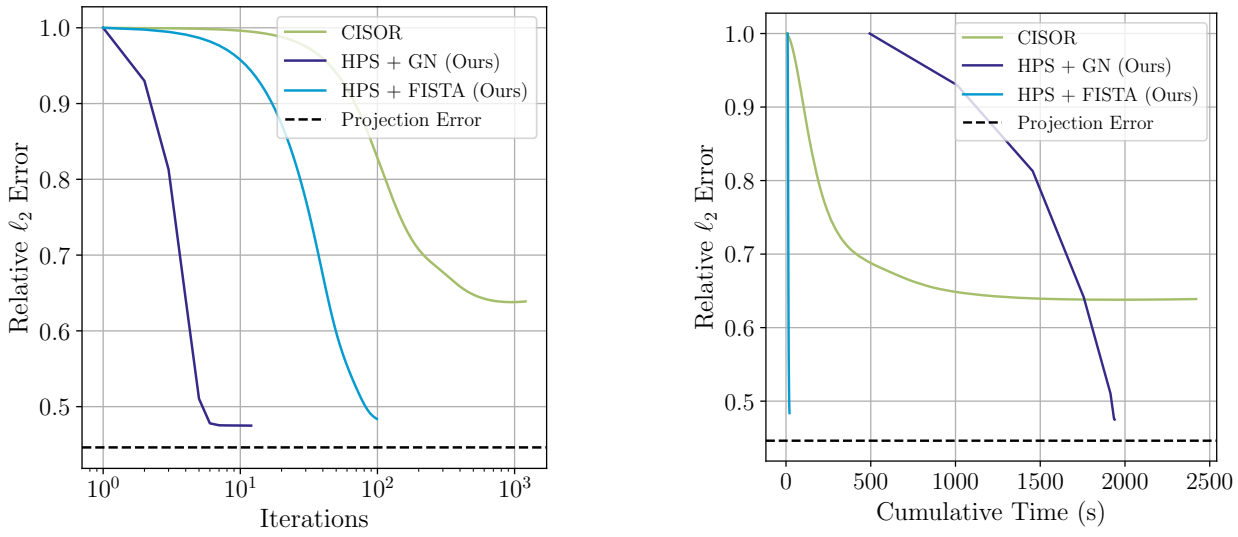
Experimental data for scattering problems have been generated in Geffrin et al. [2005]. This work carefully designed an experimental condition so that the two-dimensional PDE (Equations (6.15) and (6.16)) is an accurate model. This "Fresnel dataset" was generated using electromagnetic microwaves with wavelengths on the order of centimeters. The incident waves illuminated similarly-sized scattering objects with known dielectric constants. Receivers recorded time-harmonic measurements of the resulting wave field. The experiment was repeated for incident angular wave frequencies $\omega = 2\text{GHz}, ..., 10\text{GHz}$, with wavenumbers $k \approx 41, ..., 209$. We use the FoamDielExt phantom from this dataset in our experiments.

6.4.2 Single-Frequency Reconstruction

We compare our HPS + FISTA (Algorithm 9) and HPS + GN (Algorithm 10) algorithms with CISOR [Ma et al., 2018] by measuring the accuracy and runtime of reconstruction on the FoamDielExt target from the Fresnel dataset [Geffrin et al., 2005]. We first investigate the convergence of these algorithms using $\omega = 4\text{GHz}$. Each algorithm was terminated when the relative improvement in the objective function (Equation (6.4)) was less than 10^{-3} for five consecutive iterations. At each iteration, we compute the relative ℓ_2 reconstruction error and runtime of the iteration. For the CISOR algorithm, we search over step sizes and regularization parameters, and we present the results of the setting that produced the lowest relative ℓ_2 reconstruction error. For our algorithms, we search over the regularization parameter and bandlimit, and we present the results of the setting that produced the lowest relative ℓ_2 reconstruction error. For our method, we hold the step size $\gamma = 1.0$ fixed, but this can also be treated as a hyperparameter. All algorithms are initialized at zero and

reconstruct the images on a regular grid of size 128×128.

Figure 6.1 shows the result of this experiment when $\omega = 4$GHz. Figure 6.1a shows the HPS + GN algorithm converges to an accurate solution near the diffraction limit in a few iterations. HPS + FISTA also converges to an accurate reconstruction but requires an order of magnitude more iterations to converge. CISOR converges after a larger number of iterations, but converges to an inaccurate solution. Figure 6.1b shows the cumulative runtime of these algorithms. When measuring runtime, the HPS + FISTA algorithm is two orders of magnitude faster than the other algorithms; by using GPU acceleration and an efficient implementation of $\mathcal{F}$ and $\text{prox}_{\lambda\|\cdot\|_{\text{TV}}}$, our implementation computes about 5 iterations per second. In comparison, the HPS + GN algorithm requires about 500 seconds to compute each of the first few iterations, and CISOR requires about 2 seconds per iteration.



(a) (b)

Figure 6.1: Quantitative reconstruction results on the inverse scattering problem when $\omega = 4$GHz. Figure 6.1a show that because they have accurate gradient information, HPS + GN and HPS + FISTA converge in fewer iterations than CISOR. Figure 6.1b shows that the runtime of HPS + FISTA is much lower than the other methods.

We repeat this experiment for other values of ω and present the results in Table 6.1. As ω increases, the optimization problem becomes more oscillatory, so we can not expect the

algorithms to converge to a solution near the diffraction limit, even though the HPS algorithm is an accurate forward model in this range. We see that the HPS + FISTA algorithm is successful for $\omega \leq 6.0$ and diverges for higher frequencies. The HPS + GN algorithm diverges for $\omega \geq 5$. This behavior may be improved if we assume *a priori* knowledge of the low-frequency modes of q_*, as in Borges et al. [2017]. For some frequencies, the CISOR algorithm does not make any progress, indicated with an '—'. In these cases, the first gradient step estimates $\boldsymbol{z}_1$ with negative scattering potential, and a non-negativity projection step projects back to $\boldsymbol{s}_1 = \mathbf{0}$, so no progress is made.

ω **(GHz)**	**Projection Error**	**Method**	**Relative L2 Error**	**Runtime (s)**
4.0	0.446	CISOR	0.639	2,420.27
		HPS + GN (Ours)	0.475	1,938.86
		HPS + FISTA (Ours)	0.484	20.79
5.0	0.370	CISOR	—	—
		HPS + GN (Ours)	1.505	2,097.08
		HPS + FISTA (Ours)	0.396	37.72
6.0	0.340	CISOR	0.712	6,446.95
		HPS + GN (Ours)	1.356	2,172.77
		HPS + FISTA (Ours)	0.388	68.68
7.0	0.335	CISOR	0.893	2,518.68
		HPS + GN (Ours)	1.203	1,141.67
		HPS + FISTA (Ours)	1.173	19.54
8.0	0.301	CISOR	—	—
		HPS + GN (Ours)	1.108	986.72
		HPS + FISTA (Ours)	1.103	17.60
9.0	0.264	CISOR	0.893	2,518.68
		HPS + GN (Ours)	1.047	1,564.4
		HPS + FISTA (Ours)	1.011	80.40
10.0	0.243	CISOR	1.000	4,294.21
		HPS + GN (Ours)	0.987	1,640.41
		HPS + FISTA (Ours)	1.000	9.48

Table 6.1: Quantitative reconstruction results for the inverse scattering problem for different values of ω. At low frequencies, HPS + FISTA is accurate and two orders of magnitude faster than the other methods. At high frequencies, all methods converge to bad local minima.

6.4.3 Multi-Frequency Reconstruction

To produce more accurate estimates, we use a continuation in frequency method which solves a sequence of optimization problems, one for each incident frequency value present in the Fresnel dataset, $\omega = 2\text{GHz}, \ldots, 10\text{GHz}$. Because the HPS + FISTA algorithm is the fastest and continuation in frequency is necessarily serial, we choose this algorithm for our experiment. For each frequency ω, we run Algorithm 9 to compute an estimate $\hat{\boldsymbol{q}}_\omega$. For the lowest frequency $\omega = 2$, we initialize at zero, and for all other values of ω, we initialize the optimization at $\hat{\boldsymbol{q}}_{\omega-1}$. We use the same convergence criterion as before, which terminates the optimization when the relative improvement in the objective function (Equation (6.4)) is less than 10^{-3} for five successive iterations. Again, we search over regularization parameters and bandlimits, and report the results with the lowest relative ℓ_2 error.

Figure 6.2 shows the runtime and error convergence of this method. The vertical lines show that the optimization spends the most time at the lowest frequency, and then approximately equal times for the other frequencies. Each frequency makes an improvement on the relative ℓ_2 error metric. Figure 6.3 shows the reconstruction produced by the HPS + FISTA continuation in frequency method.

6.5 Diffuse Optical Tomography

Diffuse optical tomography, also known as near-infrared spectroscopy, is a prospective medical imaging modality which uses near-infrared light to gain functional images of the interior of biological samples [Boas et al., 2001, Gibson et al., 2005]. The imaging modality uses two key optical properties of biomedical imaging targets. First, these targets are highly-scattering media, meaning that photon density in the target can be modeled as a diffusion process. Second, at near-infrared wavelengths, hemoglobin molecules absorb photons at much higher rates than other biological media, meaning that blood flow can be localized to certain parts of the sample if the spatially varying photon absorption rate can be estimated from surface

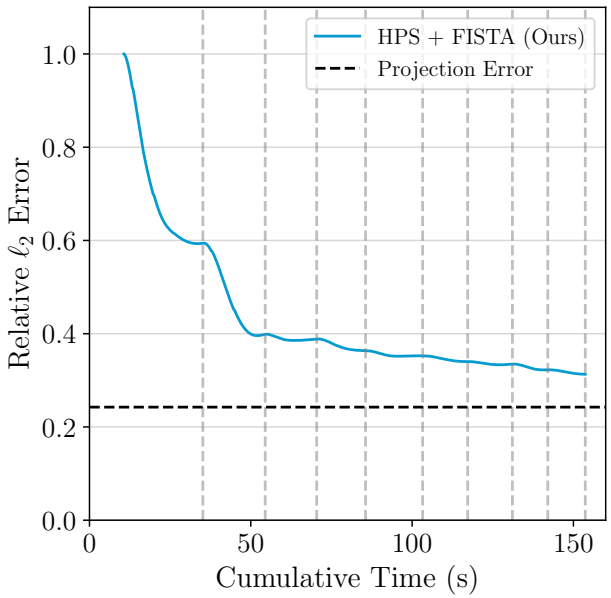


Figure 6.2: The continuation in frequency method quickly converges to a solution near the diffraction limit of the inverse scattering problem. Vertical lines show when the optimization advances in frequency.

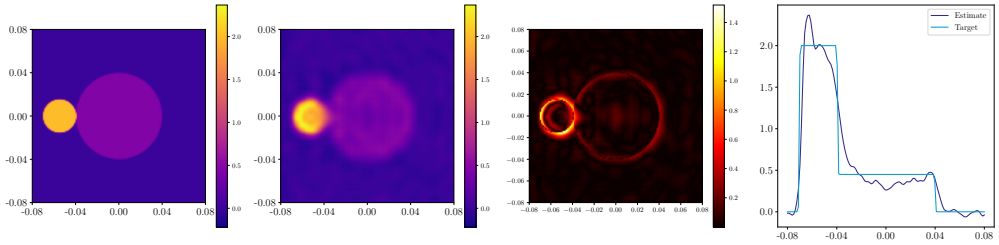
photon density measurements.

Consider the following model of continuous-wave diffuse optical tomography measurements, in which the solution $u(x)$ represents the steady-state photon density of an optical system:

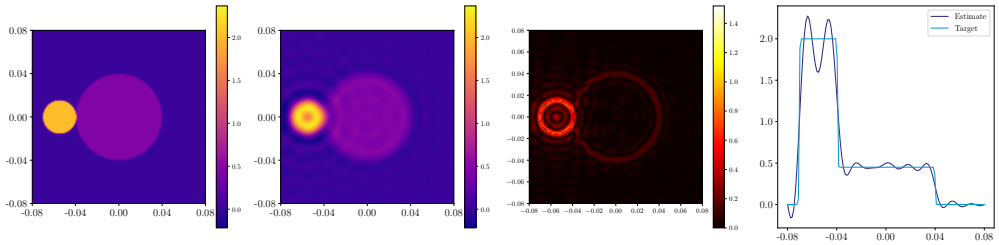
$$-\nabla \cdot (D(x)\nabla u(x)) + c\mu_a(x)u(x) = 0, \qquad x \in \Omega, \tag{6.19}$$

The system is driven by an illumination source on the boundary. The photon density diffuses throughout the medium with rate $D(x) \approx \frac{c}{2\mu_s'(x)}$ where $\mu_s'(x)$ is the reduced scattering coefficient and c is the speed of light. The absorption coefficient $\mu_a(x)$ gives the rate at which photons are absorbed throughout the domain. In a functional imaging setting, we expect the regions of high μ_a values to correspond to regions with concentrated blood flow.

In this model, it is impossible to separately determine the diffusion coefficient $D(x)$ and the absorption coefficient $\mu_a(x)$, so we instead work in a setting where the diffusion coefficient



(a) FISTA + HPS (Ours)



(b) $q_{\|}$

Figure 6.3: The inverse scattering reconstructions produced by the HPS + FISTA continuation in frequency method (Figure 6.3a) reconstruct the shape and contrast levels almost as well as the projection of the ground-truth (Figure 6.3b). The first panel shows the ground-truth q_*; the second panel shows the reconstruction; the third panel shows the absolute value difference of the first two panels. The fourth panel shows the trace of the reconstruction along $x_2 = 0$.

is constant inside Ω [Konecky et al., 2008]. After setting $D(x) = D$, we can divide both sides of Equation (6.19) by D, and represent the absorption as a background level of absorption μ_0 and a perturbation $q(x)$:

$$\mu_a(x) = \mu_0 \left(1 + q(x)\right). \tag{6.20}$$

we can then re-write Equation (6.19):

$$-\Delta u(x) + k_0^2 \left(1 + q(x)\right) u(x) = 0, \qquad x \in \Omega, \tag{6.21}$$

$$u(x) + \ell u_n(x) = J(x), \qquad x \in \partial\Omega. \tag{6.22}$$

The boundary condition (Equation (6.22)) specifies partial internal reflection, depending on

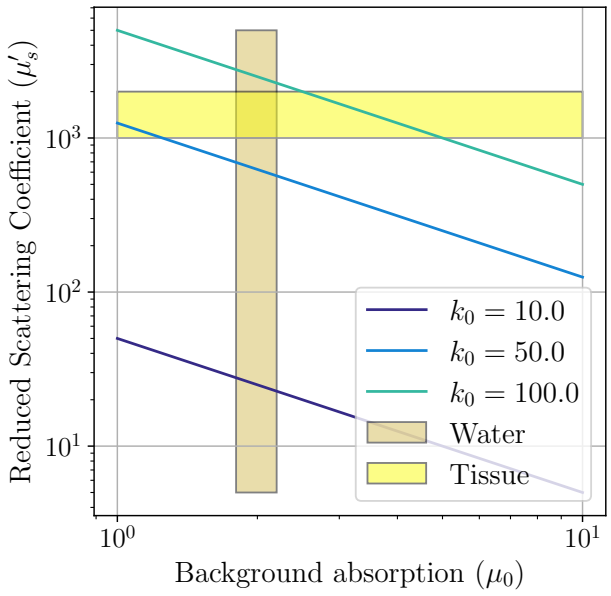


Figure 6.4: Showing the relationship between the diffuse wavenumber k_0 and optical parameters collected from the literature. The absorption coefficient of 750nm-wavelength light in water is from Boas et al. [2001]. The reduced scattering coefficient of soft tissue at 750nm-wavelength light is from Jacques [2013].

the parameter ℓ, which is determined by $\mu_s'(x)$ inside and outside Ω. A typical value for this parameter is $\ell \approx 1.0$. The parameter k_0, sometimes called a "diffuse wavenumber", is $k_0 = \sqrt{2\mu_s'\mu_0}$. Figure 6.4 shows how k_0 relates to biologically-relevant samples. The diffuse optical tomography forward model maps the perturbation in the absorption coefficient q to the solution on the boundary:

$$\mathcal{F}[q] = u|_{\partial\Omega} \qquad u \text{ solves Equations (6.21) and (6.22).} \tag{6.23}$$

6.5.1 Forward Simulations

We use the GPU-accelerated HPS method to compute solutions of Equations (6.21) and (6.22). We use a scaled version of the FoamDielExt phantom described in Section 6.4 to qualitatively understand the effect of a large absorbing region with differing contrast

levels. Figure 6.5 shows the result of this simulation for different values of k_0. As k_0 increases, the sensitivity of the boundary data to the perturbation decreases, indicating that the conditioning of the problem worsens.

6.5.2 Related Work

Methods similar to the ones described in Section 6.3 have been applied to reconstruct absorption coefficients in diffuse optical tomography. Proximal quasi-Newton methods with total variation regularization have been applied to the diffuse optical tomography problem [Douiri et al., 2005, Lu et al., 2019, Tang, 2021]. Chen et al. [2014] use FISTA and a group norm regularization to promote clustered sparsity in diffuse optical tomography reconstructions. These methods all implement $\mathcal{F}$ with a simple finite element discretization or linear approximation. Partial work toward a GPU-accelerated implementation of $\mathcal{F}$ appears in Doulgerakis-Kontoudis et al. [2017], but they do not apply this method toward inverse problems, and the accuracy of their method is limited due to the small number of discretization points able to fit into GPU memory at once. We believe the high-order spectral discretization described in Section 6.2.3 will afford greater accuracy at a fixed number of discretization points. Experimental datasets of continuous-wave optical tomography measurements have been constructed and used to evaluate numerical inversion methods [Wang et al., 2005, Konecky et al., 2008].

6.6 Conclusion

This chapter develops optimization methods that are fast and accurate when applied to PDE-based imaging inverse problems. We are able to reconstruct scattering potentials from the Fresnel dataset to an accuracy near the diffraction limit, two orders of magnitude faster than past methods. An important and surprising contribution of this work is the experimental evidence suggesting that first-order proximal methods, namely FISTA, are much faster than

the Gauss-Newton methods commonly used to optimize inverse problems.

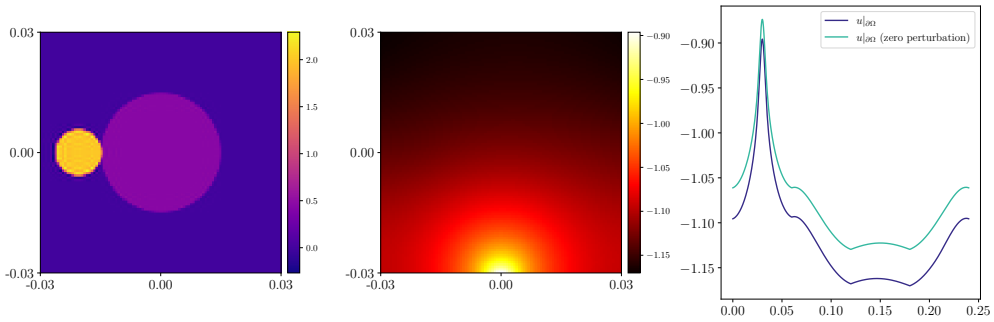
6.6.1 Future Work

This chapter leaves open future work needed to develop this research into a full publication.

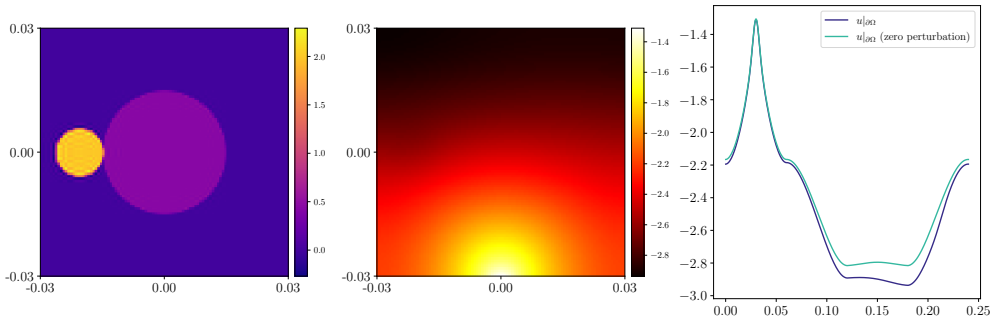
First, experiments using the analytical formulation of Jacobian-vector and vector-Jacobian products proposed in Borges et al. [2017] must be conducted. We plan to interpret these analytical formulations as the Jacobian-vector and vector-Jacobian products corresponding to an approximate scattering forward model. We plan to extend this method to the diffuse optical tomography setting as well.

More inverse scattering experiments are necessary. We plan to evaluate our methods on more complex imaging targets, such as the scattering potentials from Melia et al. [2025b], which contain piecewise-constant shapes overlaid on a smoothly varying background.

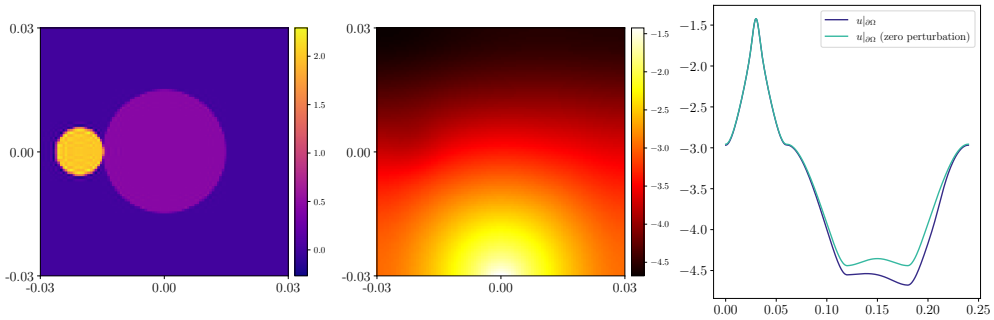
Finally, experiments solving the diffuse optical tomography inverse problem are also necessary. We plan to use an alternate numerical method [Burns et al., 2020] to generate data and evaluate the algorithms proposed in Section 6.3 on this problem.



(a) $k_0 = 10.0$



(b) $k_0 = 50.0$



(c) $k_0 = 100.0$

Figure 6.5: We show simulations of the diffuse optical tomography model (Equations (6.21) and (6.22)) for the same perturbation q and different diffuse wavenumbers k_0. The first panel in each row shows the perturbation q. The second panel shows $\log_{10}(u(x))$ on the interior of Ω, when the illumination source is placed at $(0.0, -0.03)$ on the boundary. The third panel shows $\log_{10}$ of the trace of the solution along the boundary starting at the bottom-left corner, proceeding counter-clockwise. The trace of the solution computed with the perturbation (dark blue) and without the perturbation (light blue) are both shown to describe the sensitivity of the measurements to perturbations q.

CHAPTER 7

CONCLUSION

This thesis considers the hypothesis that algorithms from numerical analysis are a useful design tool to improve the practice of scientific machine learning. The areas in which we test this thesis are broad; we show that prediction tasks in computational chemistry, deep neural network design, and image reconstruction problems all benefit from the application of numerical analysis tools. In particular, we contribute the following:

- A rotation-invariant random feature method designed for fast inference on point clouds (Chapter 2).
- A deep neural network and training method designed to emulate recursive linearization algorithms for solving multi-frequency inverse scattering tasks (Chapter 3).
- Algorithm developments for the fast execution of fast direct solvers of elliptic PDEs on GPUs (Chapter 4).
- An open-source software package implementing these algorithms (Chapter 5).
- Algorithms for using GPU-accelerated fast direct solvers for inverse problems in scientific imaging (Chapter 6).

In the rest of this chapter, we describe how the contributions of this thesis can lead to further improvements in the field of scientific machine learning.

7.1 Future work

7.1.1 Developing Machine Learning Methods with Insights from Numerical Analysis

This thesis makes contributions that show the benefit of designing machine learning methods using insights from algorithms in numerical analysis. The field of scientific machine learning is broad and still developing; this section discusses some of the many areas of scientific machine learning that may benefit from this approach.

One potential avenue for improvement is the design of neural operators, deep neural networks trained to approximate coefficient-to-solution maps or time advancement operators. Chapter 4 shows that domain decomposition methods offer computational benefits when exactly implementing the coefficient-to-solution map in a classical setting without machine learning. Neural operators approximating coefficient-to-solution maps may see an analogous statistical benefit by only requiring the network to learn a local coefficient-to-solution map on a small patch of the domain. Such patch-based methods have been shown to be useful in increasing the data efficiency of diffusion models trained to solve imaging inverse problems [Hu et al., 2024]. How to efficiently and accurately merge patch-based information together into a global solution without relying on dense matrix inversions is a major technical challenge in this direction.

Other advances could come from neural operators approximating time advancement operators in time-dependent PDEs. Most neural operators are designed to operate auto-regressively, where the network takes as input the state at time t and predicts the state at time $t + \Delta t$. This is based off a forward Euler time integration scheme, but numerical analysis offers a large variety of "time integrators", or methods for advancing the solution in time. It may be the case that emulating different time integrators provides benefits in certain problem settings. For example, Jiang et al. [2025] show that emulating implicit time

integration schemes can greatly increase the prediction stability when emulating chaotic fluid dynamics. One interesting class of time integrators is "operator splitting" schemes, which treat different parts of the PDE differently. Neural operators designed to emulate operator splitting schemes could be implemented as a collection of networks; this may increase the stability of long-rollout predictions.

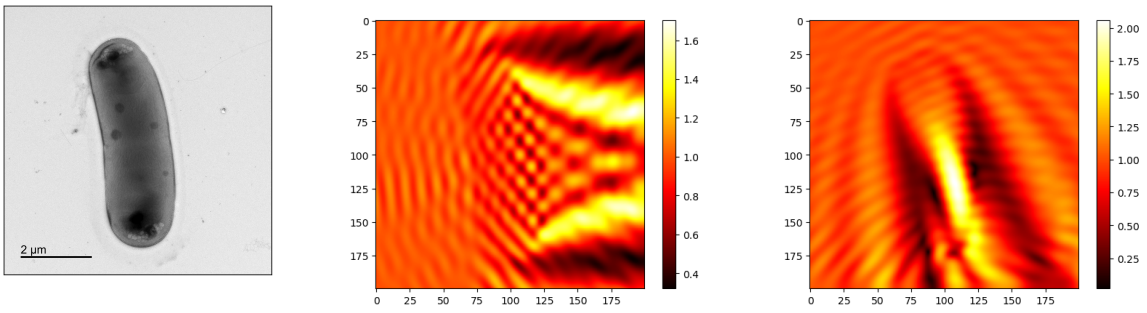
In a different area of scientific machine learning, the random feature method from Chapter 2 could be combined with fast multipole methods [Beatson and Greengard, 1997] to efficiently compute group-invariant kernel interactions between all pairs in extremely large point clouds. This could be particularly useful for particle jet tagging [Kasieczka et al., 2019], which requires inference on large point clouds in extremely low-latency environments.

7.1.2 Developing Numerical PDE Solvers for use in Deep Learning Settings

Chapters 4 and 5 make contributions to the field of scientific machine learning by developing computational methods for elliptic PDEs on simple geometries. Many problems in biology, engineering, and computer graphics require fast, accurate solvers of PDEs defined on more complicated geometries, such as smooth curved surfaces or surfaces with sharp edges and corners. The field lacks high-quality GPU-accelerated solvers of surface PDEs. The surface HPS method developed in Fortunato [2024] is a promising candidate for GPU acceleration using methods developed in Chapter 4.

One planned application of the solver developed in Chapters 4 and 5 is to simulate physical phenomena to augment an existing biological dataset. The augmented data can then be used to fit a model. We are pursuing a study of phototactic behavior in rod-shaped cyanobacteria [Burriesci and Bhaya, 2008], using experimental tracking data [Bunbury et al., 2022] which measures phototactic responses of bacteria under varying light conditions. We will augment this dataset with simulations of light propagation inside the cell cytoplasm; see Figure 7.1. The simulations in Figure 7.1 show that when the cell is aligned with the axis

of wave propagation, the cytoplasm focuses the light on the rear cell wall. This supports a prevailing hypothesis in the field that phototaxis is modulated by cell cytoplasm focusing light onto a portion of the rear cell wall [Schuergers et al., 2017]. We plan to use tools from computer vision and sparse equation discovery to discover a mathematical model of phototaxis in cyanobacteria. While this exposition discusses one particular application, the idea of using a PDE solver to augment an existing dataset has broad applicability.



(a) (b) (c)

Figure 7.1: Figure 7.1a shows *Synechococcus*, a rod-shaped cyanobacteria; this image was provided by Frederick Bunbury. Figures 7.1b and 7.1c show a simulation of 780nm light propagating through the cell from the left and top, respectively.

7.1.3 Building a Computational Toolbox for PDE-Based Inverse Problems in Scientific Imaging

One major theme of this thesis is building a computational toolbox for PDE-based inverse problems. This section describes future advances in this direction.

One particular set of missing tools is efficient solvers for 3D problems. For many of the PDE-based imaging problems discussed in this thesis, the physical interactions are nonlocal, which means that 2D simulations are insufficient models of 3D experimental data. This occurs because interactions propagate in and out of any 2D slice of the 3D domain. While Chapter 4 describes algorithms and experiments for 3D PDEs, the forward models are limited in the size of systems we can resolve. We have identified other computational tools, such as

matrix compression or iterative solvers, which may be necessary for high-resolution forward models in 3D.

For wave scattering simulations in particular, the method presented in Chapter 4 uses a boundary integral equation method to enforce the infinite radiation condition in 2D. In three dimensions, discretizing and inverting this boundary integral equation will become prohibitively expensive, so other methods will become necessary. We are actively pursuing the use of perfectly-matched layers [Berenger, 1994] to simulate the 3D radiation condition while avoiding the boundary integral equation.

Finally, we believe that the contributions in Chapters 4 to 6 will enable more sophisticated deep learning approaches to PDE-based inverse problems. One such approach is a plug-and-play priors approach with a deep neural network denoiser. The algorithms developed in Chapter 6 will directly apply in this setting by replacing the proximal regularization steps with evaluations of the neural network.

We believe that our contributions will be useful in the development of unrolled optimization approaches, where steps solving a PDE are interlaced with deep neural network blocks. This will require managing the memory requirements on the GPU. During training, we need to store mini-batches of data, the weights and activation of the deep neural network, and the data used by the fast direct PDE solver.

APPENDIX A

ROTATION-INVARIANT RANDOM FEATURES: OMITTED DETAILS AND EXPERIMENTS

A.1 Full Integration Details

We wish to solve the following integral, which appears in the definition of rotation-invariant random features Equation (2.6):

$$\int_{SO(3)} \langle Q \circ p_i, g \rangle^2 dQ \tag{A.1}$$

From Section 2.4.1, we know this integral can decompose into

$$\int_{SO(3)} \langle Q \circ p_i, g \rangle^2 dQ = \sum_{j_1, j_2 = 1}^{N_i} \int_{SO(3)} g(Qx_{i,j_1}) g(Qx_{i,j_2}) dQ \tag{A.2}$$

And we also have chosen a particular functional form for our random function g:

$$g(x) = \sum_{k=1}^{K} \sum_{\ell=0}^{L} \sum_{m=-\ell}^{\ell} w_{m,k}^{(\ell)} Y_m^{(\ell)}(\hat{x}) R_k(\|x\|) \tag{A.3}$$

Repeated application of the linearity of integration gives us:

$$\int_{SO(3)} \langle Q \circ p_i, g \rangle^2 dQ = \tag{A.4}$$

$$= \sum_{j_1, j_2 = 1}^{N} \int_{SO(3)} g(Qx_{i,j_1}) g(Qx_{i,j_2}) dQ \tag{A.5}$$

$$= \sum_{j_1,j_2=1}^{N_i} \sum_{k_1,k_2=1}^{K} \sum_{\ell_1,\ell_2=0}^{L} \sum_{m_1=-\ell_1}^{\ell_1} \sum_{m_2=-\ell_2}^{\ell_2} w_{m_1,k_1}^{(\ell_1)} w_{m_2,k_2}^{(\ell_2)} R_{k_1}(\|x_{i,j_1}\|) R_{k_2}(\|x_{i,j_2}\|)$$
$$\int_{SO(3)} Y_{m_1}^{(\ell_1)}(Q\hat{x}_{i,j_1}) Y_{m_2}^{(\ell_2)}(Q\hat{x}_{i,j_2}) dQ \quad \text{(A.6)}$$

We can now focus on the integral in Equation (A.6), apply the rotation rule for spherical harmonics introduced in Section 2.3.1, and again exploit the linearity of the integral.

$$\int_{SO(3)} Y_{m_1}^{(\ell_1)}(Q\hat{x}_{i,j_1}) Y_{m_2}^{(\ell_2)}(Q\hat{x}_{i,j_2}) dQ \quad \text{(A.7)}$$
$$= \int_{SO(3)} \left(\sum_{m_1'=-\ell_1}^{\ell_1} Y_{m_1'}^{(\ell_1)}(\hat{x}_{i,j_1}) D^{(\ell_1)}(Q)_{m_1,m_1'} \right) \left(\sum_{m_2'=-\ell_2}^{\ell_2} Y_{m_2'}^{(\ell_2)}(\hat{x}_{i,j_2}) D^{(\ell_2)}(Q)_{m_2,m_2'} \right) dQ \quad \text{(A.8)}$$
$$= \sum_{m_1'=-\ell_1}^{\ell_1} \sum_{m_2'=-\ell_2}^{\ell_2} Y_{m_1'}^{(\ell_1)}(\hat{x}_{i,j_1}) Y_{m_2'}^{(\ell_2)}(\hat{x}_{i,j_2}) \int_{SO(3)} D^{(\ell_1)}(Q)_{m_1,m_1'} D^{(\ell_2)}(Q)_{m_2,m_2'} dQ \quad \text{(A.9)}$$
$$\quad \text{(A.10)}$$

At this point, we are able to apply the integration rule for elements of Wigner-D matrices introduced in Section 2.3.1.

$$\int_{SO(3)} Y_{m_1}^{(\ell_1)}(Q\hat{x}_{i,j_1}) Y_{m_2}^{(\ell_2)}(Q\hat{x}_{i,j_2}) dQ \quad \text{(A.11)}$$
$$= \sum_{m_1'=-\ell_1}^{\ell_1} \sum_{m_2'=-\ell_2}^{\ell_2} Y_{m_1'}^{(\ell_1)}(\hat{x}_{i,j_1}) Y_{m_2'}^{(\ell_2)}(\hat{x}_{i,j_2}) (-1)^{m_1-m_1'} \frac{8\pi^2}{2\ell+1} \delta_{-m_1,m_2} \delta_{-m_1',m_2'} \delta_{\ell_1,\ell_2} \quad \text{(A.12)}$$
$$= \delta_{\ell_1,\ell_2} \delta_{-m_1,m_2} \frac{8\pi^2}{2\ell+1} \sum_{m_1'=-\ell_1}^{\ell_1} (-1)^{m_1-m_1'} Y_{m_1'}^{(\ell_1)}(\hat{x}_{i,j_1}) Y_{-m_1'}^{(\ell_1)}(\hat{x}_{i,j_2}) \quad \text{(A.13)}$$

To put it all together, we substitute Equation (A.13) into Equation (A.6), and we see many terms drop out of the sums:

$$(A.6) = \sum_{j_1,j_2=1}^{N_i} \sum_{k_1,k_2=0}^{K} \sum_{\ell_1,\ell_2=0}^{L} \sum_{m_1=-\ell_1}^{\ell_1} \sum_{m_2=-\ell_2}^{\ell_2} w^{(\ell_1)}_{m_1,k_1} w^{(\ell_2)}_{m_2,k_2} R_{k_1}(\|x_{i,j_1}\|) R_{k_2}(\|x_{i,j_2}\|)$$
$$\delta_{\ell_1,\ell_2}\delta_{-m_1,m_2} \frac{8\pi^2}{2\ell+1} \sum_{m_1'=-\ell_1}^{\ell_1} (-1)^{m_1-m_1'} Y^{(\ell_1)}_{m_1'}(\hat{x}_{i,j_1}) Y^{(\ell_1)}_{-m_1'}(\hat{x}_{i,j_2}) \quad \text{(A.14)}$$

$$\int_{SO(3)} \langle Q \circ p_i, g \rangle^2 dQ = \sum_{j_1,j_2=1}^{N_i} \sum_{k_1,k_2=0}^{K} \sum_{\ell_1=0}^{L} \sum_{m_1=-\ell_1}^{\ell_1} w^{(\ell_1)}_{m_1,k_1} w^{(\ell_1)}_{-m_1,k_2} R_{k_1}(\|x_{i,j_1}\|) R_{k_2}(\|x_{i,j_2}\|)$$
$$\frac{8\pi^2}{2\ell+1} \sum_{m_1'=-\ell_1}^{\ell_1} (-1)^{m_1-m_1'} Y^{(\ell_1)}_{m_1'}(\hat{x}_{i,j_1}) Y^{(\ell_1)}_{-m_1'}(\hat{x}_{i,j_2}) \quad \text{(A.15)}$$

Once the spherical harmonics and radial functions are evaluated, this sum has $O(N^2K^2L^3)$ terms.

A.1.1 Basis Expansion

When implementing Equation (A.15) to compute rotationally-invariant random features, one can see that a large amount of computational effort can be re-used between different random features evaluated on the same sample. A table of spherical harmonic evaluations for all the points $\{\hat{x}_{i,1}, \ldots \hat{x}_{i,N_i}\}$ can be pre-computed once and re-used. Also, by re-arranging the

order of summation in Equation (A.15), we arrive at this expression:

$$\int_{SO(3)} \langle Q \circ p_i, g \rangle^2 dQ = \sum_{k_1,k_2=1}^{K} \sum_{\ell_1=0}^{L} \sum_{m_1=-\ell_1}^{\ell_1} w^{(\ell_1)}_{m_1,k_1} w^{(\ell_1)}_{-m_1,k_2}$$
$$\sum_{j_1,j_2=1}^{N_i} R_{k_1}(\|x_{i,j_1}\|) R_{k_2}(\|x_{i,j_2}\|) \frac{8\pi^2}{2\ell+1} \sum_{m_1'=-\ell_1}^{\ell_1} (-1)^{m_1-m_1'} Y^{(\ell_1)}_{m_1'}(\hat{x}_{i,j_1}) Y^{(\ell_1)}_{-m_1'}(\hat{x}_{i,j_2}) \quad \text{(A.16)}$$

Pre-computing everything after the line break allows for the re-use of a large amount of computational work between different random feature evaluations. We can interpret this pre-computation as an expansion of our point cloud in a rotationally-invariant basis B, which depends on the choice of radial function. Appendix A.2 relates this basis expansion to the Atomic Cluster Expansion method.

$$B[\ell, m, k_1, k_2] = \sum_{j_1,j_2=1}^{N_i} R_{k_1}(\|x_{i,j_1}\|) R_{k_2}(\|x_{i,j_2}\|) \frac{8\pi^2}{2\ell+1} \sum_{m'=-\ell}^{\ell} (-1)^{m-m'} Y^{(\ell)}_{m'}(\hat{x}_{i,j_1}) Y^{(\ell)}_{-m'}(\hat{x}_{i,j_2}) \quad \text{(A.17)}$$

$$= \sum_{j_1,j_2=1}^{N_i} R_{k_1}(\|x_{i,j_1}\|) R_{k_2}(\|x_{i,j_2}\|) \int_{SO(3)} Y^{(\ell)}_{m}(Q\hat{x}_{i,j_1}) Y^{(\ell)}_{-m}(Q\hat{x}_{i,j_2}) dQ \quad \text{(A.18)}$$

In our implementation we precompute this B tensor once for each sample, and then we perform a tensor contraction with the random weights:

$$\int_{SO(3)} \langle Q \circ p_i, g \rangle^2 dQ = \sum_{k_1,k_2=1}^{K} \sum_{\ell_1=0}^{L} \sum_{m_1=-\ell_1}^{\ell_1} w^{(\ell_1)}_{m_1,k_1} w^{(\ell_1)}_{-m_1,k_2} B[\ell_1, m_1, k_1, k_2] \quad \text{(A.19)}$$

A.2 Connection Between Our Method and Randomized ACE Methods

A.2.1 Overview of the ACE basis

The atomic cluster expansion (ACE) method builds a rotationally-invariant basis for point clouds that are an extension of our pre-computed basis expansion described in Appendix A.1.1. In the ACE method, a preliminary basis $A_{k,\ell,m}$ is formed by projecting a point cloud function $p_i(x) = \sum_j \delta(x - x_{i_j})$ onto a single-particle basis function $\varphi_{k,\ell,m}(x)$:

$$\varphi_{k,\ell,m}(x) = R_k(\|x\|)Y_m^{(\ell)}(x) \tag{A.20}$$

$$A_{k,\ell,m}(p_i) = \langle \varphi_{k,\ell,m}, p_i \rangle \tag{A.21}$$

$$= \sum_j \varphi_{k,\ell,m}(x_{i_j}) \tag{A.22}$$

where R_k is a set of radial functions. In Kovács et al. [2021], this set of radial functions is defined by taking a nonlinear radial transformation, and then the radial functions are a set of orthogonal polynomials defined on the transformed radii. This basis set is augmented with auxiliary basis functions to ensure the potential energy of two atoms at extremely nearby points diverges to infinity.

The $A_{k,\ell,m}$ basis is a first step, but it only models single particles at a time. To model pairwise particle interactions, or higher-order interactions, the A basis is extended by taking tensor products. First, a "correlation order" N is chosen. Then, the A basis is extended via tensor products:

$$A_{\underline{k},\underline{\ell},\underline{m}}(p_i) = \prod_{\alpha=1}^{N} A_{k_\alpha,\ell_\alpha,m_\alpha}(p_i) \tag{A.23}$$

where $\underline{k} = (k_\alpha)_{\alpha=1}^N$, $\underline{\ell} = (\ell_\alpha)_{\alpha=1}^N$, and $\underline{m} = (m_\alpha)_{\alpha=1}^N$ are multi-indices.

Finally, to ensure invariance with respect to permutations, reflections, and rotations of the point cloud, the ACE method forms a Haar measure dg over $O(3)$ and computes a symmetrized version of the A basis:

$$B_{\underline{k},\underline{\ell},\underline{m}}(p_i) = \int_{O(3)} A_{\underline{k},\underline{\ell},\underline{m}}(H \circ p_i) dH \tag{A.24}$$

Drautz [2019] gives a detailed description of how this integral is performed. Applications of representation theory give concise descriptions of which multi-indices $\underline{k}, \underline{\ell}, \underline{m}$ remain after performing this integral; many are identically zero.

A.2.2 Connection to Our Method

If one chooses to compute an ACE basis with correlation order $N = 2$, the resulting basis is very similar to our pre-computed basis expansion in Equation (A.18).

$$B_{\underline{k},\underline{\ell},\underline{m}}(p_i) = \int_{O(3)} A_{\underline{k},\underline{\ell},\underline{m}}(H \circ p_i) dH \tag{A.25}$$

$$= \int_{O(3)} A_{k_1,\ell_1,m_1}(H \circ p_i) A_{k_2,\ell_2,m_2}(H \circ p_i) dH \tag{A.26}$$

$$= \int_{O(3)} \left(\sum_{j_1} \varphi_{k_1,\ell_1,m_1}(H x_{i_{j_1}}) \right) \left(\sum_{j_2} \varphi_{k_2,\ell_2,m_2}(H x_{i_{j_2}}) \right) dH \tag{A.27}$$

$$= \int_{O(3)} \left(\sum_{j_1} R_{k_1}(\|x_{i_{j_1}}\|) Y_{m_1}^{(\ell_1)}(H \hat{x}_{i_{j_1}}) \right) \left(\sum_{j_2} R_{k_2}(\|x_{i_{j_2}}\|) Y_{m_2}^{(\ell_2)}(H \hat{x}_{i_{j_2}}) \right) dH \tag{A.28}$$

$$= \sum_{j_1,j_2} R_{k_1}(\|x_{i_{j_1}}\|) R_{k_2}(\|x_{i_{j_2}}\|) \int_{O(3)} Y_{m_1}^{(\ell_1)}(H \hat{x}_{i_{j_1}}) Y_{m_2}^{(\ell_2)}(H \hat{x}_{i_{j_2}}) dH \tag{A.29}$$

The remaining difference between our basis expansion in Equation (A.18) and the ACE basis expansion with correlation order $N = 2$ is the ACE basis integrates over all of $O(3)$, while our method only enforces invariance with respect to $SO(3) \subset O(3)$.

Method	Test Error (eV)
Spherical CNNs	0.1565
FCHL19	0.0541
Random Features (Ours)	0.0660
Linear Model of B basis (Ours)	0.1542

Table A.1: Reported are test errors on the QM7 dataset for Spherical CNNs, FCHL19, and two variants of our method. The first variant is the full rotation-invariant random features model. The second variant is a linear model of our basis expansion. We note that while our random feature model is the second-best performing model on this benchmark, our linear model is the nearly the worst-performing model.

So one can describe our random features as random nonlinear projections of a simplified version of the ACE basis with correlation order $N = 2$ using extremely simple radial functions.

A.2.3 Benefit of Random Features

We conduct the following experiment to quantify the benefit of using random nonlinear features over fitting a linear model in our simplified version of the ACE basis. Using the same hyperparameters as in our energy regression experiments, we compute our basis expansion for all the samples in the QM7 dataset. Using the same train/validation/test split, we perform ridge regression by fitting a linear model in our basis B with L^2 regularization. We use the validation set to identify the best-performing model and report errors on the test set in Table A.1. We note that the best-performing linear model has more than twice the error of the best performing random features model.

A.3 Representation Theory of the 3D Rotation Group

A.3.1 Conjugation of Wigner-D Matrices

In the following, we derive a formula for conjugating an element of a Wigner-D matrix. First, we use definition 6.44 in Thompson [1994] for the Wigner-D matrices:

$$D^{(\ell)}_{m',m}(\alpha,\beta,\gamma) = e^{-im'\alpha} d^{(\ell)}_{m',m}(\beta) e^{-im\gamma} \tag{A.30}$$

Here, (α,β,γ) are Euler angles parameterizing a rotation, and $d^{(\ell)}_{m',m}$ is an element of a Wigner-d (small d) matrix. We omit the exact definition of $d^{(\ell)}_{m',m}$, but we note the following symmetry properties:

$$d^{(\ell)}_{-m',-m}(\beta) = d^{(\ell)}_{m',m}(-\beta) \tag{A.31}$$

$$d^{(\ell)}_{m',m}(-\beta) = d^{(\ell)}_{m,m'}(\beta) \tag{A.32}$$

$$d^{(\ell)}_{m,m'}(\beta) = (-1)^{m-m'} d^{(\ell)}_{m',m}(\beta) \tag{A.33}$$

The relationship between the Wigner d matrix elements allows us to derive a formula for conjugating an element of a Wigner D matrix:

$$D^{(\ell)}_{m',m}(\alpha,\beta,\gamma)^* = e^{im'\alpha} d^{(\ell)}_{m',m}(\beta)^* e^{im\gamma} \tag{A.34}$$

$$= e^{-i(-m')\alpha} d^{(\ell)}_{m',m}(\beta) e^{-i(-m)\gamma} \tag{A.35}$$

$$= e^{-i(-m')\alpha} d^{(\ell)}_{-m',-m}(-\beta) e^{-i(-m)\gamma} \tag{A.36}$$

$$= e^{-i(-m')\alpha} d^{(\ell)}_{-m,-m'}(\beta) e^{-i(-m)\gamma} \tag{A.37}$$

$$= e^{-i(-m')\alpha} (-1)^{-m+m'} d^{(\ell)}_{-m',-m}(\beta) e^{-i(-m)\gamma} \tag{A.38}$$

$$= (-1)^{m'-m} D^{(\ell)}_{-m',-m}(\alpha,\beta,\gamma) \tag{A.39}$$

A.3.2 Orthogonality Relations

We use the following orthogonality relationship between elements of Wigner D matrices:

$$\int_{SO(3)} D^{(\ell_1)}(Q)^*_{m_1,k_1} D^{(\ell_2)}(Q)_{m_2,k_2} dQ = \frac{8\pi^2}{2\ell+1} \delta_{m_1,m_2} \delta_{k_1,k_2} \delta_{\ell_1,\ell_2} \tag{A.40}$$

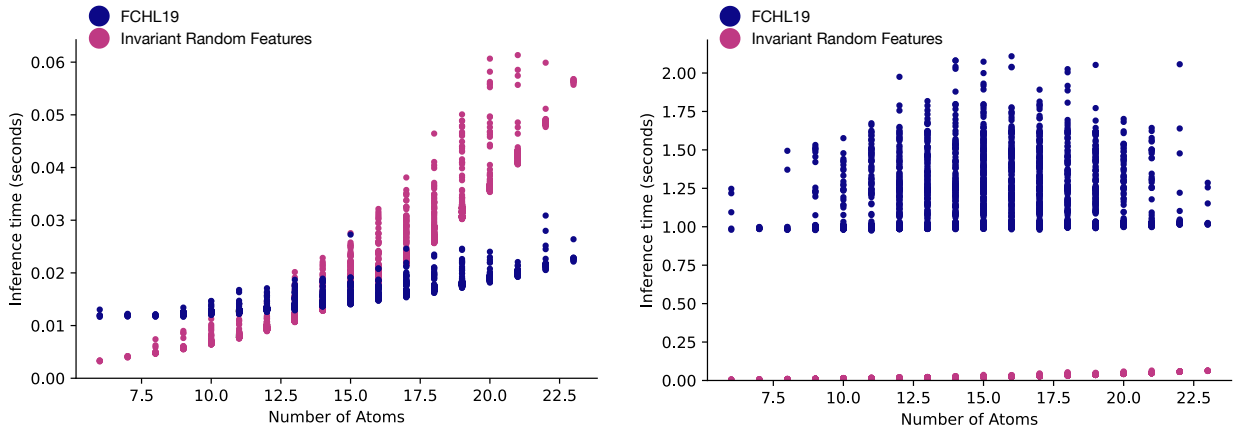
This is a well-known fact about the Wigner-D matrices. For a proof, one can derive this relationship from Schur orthogonality relations, which are a consequence of Schur's lemma. A direct calculation can be found in section 6.4.2 of Thompson [1994]. Combining the two facts from above, we arrive at the following identity, which we use to compute our rotation-invariant random features.

$$\int_{SO(3)} D^{(\ell_1)}(Q)_{m_1,k_1} D^{(\ell_2)}(Q)_{m_2,k_2} dQ = (-1)^{m_1-k_1} \frac{8\pi^2}{2\ell+1} \delta_{-m_1,m_2} \delta_{-k_1,k_2} \delta_{\ell_1,\ell_2} \tag{A.41}$$

A.4 Additional Latency Experiments

In this section we present extra experiments about the prediction latency of our method. Figure A.1 compares the effect of molecule size on prediction latency for our method and FCHL19. Importantly, both methods have a precomputation step that has complexity $O(N^2)$ where N is the number of atoms. For our experiments on ModelNet40, the quadratic dependence on the number of points causes slow prediction latency, but we are able to alleviate this latency by reducing other hyperparameters (Figure A.2).

We also investigate whether using deep multilayer perceptrons (MLPs) provides an advantageous accuracy-latency tradeoff compared to our original ridge regression method. We first investigate the prediction latency incurred by using MLPs with different widths and



(a) Latency for small models

(b) Latency for large models

Figure A.1: The number of atoms in a molecule affects the prediction latency for both FCHL19 and the invariant random features method. In Figure A.1a, we show the prediction latencies for the smallest models tested, corresponding to 250 random features for our method and 200 training samples for FCHL19. For this model size, a quadratic scaling in the number of atoms largely determines the prediction latency. Figure A.1b shows that for larger models (2,000 random features for our method and 5,000 training samples for FCHL19), the random features method is still in a regime where prediction latency is determined by the number of atoms, while FCHL19's prediction latency is dominated by the number of inner products required to evaluate the kernel.

depths. In Figure A.3, we observe that models with widths 512 and 1024 only moderately increase prediction latency and are therefore good architecture candidates for the prediction step. Models with higher widths incur up to four times the prediction latency of ridge regression. However, in Figure A.4, we find that after training these architectures to convergence, their test error does not outperform ridge regression. The input to our MLPs were 2,000 random features from the QM7 dataset. We trained the MLPs to predict atomization energy by minimizing the mean squared error using the Adam optimizer. We searched over learning rate schedules and weight decay hyperparameters.

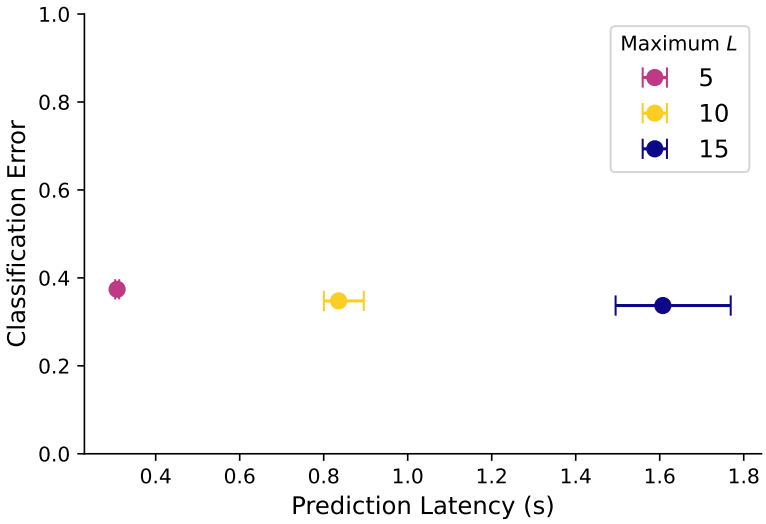


Figure A.2: When training classification models on the ModelNet40 dataset, we are able to trade off prediction accuracy for prediction latency by changing the maximum frequency of spherical harmonics L. We represent samples from the ModelNet40 dataset using point clouds of size N=1,024. Computing random features has computational complexity scaling quadratically in N, which significantly slows down prediction latency. However, the prediction error of our model is relatively insensitive to the maximum frequency L (see Figure A.7a), so we can greatly reduce the prediction latency by decreasing L. We evaluate prediction latency on a machine with 24 physical CPU cores. In the figure, the centers show the median latency sample, and the error bars span the 25^{th} and 75^{th} percentiles.

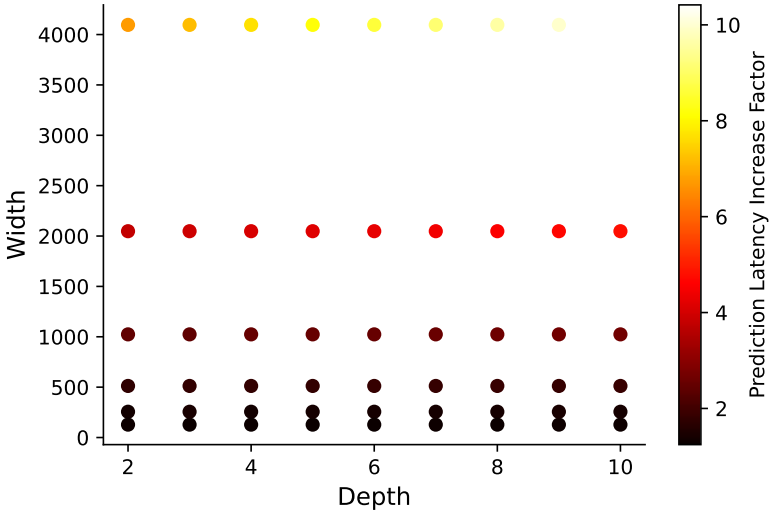
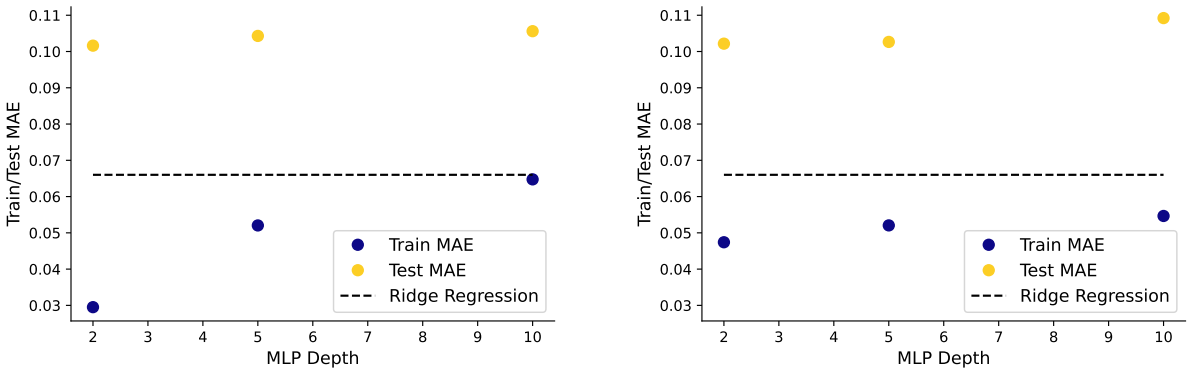


Figure A.3: We measure the prediction latency of computing 2,000 random features and predicting using different MLP architectures. We present increase in latency when compared to ridge regression. A value of 2.0 indicates that predictions take twice as long as predictions using ridge regression.



(a) MLPs with width = 512

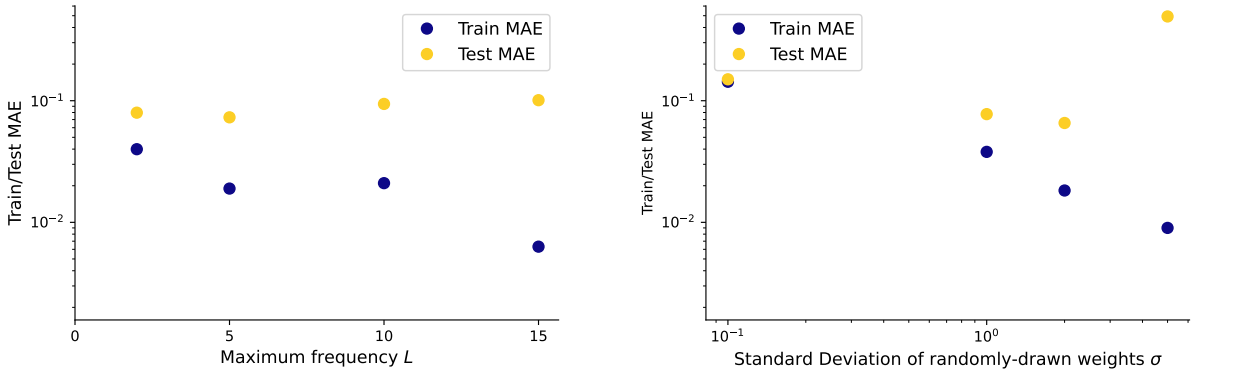
(b) MLPs with width = 1024

Figure A.4: The ridge regression baseline outperforms MLP architectures with low prediction latency.

A.5 Hyperparameter Sensitivity Analysis

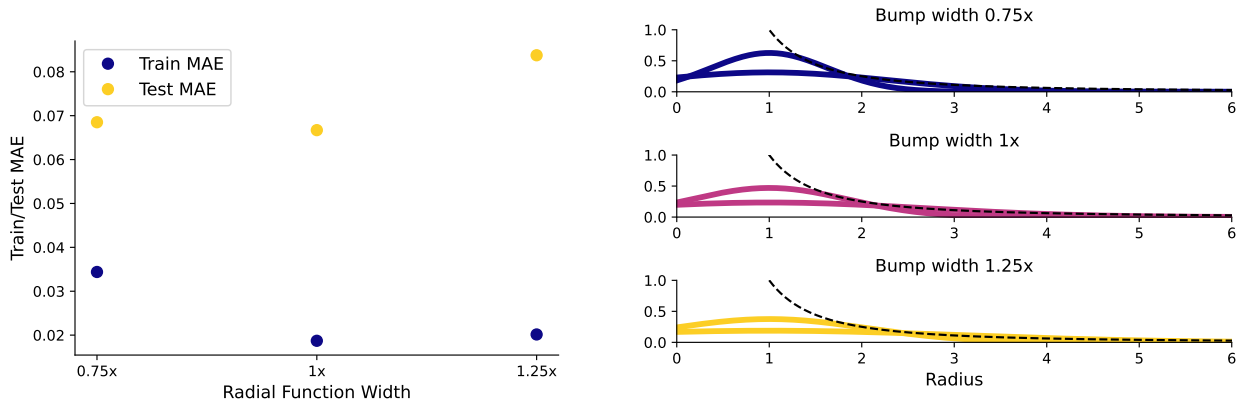
We perform multiple experiments to evaluate the sensitivity of our random feature models' performance. In Figures A.5 and A.6, we find that our models trained on the QM7 dataset are most sensitive to σ, the standard deviation of the random weights. In Figures A.7

and A.8, we find the same result for models trained on the ModelNet40 dataset.



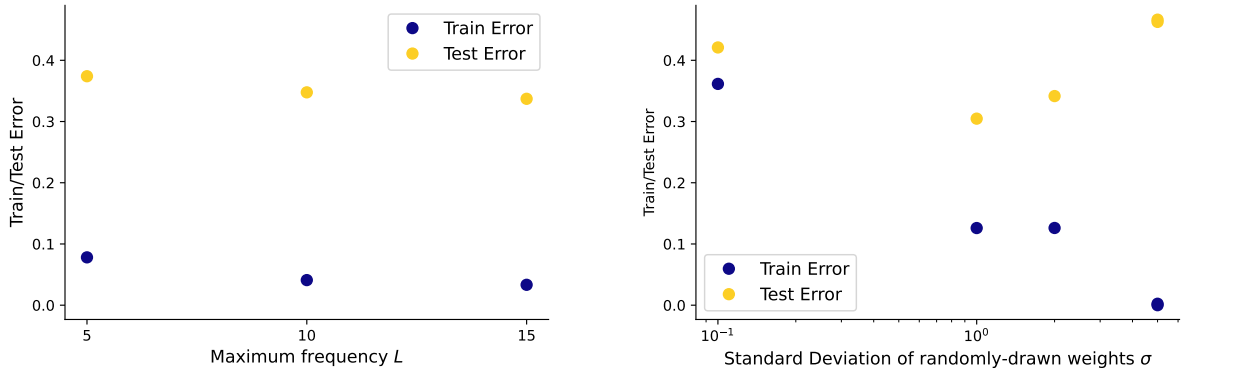
(a) Effect of maximum frequency L (b) Effect of random weight distribution $\mathcal{N}(0, \sigma^2)$

Figure A.5: In the QM7 experiments, the standard deviation of the weights is the most important hyperparameter. For this sensitivity experiment, we use a standard hyperparameter setting and vary one hyperparameter at a time. The standard setting is 2,000 random features; standard deviation of the random weights $\sigma = 2.0$; and the maximum frequency $L = 5$. In all experiments, we search over a pre-defined grid of L^2 regularization parameters, and select the best model using a held-out validation set. In Figure A.5a, we vary the maximum frequency of the spherical harmonics, and we see this does not have a large effect on the test error. In Figure A.5b, we vary σ, and we see this has a large effect on both the train and test error.



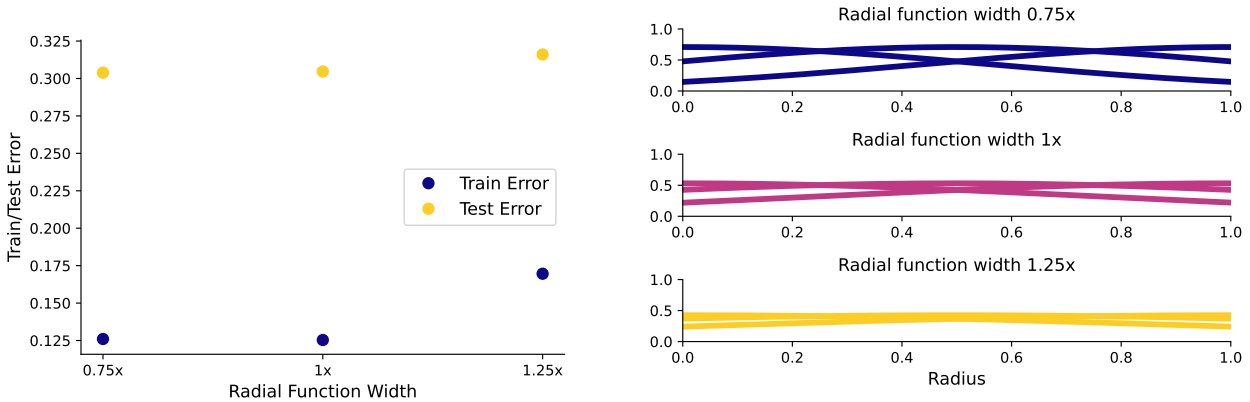
(a) Effect of radial function width (b) Visualizing different radial function widths

Figure A.6: Our results on the QM7 dataset are slightly sensitive to the width of the chosen radial functions. For the molecular energy regression experiments, our radial functions are two Gaussians centered at 1, with different widths $[\sigma_1, \sigma_2]$. We train and test models with different width scales: $0.75 \times [\sigma_1, \sigma_2]$ and $1.25 \times [\sigma_1, \sigma_2]$. We show the resulting train and test errors in Figure A.6a and visualize the resulting radial functions in Figure A.6b.



(a) Effect of maximum frequency L (b) Effect of random weight distribution $\mathcal{N}(0, \sigma^2)$

Figure A.7: In the ModelNet40 experiments, the standard deviation of the weights is the most important hyperparameter. For this sensitivity experiment, we use a standard hyperparameter setting and vary one hyperparameter at a time. The standard setting is 15,000 random features; standard deviation of the random weights $\sigma = 2.0$; and the maximum frequency $L = 10$. In Figure A.7a, we vary the maximum frequency of spherical harmonics L, and we see that increasing L slightly improves the performance. In Figure A.7b, we vary σ, and we see the model's performance is sensitive to this parameter.



(a) Effect of radial function width (b) Visualizing different radial function widths

Figure A.8: Our results on the ModelNet40 dataset are very slightly sensitive to the width of the chosen radial functions. Our original radial functions are three Gaussians with centers $[0, 0.5, 1.0]$ and width σ. We train and test models with different radial function widths: $0.75 \times \sigma$, $1 \times \sigma$, and $1.25 \times \sigma$. Figure A.8a shows the resulting train and test errors, and Figure A.8b shows the resulting radial functions.

A.6 Solving Ridge Regression Problems for Random Features

This section outlines numerical methods for solving ridge regression problems arising in our random features models. In general, we want to solve the following problem:

$$\underset{\beta}{\text{argmin}} \ \|\Phi\beta - y\|_2^2 + \lambda\|\beta\|_2^2 \tag{A.42}$$

In our problem instances, $\Phi \in \mathbb{R}^{n\times d}$ is the matrix of random features. Φ is a dense matrix; there is no structured sparsity. n is fixed; it is the number of samples in the dataset. The number of random features d can be treated as a hyperparameter, but we find that $d > n$ is required for optimal test error, so the system of equations is underdetermined. λ is the regularization parameter. Larger λ corresponds to more regularization. Generally, when solving a large system, we approximately know a good value for λ and want to keep it fixed.[1] To recap, we have an instance of a dense, overdetermined ridge regression problem.

By taking the gradient of Equation (A.42) with respect to β we arrive at the normal equation:

$$0 = \left(\Phi^\top\Phi + \lambda I\right)\beta^* - \Phi^\top y \tag{A.43}$$

A.6.1 Exact Solution via the Singular Value Decomposition

From Equation (A.43), we see that the solution to the ridge regression problem is

$$\begin{aligned}\beta^* &= \left(\Phi^\top\Phi + \lambda I\right)^{-1}\Phi^\top y \\ &= V\left(\Sigma^2 + \lambda I\right)^{-1}\Sigma^\top U^\top y\end{aligned} \tag{A.44}$$

1. As opposed to other optimization problems, where increasing λ is an acceptable method of improving the conditioning of the system.

where $\Phi = U\Sigma V^\top$ is the singular value decomposition (SVD). Because $\left(\Sigma^2 + \lambda I\right)$ is a diagonal matrix, we can exactly and efficiently compute its inverse. This suggests a method for solving Equation (A.42): compute the SVD of Φ, efficiently compute $\left(\Sigma^2 + \lambda I\right)^{-1}$, and reconstruct β^* via Equation (A.44). This solution method is stable and relatively performant for small problem instances ($n, d < 10,000$). We use this method in the QM7 experiments. Evaluating the full SVD of the feature matrix generated in the QM9 experiment is prohibitively costly, so we experimented with other solution methods.

A.6.2 Approximate Solution via Iterative Methods

Equation (A.42) can be treated as an optimization problem in the variables $\beta \in \mathbb{R}^d$. The objective is strongly convex, with smoothness $\sigma^2_{\max}(\Phi)$ and strong convexity parameter[2] λ. This motivates the application of standard iterative methods to find an approximate solution. Gradient descent, for instance, is known to enjoy linear convergence (on a log-log plot) for smooth and strongly-convex objectives. However, the rate of convergence is $\frac{\lambda}{\sigma^2_{\max}(\Phi)}$, which is very slow for typical problem instances.

A family of iterative algorithms known as Krylov subspace methods are designed for linear least-squares problems like Equation (A.42). These methods include the conjugate gradient method and variations. In the QM9 experiment, we use LSQR [Paige and Saunders, 1982a,b] implemented in scipy [Virtanen et al., 2020].

While they are empirically faster than applying gradient descent to the objective and are known to converge to the exact solution in a finite number of iterations, the conjugate gradient method and LSQR both suffer from poor convergence dependence on the quantity $\frac{\sigma^2_{\max}(\Phi)}{\lambda}$. We have found that on our problem instance, LSQR requires approximately 100,000 iterations to converge to a solution with low training error.

2. The strong convexity parameter is actually $\max\{\lambda, \sigma^2_{\min}(\Phi)\}$, but in practice, our random feature matrices are approximately low rank, so $\sigma^2_{\min}(\Phi) \ll \lambda$.

A.6.3 Approximate Solution via Principal Components Regression

We also attempt to approximately solve the ridge regression problem with principal components regression. We compute a truncated SVD with scipy's sparse linear algebra package [Virtanen et al., 2020]. The runtime of this operation depends on k, the number of singular vectors we choose to resolve. After computing the truncated SVD, we construct an approximate solution:

$$\hat{\beta}_k = V_k \left(\Sigma_k^2 + \lambda I\right)^{-1} \Sigma_k^\top U_k^\top y$$

Empirically, we find that we need to choose a large k to find a solution with low training error, which incurs a large runtime. We find LSQR arrives at an approximate solution faster than principal components regression.

A.6.4 Future Work

We have identified two different classes of methods from numerical linear algebra as promising candidates for improving our scaling to larger datasets.

Preconditioning Iterative Methods

The runtime of iterative methods like LSQR [Paige and Saunders, 1982a,b] depend adversely on the problem's conditioning $\frac{\sigma_{\max}^2(\Phi)}{\lambda}$. One way to improve the convergence of methods like LSQR is to find a preconditioning matrix $M \in \mathbb{R}^{n \times n}$ where $\sigma_{\max}^2(M\Phi) \ll \sigma_{\max}^2(\Phi)$. Once a preconditioner is formed, one solves the augmented problem

$$\underset{\beta}{\operatorname{argmin}} \ \|M\Phi\beta - My\|_2^2 + \lambda\|\beta\|_2^2$$

using just a few iterations of an iterative solver. There are a number of classical preconditioning techniques and review articles, including Wathen [2015], Benzi [2002].

LSRN [Meng et al., 2014] offers a promising method of computing a preconditioner designed for dense, overdetermined ridge regression problems. The algorithm constructs a preconditioner by subsampling columns of the matrix Φ and computing a singular value decomposition of the subsampled matrix. We believe that using a preconditioner like LSRN is a promising future research direction.

Approximate Solution via Sketching

Matrix sketching methods from the field of randomized numerical linear algebra [Drineas and Mahoney, 2016] aim to reduce the dimension of a linear system while approximately preserving the solution. This can be performed by randomly selecting rows of the (possibly preconditioned) matrix Φ and solving the smaller problem instance. Sketching algorithms specifically for overdetermined ridge regression have been introduced in recent years Chowdhury et al. [2018], Kacham and Woodruff [2022].

APPENDIX B

MUTLI-FREQUENCY PROGRESSIVE REFINEMENT FOR LEARNED INVERSE SCATTERING: ADDITIONAL RESULTS AND DETAILS

B.1 Distribution of Scattering Potentials

To generate samples from $\mathcal{D}$, our distribution of scattering potentials, we draw a random smoothly-varying background and three shapes with random sizes, positions, and rotations. This section provides details about the generation of these scattering objects.

The random low-frequency backgrounds were generated by drawing random Fourier coefficients and filtering out the high frequencies using $\mathsf{LPF}_{7.\bar{1}\pi}$. The resulting background was transformed to Cartesian coordinates, and then shifted and scaled so the maximum value was 2.0 and the minimum value was 0.0. The background was truncated to 0.0 outside of the disk of radius 0.4. Three shapes were randomly chosen among equilaterial triangles, squares, and ellipses. The three shapes had randomly-chosen centers and rotations, constrained to be non-overlapping and fit inside the disk of radius 0.4. The side lengths of the squares and triangles were uniformly sampled from $[0.1, 0.15]$. The major axis lengths of the ellipses were uniformly sampled from $[0.1, 0.15]$, and the minor axis lengths were uniformly sampled from $[0.05, 0.1]$. Finally, $\mathsf{LPF}_{32\pi}$ was applied to the scattering potential.

B.2 Hyperparameter Search

For our hyperparameter searches, we trained models on a grid of hyperparameters and evaluated them on a validation set every 5 epochs. We found the epoch and hyperparameter setting which produced the lowest error on the validation set, and used those model weights for final evaluation on a held-out test set.

B.2.1 FYNet and MFISNet Models

We train all models using the Adam algorithm, with a batch size of 16 samples. All of the FYNet, MFISNet-Parallel, MFISNet-Fused, and MFISNet-Refinement models tested have 3 1D convolutional layers followed by 3 2D convolutional layers; ReLU activations are used between layers. In the FYNet, MFISNet-Parallel and MFISNet-Refinement models, we use 1D and 2D convolutional kernels with 24 channels following [Fan and Ying, 2022]; in the MFISNet-Fused models, we use 1D and 2D convolutional kernels with $24N_k$ channels to adjust for the increased input size. To train MFISNet-Fused, MFISNet-Parallel and MFISNet-Refinement, we search over a grid of architecture and optimization hyperparameters. We report the optimal hyperparameters we found in Tables B.1 to B.4 and B.6.

The FYNet model and all MFISNet models use no input or output normalization. The parameters for the 1D convolutional layers were initialized by drawing from a uniform distribution $U[0, 2/d_{in}]$, where d_{in} is the input dimension for a given layer. The parameters for the 2D convolutional layers were initialized by the standard initialiaztion scheme for 2D convolutional layers in `Pytorch`, We performed preliminary investigation into different initialization schemes and found that the initialization scheme had a smaller effect than architecture or optimization hyperparameters. As a result, we decided to restrict our hyperparameter search to the architecture and optimization hyperparameters.

1d kernel size This is the number of frequency components in the 1D convolutional filters emulating F_k^*. We search over values $\{20, 40, 60\}$.

2d kernel size This is the size (in pixels) of the 2D convolutional kernel used in the layers emulating $\left(F_k^* F_k + \mu I\right)^{-1}$. We search over values $\{5, 7\}$.

Weight decay The weight decay parameter adds an ℓ_2 weight regularization term to the loss function. This hyperparameter determines the coefficient of this regularization term. We search over values $\{0.0, 1 \times 10^{-3}\}$.

Learning rate This is the step size for the Adam optimization algorithm. We search over values $\{1 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}\}$.

LR decrease For MFISNet-Refinement models only, we decrease the learning rate each time we begin training a new network block. This parameter determines the multiplicative decrease that we apply to the learning rate. We search over values $\{1.0, 0.25\}$.

Hyperparameter	**Data Setting (FYNet)**	
	$\delta = 0.0; N_k = 1$	$\delta = 0.1; N_k = 1$
1d kernel size	60	60
2d kernel size	5	5
Weight decay	0.0	0.0
Learning Rate	1×10^{-3}	1×10^{-3}

Table B.1: Optimal hyperparameters for FYNet.

Hyperparameter	**Data Setting (MFISNet-Refinement)**							
	$N_k = 2$	$N_k = 3$	$N_k = 4$	$N_k = 5$	$N_k = 2$	$N_k = 3$	$N_k = 4$	$N_k = 5$
	$\delta = 0.0$				$\delta = 0.1$			
1d kernel size	40	20	40	40	40	20	40	20
2d kernel size	7	5	5	5	5	5	5	5
Weight decay	1×10^{-3}	0.0	1×10^{-3}	0.0	0.0	0.0	1×10^{-3}	1×10^{-3}
Learning rate	5×10^{-4}	5×10^{-4}	1×10^{-3}	5×10^{-4}	1×10^{-3}	5×10^{-4}	1×10^{-4}	1×10^{-4}
LR decrease	0.25	1.0	0.25	1.0	0.25	1.0	1.0	1.0

Table B.2: Optimal hyperparameters for MFISNet-Refinement.

Hyperparameter	**Data Setting (MFISNet-Parallel)**							
	$N_k = 2$	$N_k = 3$	$N_k = 4$	$N_k = 5$	$N_k = 2$	$N_k = 3$	$N_k = 4$	$N_k = 5$
	$\delta = 0.0$				$\delta = 0.1$			
1d kernel size	40	20	20	20	40	40	20	20
2d kernel size	5	7	5	7	5	5	7	7
Weight decay	0.0	1×10^{-3}	0.0	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	0.0
Learning rate	5×10^{-4}	5×10^{-4}	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	5×10^{-4}

Table B.3: Optimal hyperparameters for MFISNet-Parallel.

Hyperparameter	Data Setting (MFISNet-Fused)							
	$N_k = 2$	$N_k = 3$	$N_k = 4$	$N_k = 5$	$N_k = 2$	$N_k = 3$	$N_k = 4$	$N_k = 5$
	$\delta = 0.0$				$\delta = 0.1$			
1d kernel size	40	20	20	40	20	20	20	40
2d kernel size	5	5	5	5	7	5	5	5
Weight decay	0	1×10^{-3}	0	0	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}
Learning Rate	5×10^{-4}	5×10^{-4}	1×10^{-4}	1×10^{-3}	1×10^{-3}	5×10^{-4}	1×10^{-4}	1×10^{-3}

Table B.4: Optimal hyperparameters for MFISNet-Fused.

Hyperparameter	Data Setting (MFISNet-Refinement) with No Progressive Refinement Training			
	$N_k = 2$	$N_k = 3$	$N_k = 4$	$N_k = 5$
1d kernel size	40	20	40	40
2d kernel size	5	5	5	7
Weight decay	0.0	1×10^{-3}	1×10^{-3}	0.0
Learning rate	5×10^{-4}	5×10^{-4}	1×10^{-4}	1×10^{-4}

Table B.5: Optimal hyperparameters for the MFISNet-Refinement models trained with the "No Progressive Refinement" training condition (Section 3.5.5). All of the models were trained on noiseless training samples.

Hyperparameter	Data Setting (MFISNet-Refinement) with No Homotopy through Frequency Training			
	$N_k = 2$	$N_k = 3$	$N_k = 4$	$N_k = 5$
1d kernel size	40	20	40	40
2d kernel size	5	7	7	7
Weight decay	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}
Learning rate	5×10^{-4}	5×10^{-4}	5×10^{-4}	1×10^{-4}
γ	1.0	1.0	1.1	1.1

Table B.6: Optimal hyperparameters for the MFISNet-Refinement models trained with the "No Homotopy through Frequency" training condition (Section 3.5.5). Here, γ is the factor which weights different loss terms (cf. (3.18)). We searched over values $\gamma = \{0.9, 1.0, 1.1\}$. All of the models were trained on noiseless training samples.

B.2.2 Wide-Band Butterfly Network

To find the optimal Wide-Band Butterfly Network, we optimized over the following hyperparameters. We defined the grid of hyperparameters by taking the original hyperparameter

from Li et al. [2022] and adding both higher and lower values, where possible. See Table B.7 for the selected values.

Rank This parameter controls the rank of the compression of local patches in the butterfly factorization. Increasing this rank parameter increases the number of learnable parameters in the part of the network emulating F_k^*. We found that increasing the rank decreased the train and validation errors, and we increased the rank until we were unable to fit the model and data onto a single GPU. We searched over values $\{2, 3, 5, 10, 15, 20, 30, 50\}$.

Initial Learning Rate We decrease the learning rate by a multiplicative factor after $2,000$ minibatches, as suggested by Li et al. [2022]. This parameter is the initial learning rate for the optimization algorithm. We searched over values $\{5\times10^{-4}, 1\times10^{-3}, 5\times10^{-3}\}$.

Learning Rate Decay This is the multiplicative decay parameter for the learning rate schedule. We searched over values $\{0.85, 0.95\}$.

Sigma Li et al. [2022] suggest training the network to match slightly-filtered versions of the ground-truth q. This is performed by applying a Gaussian filter to the targets $q^{(i)}$ before training. Sigma is the standard deviation of this Gaussian filter. We do not blur the targets in the test set. We searched over values $\{0.75, 1.125, 1.5\}$.

Batch Size This is the number of samples per minibatch. We searched over values $\{16, 32\}$.

B.3 Training Method with Warm-Start Initialization

In this section, we give details of the "Algorithm 2 + Warm-Start Initialization" training procedure described in Section 3.5.6. This training procedure leverages the intuition that sequential blocks in our MFISNet-Refinement architecture learn similar functions. In Algorithm 11 we give full pseudocode for the warm-start training procedure. In this pseudocode,

Hyperparameter	Data Setting	
	$\delta = 0.0; N_k = 3$	$\delta = 0.1; N_k = 3$
Rank	50	50
Initial Learning Rate	5×10^{-4}	5×10^{-4}
Learning Rate Decay	0.85	0.95
Sigma	1.5	1.5
Batch Size	16	32

Table B.7: Optimal hyperparameters for Wide-Band Butterfly Networks.

we refer to the trainable parameters in block t as θ_t. If $t > 1$, these parameters contain the parameters for the FYNet block, which we call $\theta_{t,1}$ and the parameters for the extra filtering layers, which we call $\theta_{t,2}$.

Algorithm 11: Training Procedure

Input: Training data samples $\mathcal{D}_n := \left\{ (q^{(j)}, d_{k_1}^{(j)}, \ldots, d_{k_{N_k}}^{(j)}) \right\}_{j=1}^{n}$.

for $t = 1, ..., N_k$ **do**
  if $t = 1$ **then**
    Initialize $t_{1,1}$ randomly
  else if $t = 2$ **then**
    Initialize $\theta_{2,1}$ with $\theta_{1,1}$
    Initialize $\theta_{2,2}$ randomly
  else
    Initialize $\theta_{t,1}$ with $\theta_{t-1,1}$
    Initialize $\theta_{t,2}$ with $\theta_{t-1,2}$
  Set θ_t as trainable, and freeze all other weights
  if $t < N_k$ **then**
    Train θ_t by optimizing L_t // Equation (3.14)
  else
    Train θ_t by optimizing $\|\hat{q}_{k_{N_k}} - q\|_2^2$
Set all weights as trainable
Train all weights by optimizing $\|\hat{q}_{k_{N_k}} - q\|_2^2$

Result: Trained neural network parameters $\{\theta_1, ..., \theta_{N_k}\}$.

B.4 Additional empirical results

In this section, we illustrate the predictions generated by the different models used in our experiments on randomly-selected test samples from our dataset. For each prediction, we include the associated error plot. In Figure B.1, we provide more samples comparing FYNet, Wide-Band Butterfly Network, and MFISNet-Refinement. In Figure B.2, we show sample outputs from MFISNet-Fused and MFISNet-Parallel.

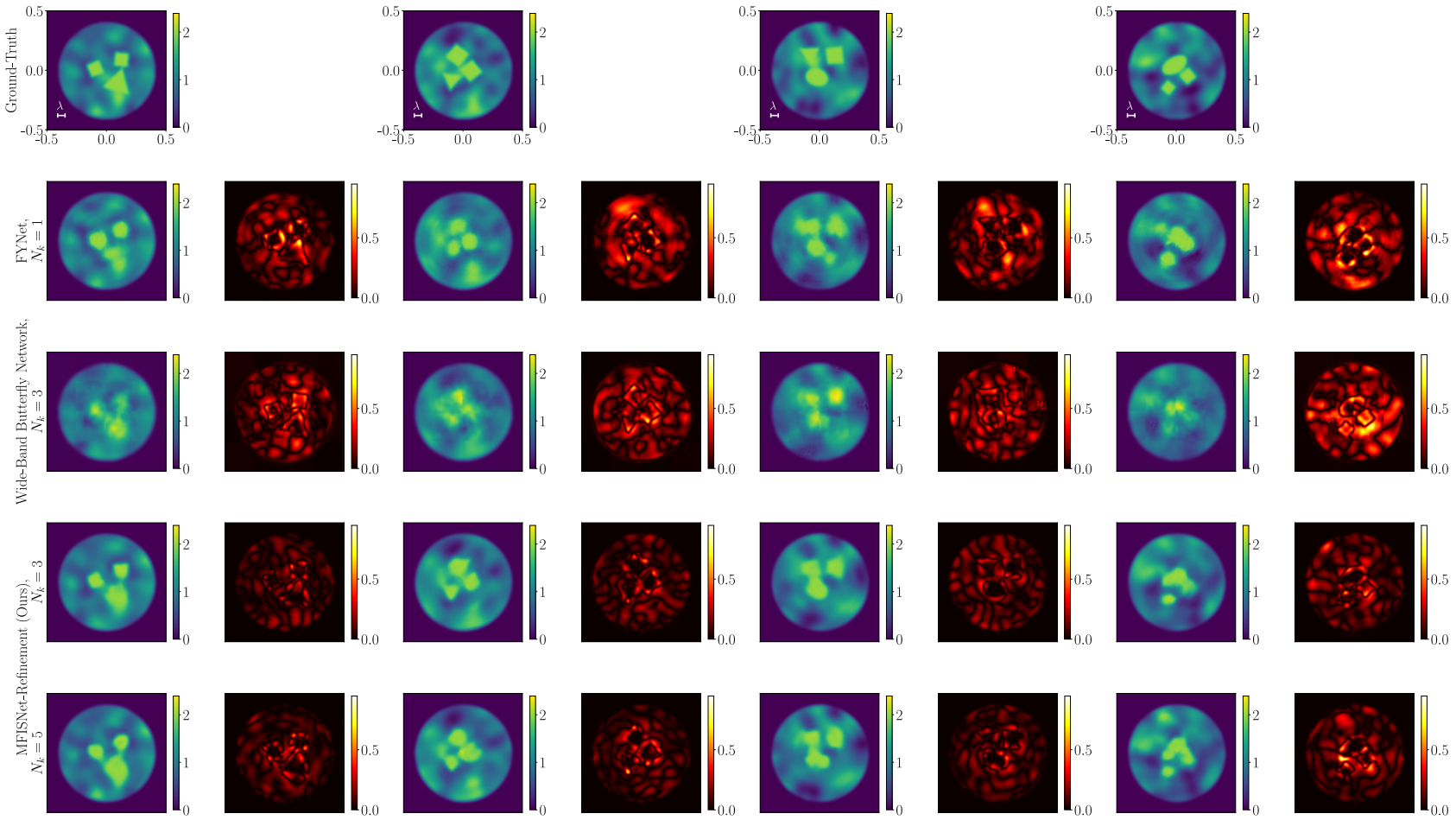


Figure B.1: Sample predictions from four randomly-selected test samples. The first row shows the ground-truth scattering potential; in this plot we show the wavelength corresponding to the maximum frequency $k = 32\pi$.

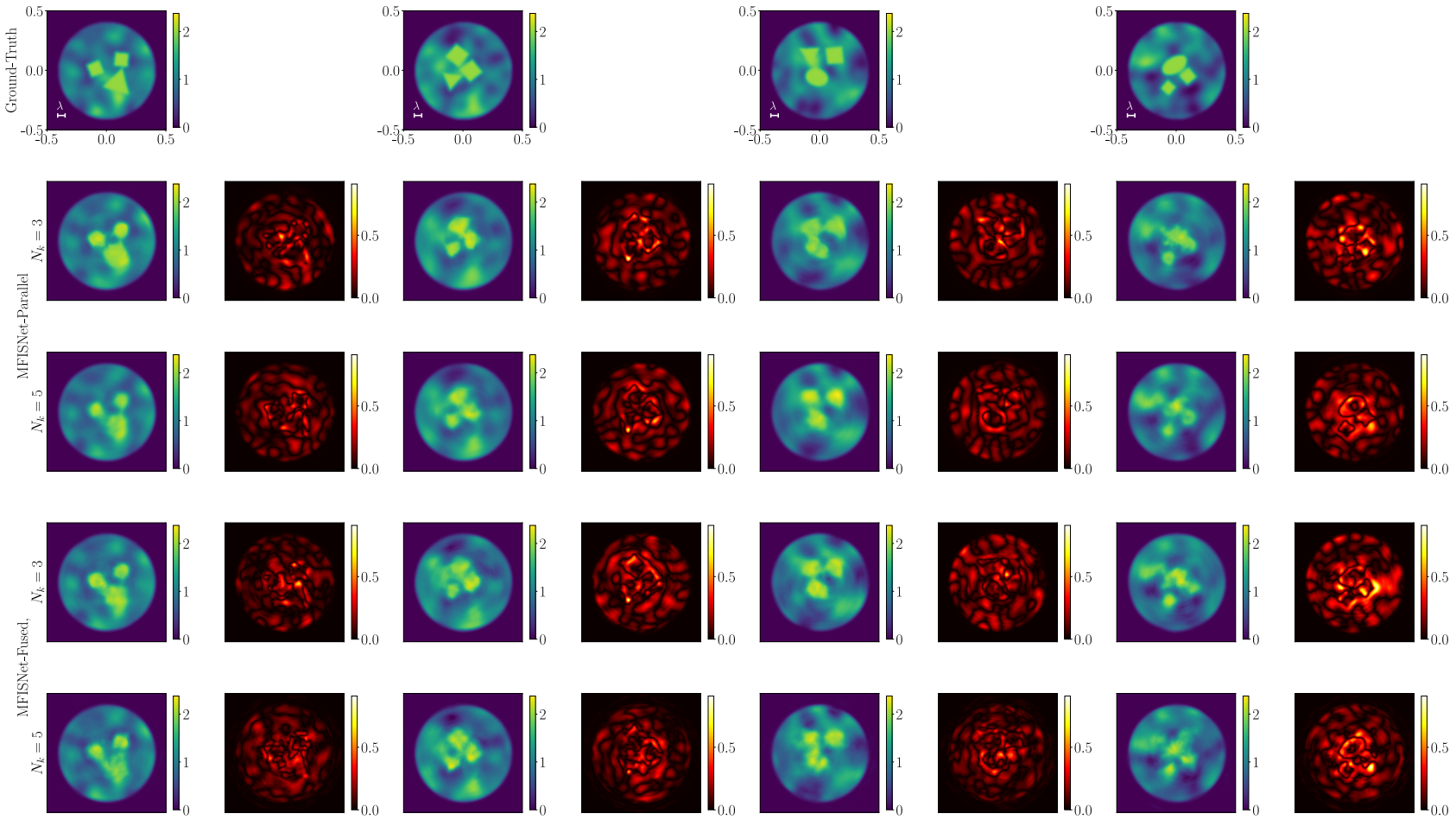


Figure B.2: Sample predictions from four randomly-selected test samples. The first row shows the ground-truth scattering potential; in this plot we show the wavelength corresponding to the maximum frequency $k = 32\pi$.

APPENDIX C

HPS ALGORITHM DETAILS

C.1 Full algorithms for 2D problems using DtN matrices

In this section, we describe the details for the two-dimensional version of our method which merges DtN matrices. In this version of the algorithm, the outgoing boundary data $\boldsymbol{h}$ tabulates the outward-pointing normal derivative of the particular solution, and the incoming boundary data $\boldsymbol{g}$ tabulates the homogenous solution values restricted to patch boundaries. $\boldsymbol{T}$ is a Dirichlet-to-Neumann matrix.

In this section, we use $\boldsymbol{I}_{a\times a}$ to denote the identity matrix of shape $a \times a$ and $\mathbf{0}_d$ to denote a length-d vector filled with 0's. When defining matrices blockwise, we use $\mathbf{0}$ to denote a block filled with 0's, and assume the shape of the block can be determined from the nonzero blocks sharing the same rows and columns.

C.1.1 Local solve stage

Recall from Section 4.3.1 that we use a tensor product of order-p Chebyshev–Lobatto points to discretize the interior of each leaf. This results in a grid with p^2 discretization points, and $4p-4$ of these points lie on the boundary of the leaf. We use order-q Gauss–Legendre points to discretize each side of the leaf's boundary, so there are $4q$ boundary points in total. Thus, to translate the information between the interior and boundary of each leaf, we need to compute spectral differentiation matrices and matrices interpolating between the p Chebyshev and q Gauss points. In particular, we need to precompute the following matrices:

- $\boldsymbol{P}$, with shape $4p-4\times 4q$, is the operator mapping data sampled on the Gauss boundary points to data sampled on the $4p-4$ Chebyshev points located on the boundary of the leaf. This matrix is constructed using a barycentric Lagrange interpolation matrix mapping from Gauss to Chebyshev points on one side of the leaf; this interpolation

matrix is repeated for the other sides. Rows corresponding to the Chebyshev points on the corners of the leaf average the contribution from the two adjoining panels.

- $\boldsymbol{Q}$, with shape $4q \times p^2$, performs spectral differentiation on the p^2 Chebyshev points followed by interpolation to the Gauss boundary points. This matrix is formed by stacking the relevant rows of Chebyshev spectral differentiation matrices to form an operator which evaluates normal derivatives on the $4p-4$ boundary Chebyshev points, and then composing this differentiation operator with a matrix formed from barycentric Lagrange interpolation matrix blocks. These interpolation matrices each map from one Chebyshev panel to one Gauss panel.

To work with $\boldsymbol{L}^{(i)}$, the discretization of the differential operator on leaf i, it is useful to identify I_i and I_e, the sets of discretization points corresponding to the $(p-2)^2$ interior and $4p-4$ exterior Chebyshev points, respectively. Now, we can fully describe the local solve stage in Algorithm 12.

Algorithm 12: 2D DtN local solve stage.

Input: Discretized differential operators $\{\boldsymbol{L}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$; discretized source vectors $\{\boldsymbol{f}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$; precomputed interpolation and differentiation matrices $\boldsymbol{P}$ and $\boldsymbol{Q}$

for $i = 1, \ldots, n_{\text{leaves}}$ **do**

  Invert $\boldsymbol{L}^{(i)}(I_i, I_i)$

  $$\boldsymbol{Y}^{(i)} = \begin{bmatrix} \boldsymbol{I}_{4p-4 \times 4p-4} \\ -\left(\boldsymbol{L}^{(i)}(I_i, I_i)\right)^{-1} \boldsymbol{L}^{(i)}(I_i, I_e) \end{bmatrix} \boldsymbol{P}$$

  $$\boldsymbol{v}^{(i)} = \begin{bmatrix} \boldsymbol{0}_{4p-4} \\ -\left(\boldsymbol{L}^{(i)}(I_i, I_i)\right)^{-1} \boldsymbol{f}^{(i)}(I_i) \end{bmatrix}$$

  $$\boldsymbol{T}^{(i)} = \boldsymbol{Q}\boldsymbol{Y}^{(i)}$$

  $$\boldsymbol{h}^{(i)} = \boldsymbol{Q}\boldsymbol{v}^{(i)}$$

Result: Poincaré–Steklov matrices $\{\boldsymbol{T}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$; outgoing boundary data $\{\boldsymbol{h}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$; interior solution matrices $\{\boldsymbol{Y}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$, leaf-level particular solutions $\{\boldsymbol{v}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$

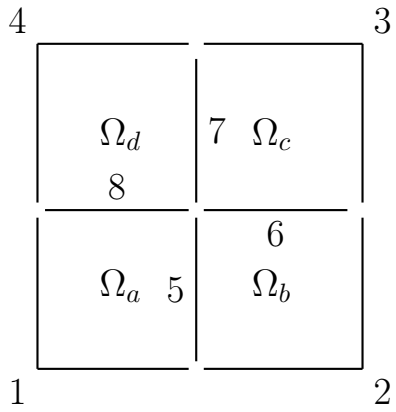


Figure C.1: Visualizing boundary elements 1 through 8 for two-dimensional merges.

C.1.2 Merge stage

In the 2D merge stage, we are merging four nodes $\Omega_a, \Omega_b, \Omega_c$, and Ω_d, which have *exterior* and *interior* discretization points. We label the exterior boundary sections $1, 2, 3$, and 4, and we label the interior boundary sections $5, 6, 7$, and 8. See Figure C.1 for a diagram of the different boundary parts. Because the merge stage operates completely on data discretized using Gauss–Legendre panels, there are no discretization points at the corners of nodes. This means each discretization point belongs to exactly one part of the boundary. During this stage of the algorithm, we will be indexing rows and columns of the Dirichlet-to-Neumann matrices according to these boundary sections. For example, we use $\boldsymbol{T}^{(a)}_{1,5}$ to indicate the submatrix of node a's DtN matrix which maps from boundary section 5 to boundary section 1. Suppose each side of $\Omega_a, \Omega_b, \Omega_c$, and Ω_d is discretized with n_{side} discretization points; in this case $\boldsymbol{T}^{(a)}_{1,5}$ will have shape $2n_{\text{side}} \times n_{\text{side}}$.

To implement the merge stage, we use sets of constraints to solve for a mapping from given $\boldsymbol{g}_{\text{ext}}$ to unknown $\boldsymbol{g}_{\text{int}}$. These are vectors tabulating the homogeneous solution along the exterior and interior boundary parts. First are constraints specifying that the solution to the PDE is continuous:

$$\boldsymbol{u}_{\text{ext}} = \boldsymbol{A}\boldsymbol{g}_{\text{ext}} + \boldsymbol{B}\boldsymbol{g}_{\text{int}} + \boldsymbol{h}^{(\text{child})}_{\text{ext}}. \tag{C.1}$$

In this equation $\boldsymbol{u}_{\text{ext}}$ is interpreted as the outward-pointing normal derivative of the solution to the PDE restricted to boundary elements $1, 2, 3$, and 4 with boundary data specified by $\boldsymbol{g}_{\text{ext}}$. In this set of constraints, we define:

$$\boldsymbol{h}_{\text{ext}}^{(\text{child})} = \begin{bmatrix} \boldsymbol{h}_1^{(a)} \\ \boldsymbol{h}_2^{(b)} \\ \boldsymbol{h}_3^{(c)} \\ \boldsymbol{h}_4^{(d)} \end{bmatrix}, \tag{C.2}$$

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{T}_{1,1}^{(a)} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{T}_{2,2}^{(b)} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{T}_{3,3}^{(c)} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{T}_{4,4}^{(d)} \end{bmatrix}, \tag{C.3}$$

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{T}_{1,5}^{(a)} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{T}_{1,8}^{(a)} \\ \boldsymbol{T}_{2,5}^{(b)} & \boldsymbol{T}_{2,6}^{(b)} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{T}_{3,6}^{(c)} & \boldsymbol{T}_{3,7}^{(c)} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{T}_{4,7}^{(d)} & \boldsymbol{T}_{4,8}^{(d)} \end{bmatrix}. \tag{C.4}$$

A second set of constraints enforces that the outward-pointing normal derivatives from neighboring nodes should sum to zero, which is equivalent to enforcing the continuity of the first derivative along the merge interfaces in their respective Cartesian directions. To that end, we use constraints $\boldsymbol{u}_5^{(a)} + \boldsymbol{u}_5^{(b)} = \boldsymbol{0}_{n_{\text{side}}}$, $\boldsymbol{u}_6^{(b)} + \boldsymbol{u}_6^{(c)} = \boldsymbol{0}_{n_{\text{side}}}$, and so on. This gives us an equation:

$$\boldsymbol{0}_{4n_{\text{side}}} = \boldsymbol{C}\boldsymbol{g}_{\text{ext}} + \boldsymbol{D}\boldsymbol{g}_{\text{int}} + \boldsymbol{h}_{\text{int}}^{(\text{child})}. \tag{C.5}$$

In Equation (C.5), we define

$$\boldsymbol{h}_{\text{int}}^{(\text{child})} = \begin{bmatrix} \boldsymbol{h}_5^{(a)} + \boldsymbol{h}_5^{(b)} \\ \boldsymbol{h}_6^{(b)} + \boldsymbol{h}_6^{(c)} \\ \boldsymbol{h}_7^{(c)} + \boldsymbol{h}_7^{(d)} \\ \boldsymbol{h}_8^{(d)} + \boldsymbol{h}_8^{(a)} \end{bmatrix}, \tag{C.6}$$

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{T}_{5,1}^{(a)} & \boldsymbol{T}_{5,2}^{(b)} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{T}_{6,2}^{(b)} & \boldsymbol{T}_{6,3}^{(c)} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{T}_{7,3}^{(c)} & \boldsymbol{T}_{7,4}^{(d)} \\ \boldsymbol{T}_{8,1}^{(a)} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{T}_{8,4}^{(d)} \end{bmatrix}, \tag{C.7}$$

$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{T}_{5,5}^{(a)} + \boldsymbol{T}_{5,5}^{(b)} & \boldsymbol{T}_{5,6}^{(b)} & \boldsymbol{0} & \boldsymbol{T}_{5,8}^{(a)} \\ \boldsymbol{T}_{6,5}^{(b)} & \boldsymbol{T}_{6,6}^{(b)} + \boldsymbol{T}_{6,6}^{(c)} & \boldsymbol{T}_{6,7}^{(c)} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{T}_{7,6}^{(c)} & \boldsymbol{T}_{7,7}^{(c)} + \boldsymbol{T}_{7,7}^{(d)} & \boldsymbol{T}_{7,8}^{(d)} \\ \boldsymbol{T}_{8,5}^{(a)} & \boldsymbol{0} & \boldsymbol{T}_{8,7}^{(d)} & \boldsymbol{T}_{8,8}^{(d)} + \boldsymbol{T}_{8,8}^{(a)} \end{bmatrix}. \tag{C.8}$$

Now that the matrices and vectors are defined, we can construct the linear system in Equation (4.3) and compute the merged data.

C.2 Full algorithms for 2D problems using ItI matrices

In this section, we describe the details for the two-dimensional version of our method which merges ItI matrices. In this version of the algorithm, the outgoing boundary data $\boldsymbol{h}$ tabulates the outgoing impedance data due to the particular solution, and the incoming boundary data $\boldsymbol{g}$ tabulates incoming impedance data due to the homogeneous solution. $\boldsymbol{T}$ is an impedance-to-impedance matrix. To define the impedance data, we need to choose a value $\eta \in \mathbb{R}_+$. In the wave scattering context, we often choose $\eta = k$ [Gillman et al., 2015].

As in the previous section, we use $\boldsymbol{I}_{a\times a}$ to denote the identity matrix of shape $a \times a$ and

$\mathbf{0}_d$ to denote a length-d vector filled with 0's. We also use $\mathbf{0}$ to denote a block filled with 0's, and assume the shape of this block can be inferred from the context.

C.2.1 Local solve stage

As before, there are $4p-4$ Chebyshev points on the boundary and $4q$ Gauss points on the boundary, and we must map between these two sets of discretization points. In particular, we need to precompute the following matrices:

- $\boldsymbol{P}$, with shape $4p-4 \times 4q$, is the operator mapping data sampled on the Gauss boundary points to data sampled on the $4p-4$ Chebyshev points located on the boundary of the leaf. This matrix is constructed using a barycentric Lagrange interpolation matrix mapping from Gauss to Chebyshev points on one side of the leaf with the final row deleted; this interpolation matrix is repeated for the other sides.

- $\boldsymbol{Q}$, with shape $4q \times 4p$, is the operator mapping data sampled on the Chebyshev points located on the boundary of the leaf to the Gauss points on the boundary of the leaf. This matrix is block-diagonal with four copies of a barycentric Lagrange interpolation matrix mapping from Chebyshev to Gauss points on one side of the leaf. Note this matrix double-counts the Chebyshev points at the corners of the leaf.

- $\boldsymbol{N}$, with shape $4p \times p^2$, is an operator mapping from interior solutions to outward-pointing normal derivative data evaluated on the boundary Chebyshev points. Note this operator counts each corner point twice. This matrix is formed by stacking relevant rows of Chebyshev spectral differentiation matrices.

- $\tilde{\boldsymbol{N}}$, with shape $4p-4 \times p^2$, is an operator mapping from interior solutions to outward-pointing normal data evaluated on the boundary Chebyshev points. Note this operator counts each corner point only once. This matrix is formed by stacking relevant rows of Chebyshev spectral differentiation matrices.

- $\boldsymbol{H}$, with shape $4p \times p^2$, is an operator mapping from interior solutions to evaluations of outgoing impedance data on the Chebyshev boundary discretization points. This matrix is constructed by taking $\boldsymbol{N}$ and subtracting $i\eta \boldsymbol{I}_{p\times p}$ from the appropriate submatrices. Again, this matrix double-counts the Chebyshev discretization points at the corners of the leaf.

- $\boldsymbol{G}$, with shape $4p - 4 \times p^2$, is an operator mapping from interior solutions to evaluations of incoming impedance data on the Chebyshev boundary discretization points. This matrix is constructed by taking $\tilde{\boldsymbol{N}}$ and adding $i\eta \boldsymbol{I}_{p-1\times p-1}$ to the appropriate submatrices.

To work with $\boldsymbol{L}^{(i)}$, the discretization of the differential operator on leaf i, it is useful to identify I_i and I_e, the sets of discretization points corresponding to the $(p-2)^2$ interior and $4p-4$ exterior Chebyshev points, respectively. As in Appendix C.1, we solve the local problem by using these precomputed operators to enforce the differential operator on the interior discretization points and the boundary condition on the boundary discretization points. In this case, the boundary condition is an incoming impedance condition, also known as a Robin boundary condition. Now, we can fully describe the local solve stage in Algorithm 13.

C.2.2 Merge stage

As in Appendix C.1, we are merging four nodes $\Omega_a, \Omega_b, \Omega_c$, and Ω_d with boundary parts labeled $1, 2, \ldots, 8$. See Figure C.1 for a diagram of the different boundary parts.

To implement the merge stage, we use sets of constraints to solve for a mapping from given $\boldsymbol{g}_{\text{ext}}$ to unknown $\boldsymbol{g}_{\text{int}}$. These are vectors tabulating incoming impedance data due to the homogeneous solution along the boundary parts. Because $\boldsymbol{g}_{\text{int}}$ tabulates impedance data, we must represent the incoming data with respect to neighboring nodes separately. For example, we must represent $\boldsymbol{g}_5^{(a)}$, the data along boundary element 5 incoming to node a,

Algorithm 13: 2D ItI local solve stage.

Input: Discretized differential operators $\{\boldsymbol{L}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$; discretized source functions $\{\boldsymbol{f}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$; precomputed interpolation and differentiation matrices $\boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{H}$, and $\boldsymbol{G}$.

for $i = 1, \ldots, n_{\text{leaves}}$ **do**

  $\boldsymbol{B}^{(i)} = \begin{bmatrix} \boldsymbol{G} \\ \boldsymbol{L}^{(i)}(I_i, :) \end{bmatrix}$

  Invert $\boldsymbol{B}^{(i)}$

  $\boldsymbol{Y}^{(i)} = \left(\boldsymbol{B}^{(i)}\right)^{-1}(:, I_e)\,\boldsymbol{P}$

  $\boldsymbol{v}^{(i)} = \left(\boldsymbol{B}^{(i)}\right)^{-1}(:, I_i)\,\boldsymbol{f}^{(i)}(I_i)$

  $\boldsymbol{T}^{(i)} = \boldsymbol{Q}\boldsymbol{H}\boldsymbol{Y}^{(i)}$

  $\boldsymbol{h}^{(i)} = \boldsymbol{Q}\boldsymbol{H}\boldsymbol{v}^{(i)}$

Result: Poincaré–Steklov matrices $\{\boldsymbol{T}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$; outgoing boundary data $\{\boldsymbol{h}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$; interior solution matrices $\{\boldsymbol{Y}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$, leaf-level particular solutions $\{\boldsymbol{v}^{(i)}\}_{i=1}^{n_{\text{leaves}}}$

separately from $\boldsymbol{g}_5^{(b)}$. To that end, we use $\boldsymbol{g}_{\text{int}} = \left[\boldsymbol{g}_5^{(a)}, \boldsymbol{g}_8^{(a)}, \boldsymbol{g}_6^{(c)}, \boldsymbol{g}_7^{(c)}, \boldsymbol{g}_5^{(b)}, \boldsymbol{g}_6^{(b)}, \boldsymbol{g}_7^{(d)}, \boldsymbol{g}_8^{(d)}\right]^\top$. The ordering of these boundary elements is chosen specifically to reduce the computation during the merge, which will become apparent later. Again, we use n_{side} to denote the number of discretization points along each side of the nodes being merged, so $\boldsymbol{g}_{\text{int}}$ has length $8n_{\text{side}}$.

The first set of constraints specify that the solution to the PDE is continuous:

$$\boldsymbol{u}_{\text{ext}} = \boldsymbol{A}\boldsymbol{g}_{\text{ext}} + \boldsymbol{B}\boldsymbol{g}_{\text{int}} + \boldsymbol{h}_{\text{ext}}^{(\text{child})}. \tag{C.9}$$

In this equation $\boldsymbol{u}_{\text{ext}}$ is interpreted as the outgoing impedance data of the solution to the PDE restricted to the merged nodes with boundary data specified by $\boldsymbol{g}_{\text{ext}}$. In this set of constraints, we define:

$$\boldsymbol{h}_{\text{ext}}^{(\text{child})} = \begin{bmatrix} \boldsymbol{h}_1^{(a)} \\ \boldsymbol{h}_2^{(b)} \\ \boldsymbol{h}_3^{(c)} \\ \boldsymbol{h}_4^{(d)} \end{bmatrix}, \tag{C.10}$$

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{T}_{1,1}^{(a)} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{T}_{2,2}^{(b)} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{3,3}^{(c)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{4,4}^{(d)} \end{bmatrix}, \tag{C.11}$$

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{T}_{1,5}^{(a)} & \boldsymbol{T}_{1,8}^{(a)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{2,5}^{(b)} & \boldsymbol{T}_{2,6}^{(b)} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{3,6}^{(c)} & \boldsymbol{T}_{3,7}^{(c)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{4,7}^{(d)} & \boldsymbol{T}_{4,8}^{(d)} \end{bmatrix}. \tag{C.12}$$

A second set of constraints specifies that the outgoing total solution's impedance data from one node must be opposite to the incoming homogeneous solution's impedance data for the neighboring node. For example, along merge interface 5, we enforce this constraint:

$$\mathbf{0}_{n_{\text{side}}} = \boldsymbol{u}_5^{(b)} + \boldsymbol{g}_5^{(a)}. \tag{C.13}$$

In this equation, $\boldsymbol{u}_5^{(b)}$ is the outgoing impedance data due to the total solution of the PDE restricted to the merged nodes with boundary condition $\boldsymbol{g}_{\text{ext}}$. The normal derivative is oriented relative to node b. We can expand $\boldsymbol{u}_5^{(b)}$ to find:

$$\mathbf{0}_{n_{\text{side}}} = \boldsymbol{g}_5^{(a)} + \boldsymbol{T}_{5,5}^{(a)}\boldsymbol{g}_5^{(a)} + \boldsymbol{T}_{5,8}^{(a)}\boldsymbol{g}_8^{(a)} + \boldsymbol{T}_{5,5}^{(b)}\boldsymbol{g}_5^{(b)} + \boldsymbol{T}_{5,6}^{(b)}\boldsymbol{g}_6^{(b)} + \boldsymbol{T}_{5,5}^{(a)}\boldsymbol{g}_5^{(a)} + \boldsymbol{T}_{5,1}^{(a)}\boldsymbol{g}_1^{(a)} + \boldsymbol{h}_5^{(a)}. \tag{C.14}$$

Similar equalities hold in each direction along each merge interface. We can expand to form a second system of constraints:

$$-\boldsymbol{h}_{\text{int}}^{(\text{child})} = \boldsymbol{C}\boldsymbol{g}_{\text{ext}} + \boldsymbol{D}\boldsymbol{g}_{\text{int}}, \tag{C.15}$$

where we define

$$\boldsymbol{h}_{\text{int}}^{(\text{child})} = \begin{bmatrix} \boldsymbol{h}_5^{(b)} \\ \boldsymbol{h}_8^{(d)} \\ \boldsymbol{h}_6^{(b)} \\ \boldsymbol{h}_7^{(d)} \\ \boldsymbol{h}_5^{(a)} \\ \boldsymbol{h}_6^{(c)} \\ \boldsymbol{h}_7^{(c)} \\ \boldsymbol{h}_8^{(a)} \end{bmatrix}, \tag{C.16}$$

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{T}_{5,2}^{(b)} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{T}_{8,4}^{(d)} \\ \boldsymbol{0} & \boldsymbol{T}_{6,2}^{(b)} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{T}_{7,4}^{(d)} \\ \boldsymbol{T}_{5,1}^{(a)} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{T}_{6,3}^{(c)} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{T}_{7,3}^{(c)} & \boldsymbol{0} \\ \boldsymbol{T}_{8,1}^{(a)} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}, \tag{C.17}$$

$$\boldsymbol{D} = \boldsymbol{I}_{8n_{\text{side}} \times 8n_{\text{side}}} + \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{T}_{5,5}^{(b)} & \boldsymbol{T}_{5,6}^{(b)} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{T}_{8,7}^{(d)} & \boldsymbol{T}_{8,8}^{(d)} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{T}_{6,5}^{(b)} & \boldsymbol{T}_{6,6}^{(b)} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{T}_{7,7}^{(d)} & \boldsymbol{T}_{7,8}^{(d)} \\ \boldsymbol{T}_{5,5}^{(a)} & \boldsymbol{T}_{5,8}^{(a)} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{T}_{6,6}^{(c)} & \boldsymbol{T}_{6,7}^{(c)} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{T}_{7,6}^{(c)} & \boldsymbol{T}_{7,7}^{(c)} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{T}_{8,5}^{(a)} & \boldsymbol{T}_{8,8}^{(a)} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}. \tag{C.18}$$

$\boldsymbol{D}$ has a special structure which allows us to efficiently compute $\boldsymbol{D}^{-1}$ via Schur complement methods. Note that we can re-write Equation (C.18) as a block matrix with 2×2 blocks:

$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{I}_{4n_{\text{side}} \times 4n_{\text{side}}} & \boldsymbol{D}_{12} \\ \boldsymbol{D}_{21} & \boldsymbol{I}_{4n_{\text{side}} \times 4n_{\text{side}}} \end{bmatrix}. \tag{C.19}$$

This structure allows us to construct $\boldsymbol{W} = \boldsymbol{I}_{4n_{\text{side}} \times 4n_{\text{side}}} - \boldsymbol{D}_{12}\boldsymbol{D}_{21}$, the Schur complement of the lower-right $\boldsymbol{I}_{4n_{\text{side}} \times 4n_{\text{side}}}$ block in $\boldsymbol{D}$; this is the only matrix we need to invert to compute $\boldsymbol{D}^{-1}$:

$$\boldsymbol{D}^{-1} = \begin{bmatrix} \boldsymbol{W}^{-1} & -\boldsymbol{W}^{-1}\boldsymbol{D}_{12} \\ -\boldsymbol{D}_{21}\boldsymbol{W}^{-1} & \boldsymbol{I}_{4n_{\text{side}} \times 4n_{\text{side}}} + \boldsymbol{D}_{21}\boldsymbol{W}^{-1}\boldsymbol{D}_{12} \end{bmatrix}. \tag{C.20}$$

We use Equation (C.20) to compute $\boldsymbol{D}^{-1}$ and then construct the outputs of the merge stage using Equations (4.4) and (4.5).

C.3 Full algorithms for 3D problems using DtN matrices with a uniform discretization

In this section, we describe the details for the three-dimensional version of our method which merges DtN matrices. In this version of the algorithm, the outgoing boundary data $\boldsymbol{h}$ tabulates the outward-pointing normal derivative of the particular solution, and the incoming boundary data $\boldsymbol{g}$ tabulates the homogenous solution values restricted to patch boundaries. $\boldsymbol{T}$ is a Dirichlet-to-Neumann matrix.

As in the previous section, we use $\boldsymbol{I}_{a\times a}$ to denote the identity matrix of shape $a \times a$ and $\mathbf{0}_d$ to denote a length-d vector filled with 0's. We also use $\mathbf{0}$ to denote a matrix block filled with 0's, and assume the shape of this block can be inferred from its context.

C.3.1 Local solve stage

Recall from Section 4.3.1 that we use a tensor product of order-p Chebyshev–Lobatto points to discretize the interior of each leaf. This results in a grid with p^3 discretization points, and $p^3 - (p-2)^3$ of these points lie on the boundary of the leaf. We use order-q Gauss–Legendre points to discretize each side of the leaf's boundary, so there are $6q^2$ boundary points in total. Thus, to translate the information between the interior and boundary of each leaf, we need to compute spectral differentiation matrices and matrices interpolating between the p^2 Chebyshev and q^2 Gauss points on each face of the leaf. In particular, we need to precompute the following matrices:

- $\boldsymbol{P}$, with shape $p^3 - (p-2)^3 \times 6q^2$, is the operator mapping data sampled on the Gauss boundary points to data sampled on the Chebyshev points located on the boundary of the leaf. This matrix is constructed using a barycentric Lagrange interpolation matrix mapping from Gauss to Chebyshev points on one face of the leaf; this interpolation matrix is repeated for the other sides. Rows corresponding to the Chebyshev points on

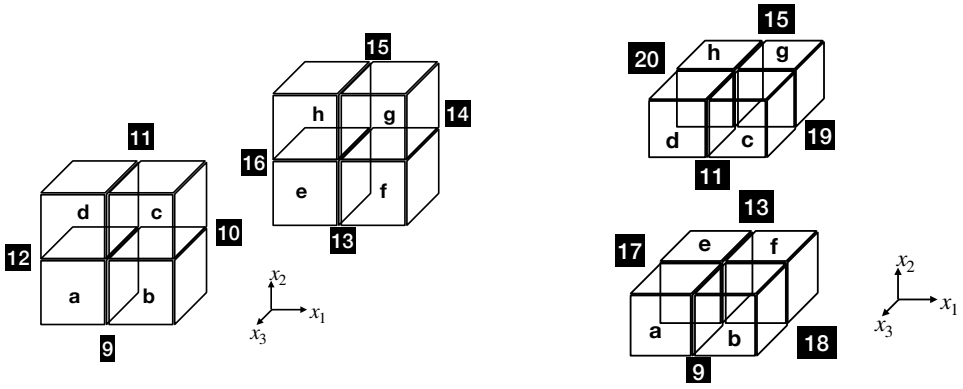


Figure C.2: Visualizing boundary elements 9 through 20 for three-dimensional merges.

the corners (edges) of the leaf average the contribution from the three (two) adjoining panels.

- $\boldsymbol{Q}$ with shape $6q^2 \times p^3$, performs spectral differentiation on the p^3 Chebyshev points followed by interpolation to the Gauss boundary points. This matrix is formed by stacking the relevant rows of Chebyshev spectral differentiation matrices to form an operator which evaluates normal derivatives on the boundary Chebyshev points, and then composing this differentiation operator with a matrix formed from barycentric Lagrange interpolation matrix blocks. These interpolation matrices each map from one Chebyshev face to one Gauss face.

To work with $\boldsymbol{L}^{(i)}$, the discretization of the differential operator on leaf i, it is useful to identify I_i and I_e, the sets of discretization points corresponding to the $(p-2)^3$ interior and $p^3 - (p-2)^3$ exterior Chebyshev points, respectively. Once the precomputed operators and index sets are correctly specified, Algorithm 12 can be re-used for the three-dimensional case.

C.3.2 Merge stage

In the 3D merge stage, we are merging eight nodes $\Omega_a, \Omega_b, \Omega_c, \Omega_d, \Omega_e, \Omega_f, \Omega_g$, and Ω_h, which have *exterior* and *interior* discretization points. We label the exterior boundary sections $1, 2, \ldots, 8$, and we label the interior boundary sections $9, 10, \ldots, 20$. See Figure C.2 for a diagram of the different boundary parts. Because the merge stage operates completely on data discretized using Gauss–Legendre panels, there are no discretization points at the corners or edges of nodes. This means each discretization point belongs to exactly one part of the boundary. During this stage of the algorithm, we will be indexing rows and columns of the Dirichlet-to-Neumann matrices according to boundary sections $1, \ldots, 20$. For example, we use $\boldsymbol{T}_{1,9}^{(a)}$ to indicate the submatrix of node a's DtN matrix which maps from boundary section 9 to boundary section 1. Suppose that each node has n_{side} discretization points on each face. Then $\boldsymbol{T}_{1,9}^{(a)}$ will have shape $3n_{\text{side}} \times n_{\text{side}}$.

Just as in the two-dimensional case, we use sets of constraints to solve for a mapping from given $\boldsymbol{g}_{\text{ext}}$ to unknown $\boldsymbol{g}_{\text{int}}$, vectors tabulating the homogeneous solution along the boundary parts. First are constraints specifying that the solution to the PDE is continuous:

$$\boldsymbol{u}_{\text{ext}} = \boldsymbol{A}\boldsymbol{g}_{\text{ext}} + \boldsymbol{B}\boldsymbol{g}_{\text{int}} + \boldsymbol{h}_{\text{ext}}^{(\text{child})}. \tag{C.21}$$

In this equation $\boldsymbol{u}_{\text{ext}}$ is interpreted as the outward-pointing normal derivative of the solution to the PDE restricted to the merged nodes with boundary data specified by $\boldsymbol{g}_{\text{ext}}$. In this set

of constraints, we define:

$$\boldsymbol{h}_{\text{ext}}^{(\text{child})} = \begin{bmatrix} \boldsymbol{h}_1^{(a)} \\ \boldsymbol{h}_2^{(b)} \\ \boldsymbol{h}_3^{(c)} \\ \boldsymbol{h}_4^{(d)} \\ \boldsymbol{h}_5^{(e)} \\ \boldsymbol{h}_6^{(f)} \\ \boldsymbol{h}_7^{(g)} \\ \boldsymbol{h}_8^{(h)} \end{bmatrix}, \tag{C.22}$$

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{T}_{1,1}^{(a)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{T}_{2,2}^{(b)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{3,3}^{(c)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{4,4}^{(d)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{5,5}^{(e)} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{6,6}^{(f)} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{7,7}^{(g)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{8,8}^{(h)} \end{bmatrix}, \tag{C.23}$$

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{T}_{1,9}^{(a)} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{1,12}^{(a)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{1,17}^{(a)} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{T}_{2,9}^{(b)} & \boldsymbol{T}_{2,10}^{(b)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{2,18}^{(b)} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{T}_{3,10}^{(c)} & \boldsymbol{T}_{3,11}^{(c)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{3,19}^{(c)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{4,11}^{(d)} & \boldsymbol{T}_{4,12}^{(d)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{4,20}^{(d)} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{5,13}^{(e)} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{5,16}^{(e)} & \boldsymbol{T}_{5,17}^{(e)} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{6,13}^{(f)} & \boldsymbol{T}_{6,14}^{(f)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{6,18}^{(f)} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{7,14}^{(g)} & \boldsymbol{T}_{7,15}^{(g)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{7,19}^{(g)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{8,15}^{(h)} & \boldsymbol{T}_{8,16}^{(h)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{8,20}^{(h)} \end{bmatrix}. \tag{C.24}$$

As in the two-dimensional case, we enforce constraints that ensure the normal derivatives

from neighboring nodes sum to zero, which gives us a system of constraints:

$$\mathbf{0}_{12n_{\text{side}}} = \boldsymbol{C}\boldsymbol{g}_{\text{ext}} + \boldsymbol{D}\boldsymbol{g}_{\text{int}} + \boldsymbol{h}_{\text{int}}^{(\text{child})}, \tag{C.25}$$

where we define

$$\boldsymbol{h}_{\text{int}}^{(\text{child})} = \begin{bmatrix} \boldsymbol{h}_9^{(a)} + \boldsymbol{h}_9^{(b)} \\ \boldsymbol{h}_{10}^{(b)} + \boldsymbol{h}_{10}^{(c)} \\ \boldsymbol{h}_{11}^{(c)} + \boldsymbol{h}_{11}^{(d)} \\ \boldsymbol{h}_{12}^{(d)} + \boldsymbol{h}_{12}^{(a)} \\ \boldsymbol{h}_{13}^{(e)} + \boldsymbol{h}_{13}^{(f)} \\ \boldsymbol{h}_{14}^{(f)} + \boldsymbol{h}_{14}^{(g)} \\ \boldsymbol{h}_{15}^{(g)} + \boldsymbol{h}_{15}^{(h)} \\ \boldsymbol{h}_{16}^{(h)} + \boldsymbol{h}_{16}^{(e)} \\ \boldsymbol{h}_{17}^{(a)} + \boldsymbol{h}_{17}^{(e)} \\ \boldsymbol{h}_{18}^{(b)} + \boldsymbol{h}_{18}^{(f)} \\ \boldsymbol{h}_{19}^{(c)} + \boldsymbol{h}_{19}^{(g)} \\ \boldsymbol{h}_{20}^{(d)} + \boldsymbol{h}_{20}^{(h)} \end{bmatrix}, \tag{C.26}$$

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{T}_{9,1}^{(a)} & \boldsymbol{T}_{9,2}^{(b)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{T}_{10,2}^{(b)} & \boldsymbol{T}_{10,3}^{(c)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{11,3}^{(c)} & \boldsymbol{T}_{11,4}^{(d)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{T}_{12,1}^{(a)} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{12,4}^{(d)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{13,5}^{(e)} & \boldsymbol{T}_{13,6}^{(f)} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{14,6}^{(f)} & \boldsymbol{T}_{14,7}^{(g)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{15,7}^{(g)} & \boldsymbol{T}_{15,8}^{(h)} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{16,5}^{(e)} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{16,8}^{(h)} \\ \boldsymbol{T}_{17,1}^{(a)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{17,5}^{(e)} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{T}_{18,2}^{(b)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{18,6}^{(f)} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{19,3}^{(c)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{19,7}^{(g)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{20,4}^{(d)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{20,8}^{(h)} \end{bmatrix}, \tag{C.27}$$

$$
\boldsymbol{D} = \begin{bmatrix}
\boldsymbol{T}^{(a)}_{9,9} + \boldsymbol{T}^{(b)}_{9,9} & \boldsymbol{T}^{(b)}_{9,10} & \mathbf{0} & \boldsymbol{T}^{(a)}_{9,12} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(a)}_{9,17} & \boldsymbol{T}^{(b)}_{9,18} & \mathbf{0} & \mathbf{0} \\
\boldsymbol{T}^{(b)}_{10,9} & \boldsymbol{T}^{(b)}_{10,10} + \boldsymbol{T}^{(c)}_{10,10} & \boldsymbol{T}^{(c)}_{10,11} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(b)}_{10,18} & \boldsymbol{T}^{(c)}_{10,19} & \mathbf{0} \\
\mathbf{0} & \boldsymbol{T}^{(c)}_{11,10} & \boldsymbol{T}^{(c)}_{11,11} + \boldsymbol{T}^{(d)}_{11,11} & \boldsymbol{T}^{(d)}_{11,12} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(c)}_{11,19} & \boldsymbol{T}^{(d)}_{11,20} \\
\boldsymbol{T}^{(a)}_{12,9} & \mathbf{0} & \boldsymbol{T}^{(d)}_{12,11} & \boldsymbol{T}^{(d)}_{12,12} + \boldsymbol{T}^{(a)}_{12,12} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(a)}_{12,17} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(d)}_{12,20} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(e)}_{13,13} + \boldsymbol{T}^{(f)}_{13,13} & \boldsymbol{T}^{(f)}_{13,14} & \mathbf{0} & \boldsymbol{T}^{(e)}_{13,16} & \boldsymbol{T}^{(e)}_{13,17} & \boldsymbol{T}^{(f)}_{13,18} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(f)}_{14,13} & \boldsymbol{T}^{(f)}_{14,14} + \boldsymbol{T}^{(g)}_{14,14} & \boldsymbol{T}^{(g)}_{14,15} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(f)}_{14,18} & \boldsymbol{T}^{(g)}_{14,19} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(g)}_{15,14} & \boldsymbol{T}^{(g)}_{15,15} + \boldsymbol{T}^{(h)}_{15,15} & \boldsymbol{T}^{(h)}_{15,16} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(g)}_{15,19} & \boldsymbol{T}^{(h)}_{15,20} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(e)}_{16,13} & \mathbf{0} & \boldsymbol{T}^{(h)}_{16,15} & \boldsymbol{T}^{(h)}_{16,16} + \boldsymbol{T}^{(e)}_{16,16} & \boldsymbol{T}^{(e)}_{16,17} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(h)}_{16,20} \\
\boldsymbol{T}^{(a)}_{17,9} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(a)}_{17,12} & \boldsymbol{T}^{(e)}_{17,13} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(e)}_{17,16} & \boldsymbol{T}^{(a)}_{17,17} + \boldsymbol{T}^{(e)}_{17,17} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\boldsymbol{T}^{(b)}_{18,9} & \boldsymbol{T}^{(b)}_{18,10} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(f)}_{18,13} & \boldsymbol{T}^{(f)}_{18,14} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(b)}_{18,18} + \boldsymbol{T}^{(f)}_{18,18} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \boldsymbol{T}^{(c)}_{19,10} & \boldsymbol{T}^{(c)}_{19,11} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(g)}_{19,14} & \boldsymbol{T}^{(g)}_{19,15} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(c)}_{19,19} + \boldsymbol{T}^{(g)}_{19,19} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(d)}_{20,11} & \boldsymbol{T}^{(d)}_{20,12} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(h)}_{20,15} & \boldsymbol{T}^{(h)}_{20,16} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T}^{(d)}_{20,20} + \boldsymbol{T}^{(h)}_{20,20}
\end{bmatrix}. \tag{C.28}
$$

Now that the matrices and vectors are defined, we can construct the linear system in Equation (4.3) and compute the merged data.

REFERENCES

Ahmad Abdelfattah, Pieter Ghysels, Wajih Boukaram, Stanimire Tomov, Xiaoye Sherry Li, and Jack Dongarra. Addressing irregular patterns of matrix computations on GPUs and their impact on applications powered by sparse direct solvers. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14, Dallas, TX, USA, November 2022. IEEE. ISBN 978-1-66545-444-5. doi:10.1109/SC41404.2022.00031. URL https://ieeexplore.ieee.org/document/10046092/.

Brandon Anderson, Truong Son Hy, and Risi Kondor. Cormorant: Covariant molecular neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/03573b32b2746e6e8ca98b9123f2249b-Paper.pdf.

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. PyTorch 2: Faster machine learning through dynamic Python bytecode transformation and graph compilation, April 2024. URL https://pytorch.org/assets/pytorch2-2.pdf. Publication Title: 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24).

Travis Askham, Manas Rachh, Michael O'Neil, Jeremy Hoskins, Daniel Fortunato, Shidong Jiang, Fredrik Fryklund, Tristan Goodwill, Hai Yang Wang, and Hai Zhu. `chunkIE`: A MATLAB integral equation toolbox, June 2024. URL https://github.com/fastalgorithms/chunkie.

Ivo M. Babuška and Stefan A. Sauter. Is the pollution effect of the fem avoidable for the helmholtz equation considering high wave numbers? *SIAM Journal on Numerical Analysis*, 34(6):2392–2423, December 1997. ISSN 0036-1429. doi:10.1137/S0036142994269186.

Gang Bao and Jun Liu. Numerical Solution of Inverse Scattering Problems with Multi-experimental Limited Aperture Data. *SIAM Journal on Scientific Computing*, 25(3): 1102–1117, January 2003. ISSN 1064-8275. doi:10.1137/S1064827502409705.

Albert P. Bartók, Risi Kondor, and Gábor Csányi. On representing chemical environments. *Physical Review B*, 87(18):184115, May 2013. ISSN 1098-0121, 1550-235X.

doi:10.1103/PhysRevB.87.184115. URL https://link.aps.org/doi/10.1103/PhysRevB.87.184115.

R. Beatson and Leslie Greengard. *A short course on fast multipole methods*, page 1–37. Numerical Mathematics and Scientific Computation. Oxford University Press, 1997.

Amir Beck and Marc Teboulle. A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, January 2009. ISSN 1936-4954. doi:10.1137/080716542.

Thomas Beck, Yaiza Canzani, and Jeremy Louis Marzuola. Quantitative bounds on impedance-to-impedance operators with applications to fast direct solvers for PDEs. *Pure and Applied Analysis*, 4(2):225–256, October 2022. ISSN 2578-5885. doi:10.2140/paa.2022.4.225. URL https://msp.org/paa/2022/4-2/p02.xhtml. Publisher: Mathematical Sciences Publishers.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi:10.1145/1553374.1553380.

Michele Benzi. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 182(2):418–477, Nov 2002. ISSN 0021-9991. doi:10.1006/jcph.2002.7176.

Jean-Pierre Berenger. A perfectly matched layer for the absorption of electromagnetic waves. *Journal of Computational Physics*, 114(2):185–200, October 1994. ISSN 0021-9991. doi:10.1006/jcph.1994.1159.

Lorenz C. Blum and Jean-Louis Reymond. 970 Million Druglike Small Molecules for Virtual Screening in the Chemical Universe Database GDB-13. *Journal of the American Chemical Society*, 131(25):8732–8733, July 2009. ISSN 0002-7863, 1520-5126. doi:10.1021/ja902302h. URL https://pubs.acs.org/doi/10.1021/ja902302h.

D.A. Boas, D.H. Brooks, E.L. Miller, C.A. DiMarzio, M. Kilmer, R.J. Gaudette, and Quan Zhang. Imaging the body with diffuse optical tomography. *IEEE Signal Processing Magazine*, 18(6):57–75, November 2001. ISSN 1558-0792. doi:10.1109/79.962278.

Alexander Bogatskiy, Timothy Hoffman, David W. Miller, and Jan T. Offermann. Pelican: Permutation equivariant and lorentz invariant or covariant aggregator network for particle physics. (arXiv:2211.00454), Dec 2022. URL http://arxiv.org/abs/2211.00454. arXiv:2211.00454 [hep-ex, physics:hep-ph].

Carlos Borges and George Biros. Reconstruction of a compactly supported sound profile in the presence of a random background medium. *Inverse Problems*, 34(11):115007, September 2018. ISSN 0266-5611. doi:10.1088/1361-6420/aadbc5.

Carlos Borges, Adrianna Gillman, and Leslie Greengard. High Resolution Inverse Scattering in Two Dimensions Using Recursive Linearization. *SIAM Journal on Imaging Sciences*, 10(2):641–664, January 2017. ISSN 1936-4954. doi:10.1137/16M1093562.

Max Born, Emil Wolf, and A. B. Bhatia. *Principles of optics: electromagnetic theory of propagation, interference and diffraction of light*. Cambridge University Press, Cambridge [England] ; New York, 7th (expanded) edition, 1999. ISBN 978-0-521-64222-4.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: Composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

Freddy Bunbury, Carlos Rivas, Victoria Calatrava, Amanda N. Shelton, Arthur Grossman, and Devaki Bhaya. Differential phototactic behavior of closely related cyanobacterial isolates from yellowstone hot spring biofilms. *Applied and Environmental Microbiology*, 88 (10):e00196–22, May 2022. ISSN 0099-2240, 1098-5336. doi:10.1128/aem.00196-22.

Keaton J. Burns, Geoffrey M. Vasil, Jeffrey S. Oishi, Daniel Lecoanet, and Benjamin P. Brown. Dedalus: A flexible framework for numerical simulations with spectral methods. *Physical Review Research*, 2(2):023068, April 2020. ISSN 2643-1564. doi:10.1103/PhysRevResearch.2.023068.

Matthew Burriesci and Devaki Bhaya. Tracking phototactic responses and modeling motility of *Synechocystis* sp. strain PCC6803. *Journal of Photochemistry and Photobiology B: Biology*, 91(2):77–86, May 2008. ISSN 1011-1344. doi:10.1016/j.jphotobiol.2008.01.012. URL https://www.sciencedirect.com/science/article/pii/S1011134408000365.

Vasileios Charisopoulos and Rebecca Willett. Nonlinear Tomographic Reconstruction via Nonsmooth Optimization. *SIAM Journal on Mathematics of Data Science*, pages 699–722, June 2025. doi:10.1137/24M1678982.

Chen Chen, Fenghua Tian, Hanli Liu, and Junzhou Huang. Diffuse Optical Tomography Enhanced by Clustered Sparsity for Functional Brain Imaging. *IEEE Transactions on Medical Imaging*, 33(12):2323–2331, December 2014. ISSN 1558-254X. doi:10.1109/TMI.2014.2338214.

Xudong Chen, Zhun Wei, Maokun Li, and Paolo Rocca. A review of deep learning approaches for inverse scattering problems (Invited review). *Progress In Electromagnetics Research*, 167:67–81, 2020. ISSN 1559-8985. doi:10.2528/PIER20030705.

Yu Chen. Recursive Linearization for Inverse Scattering. Technical Report YALEU/DCS/RR-1088, October 1995.

Yu Chen. Inverse scattering via Heisenberg's uncertainty principle. *Inverse Problems*, 13(2): 253, April 1997. ISSN 0266-5611. doi:10.1088/0266-5611/13/2/005.

Damyn Chipman. Ellipticforest: A direct solver library for ellipticpartial differential equations on adaptive meshes. *Journal of Open Source Software*, 9(96):6339, April 2024. ISSN 2475-9066. doi:10.21105/joss.06339.

Damyn Chipman, Donna Calhoun, and Carsten Burstedde. A fast direct solver for elliptic PDEs on a hierarchy of adaptively refined quadtrees, April 2024. URL http://arxiv.org/abs/2402.14936. arXiv:2402.14936 [cs, math].

Agniva Chowdhury, Jiasen Yang, and Petros Drineas. An iterative, sketching-based framework for ridge regression. In *Proceedings of the 35th International Conference on Machine Learning*, page 989–998. PMLR, Jul 2018. URL https://proceedings.mlr.press/v80/chowdhury18a.html.

Anders S Christensen, Lars A Bratholm, and Felix A Faber. FCHL revisited: Faster and more accurate quantum machine learning. *The Journal of Chemical Physics*, page 16, 2020.

Taco S. Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical CNNs. In *International Conference on Learning Representations*, 2018a. URL https://openreview.net/forum?id=Hkbd5xZRb.

Taco S. Cohen, Mario Geiger, and Maurice Weiler. Intertwiners between induced representations (with applications to the theory of equivariant neural networks), 2018b. URL https://arxiv.org/abs/1803.10743.

ATLAS Collaboration. The atlas experiment at the cern large hadron collider. *Journal of Instrumentation*, 3(08):S08003, August 2008. ISSN 1748-0221. doi:10.1088/1748-0221/3/08/S08003. URL https://dx.doi.org/10.1088/1748-0221/3/08/S08003.

José Colmenares, Jesús Ortiz, and Walter Rocchia. GPU linear and non-linear Poisson–Boltzmann solver module for DelPhi. *Bioinformatics*, 30(4):569–570, February 2014. ISSN 1367-4803. doi:10.1093/bioinformatics/btt699. URL https://doi.org/10.1093/bioinformatics/btt699.

David Colton and Rainer Kress. Looking Back on Inverse Scattering Theory. *SIAM Review*, 60(4):779–807, January 2018. ISSN 0036-1445, 1095-7200. doi:10.1137/17M1144763.

David L. Colton and Rainer Kress. *Inverse Acoustic and Electromagnetic Scattering Theory*. Number volume 93 in Applied Mathematical Sciences. Springer, New York, third edition edition, 2013. ISBN 978-1-4614-4941-6.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, page 16344–16359. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/67d57c32e20fd0a7a302cb81d36e40d5-Paper-Conference.pdf.

Congyue Deng, Or Litany, Yueqi Duan, Adrien Poulenard, Andrea Tagliasacchi, and Leonidas Guibas. Vector Neurons: A General Framework for SO(3)-Equivariant Networks. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 12180–12189, Montreal, QC, Canada, October 2021. IEEE. ISBN 978-1-66542-812-5. doi:10.1109/ICCV48922.2021.01198. URL https://ieeexplore.ieee.org/document/9711441/.

Samruddhi Deshmukh, Amartansh Dubey, and Ross Murch. Unrolled Optimization With Deep Learning-Based Priors for Phaseless Inverse Scattering Problems. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–14, 2022. ISSN 1558-0644. doi:10.1109/TGRS.2022.3214495. Conference Name: IEEE Transactions on Geoscience and Remote Sensing.

Kangning Diao, Zack Li, Richard D. P. Grumitt, and Yi Mao. `synax`: A differentiable and gpu-accelerated synchrotron simulation package. (arXiv:2410.01136), October 2024. doi:10.48550/arXiv.2410.01136. URL http://arxiv.org/abs/2410.01136. arXiv:2410.01136 [astro-ph].

Wen Ding, Kui Ren, and Lu Zhang. Coupling Deep Learning with Full Waveform Inversion, March 2022. URL http://arxiv.org/abs/2203.01799. arXiv:2203.01799 [cs, math].

Abdel Douiri, Martin Schweiger, Jason Riley, and Simon Arridge. Local diffusion regularization method for optical tomography reconstruction by using robust statistics. *Optics Letters*, 30(18):2439–2441, September 2005. ISSN 1539-4794. doi:10.1364/OL.30.002439.

Matthaios Doulgerakis-Kontoudis, Adam T. Eggebrecht, Stanislaw Wojtkiewicz, Joseph P. Culver, and Hamid Dehghani. Toward real-time diffuse optical tomography: Accelerating light propagation modeling employing parallel computing on GPU and CPU. *Journal of Biomedical Optics*, 22(12):125001, December 2017. ISSN 1083-3668, 1560-2281. doi:10.1117/1.JBO.22.12.125001.

Ralf Drautz. Atomic cluster expansion for accurate and transferable interatomic potentials. *Physical Review B*, 99(1):014104, January 2019. ISSN 2469-9950, 2469-9969. doi:10.1103/PhysRevB.99.014104. URL https://link.aps.org/doi/10.1103/PhysRevB.99.014104.

Petros Drineas and Michael W. Mahoney. Randnla: randomized numerical linear algebra. *Communications of the ACM*, 59(6):80–90, May 2016. ISSN 0001-0782. doi:10.1145/2842602.

O. G. Ernst and M. J. Gander. Why it is Difficult to Solve Helmholtz Problems with Classical Iterative Methods. In Ivan G. Graham, Thomas Y. Hou, Omar Lakkis, and Robert Scheichl, editors, *Numerical Analysis of Multiscale Problems*, pages 325–363. Springer, Berlin, Heidelberg, 2012. ISBN 978-3-642-22061-6. doi:10.1007/978-3-642-22061-6_10. URL https://doi.org/10.1007/978-3-642-22061-6_10.

Carlos Esteves, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. Learning SO(3) Equivariant Representations with Spherical CNNs, September 2018. URL http://arxiv.org/abs/1711.06721. arXiv:1711.06721 [cs].

Yuwei Fan and Lexing Ying. Solving inverse wave scattering with deep learning. *Annals of Mathematical Sciences and Applications*, 7(1):23–48, 2022.

Andreas Fichtner. *Full Seismic Waveform Modelling and Inversion*. Advances in Geophysical and Environmental Mechanics and Mathematics. Springer, Berlin, Heidelberg, 2011. ISBN 978-3-642-15806-3 978-3-642-15807-0. doi:10.1007/978-3-642-15807-0.

Daniel Fortunato. A high-order fast direct solver for surface PDEs. *SIAM Journal on Scientific Computing*, 46(4):A2582–A2606, August 2024. ISSN 1064-8275. doi:10.1137/22M1525259. URL https://epubs.siam.org/doi/abs/10.1137/22M1525259. Publisher: Society for Industrial and Applied Mathematics.

Daniel Fortunato, Nicholas Hale, and Alex Townsend. The ultraspherical spectral element method. *Journal of Computational Physics*, 436:110087, July 2021. ISSN 0021-9991. doi:10.1016/j.jcp.2020.110087.

Jean-Michel Geffrin, Pierre Sabouroux, and Christelle Eyraud. Free space experimental scattering database continuation: Experimental set-up and measurement precision. *Inverse Problems*, 21(6):S117, November 2005. ISSN 0266-5611. doi:10.1088/0266-5611/21/6/S09.

P. Geldermans and A. Gillman. An adaptive high order direct solution technique for elliptic boundary value problems. *SIAM Journal on Scientific Computing*, 41(1):A292–A315, January 2019. ISSN 1064-8275. doi:10.1137/17M1156320. URL https://epubs.siam.org/doi/abs/10.1137/17M1156320. Publisher: Society for Industrial and Applied Mathematics.

Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, April 1973. ISSN 0036-1429. doi:10.1137/0710032. URL https://epubs.siam.org/doi/10.1137/0710032. Publisher: Society for Industrial and Applied Mathematics.

Serban Georgescu, Peter Chow, and Hiroshi Okuda. GPU acceleration for FEM-based structural analysis. *Archives of Computational Methods in Engineering*, 20(2):111–121, June 2013. ISSN 1886-1784. doi:10.1007/s11831-013-9082-8. URL https://doi.org/10.1007/s11831-013-9082-8.

Pieter Ghysels and Ryan Synk. High performance sparse multifrontal solvers on modern GPUs. *Parallel Computing*, 110:102897, May 2022. ISSN 0167-8191. doi:10.1016/j.parco.2022.102897. URL https://www.sciencedirect.com/science/article/pii/S0167819122000059.

A P Gibson, J C Hebden, and S R Arridge. Recent advances in diffuse optical imaging. *Physics in Medicine & Biology*, 50(4):R1, February 2005. ISSN 0031-9155. doi:10.1088/0031-9155/50/4/R01.

A. Gillman and P. G. Martinsson. A direct solver with $O(N)$ complexity for variable coefficient elliptic PDEs discretized via a high-order composite spectral collocation method. *SIAM Journal on Scientific Computing*, 36(4):A2023–A2046, January 2014. ISSN 1064-8275. doi:10.1137/130918988. URL https://epubs.siam.org/doi/10.1137/130918988. Publisher: Society for Industrial and Applied Mathematics.

Adrianna Gillman, Alex H. Barnett, and Per-Gunnar Martinsson. A spectrally accurate direct solution technique for frequency-domain scattering problems with variable media. *BIT Numerical Mathematics*, 55(1):141–170, March 2015. ISSN 1572-9125. doi:10.1007/s10543-014-0499-8.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1263–1272. JMLR.org, 2017.

Shiqi Gong, Qi Meng, Jue Zhang, Huilin Qu, Congqiao Li, Sitian Qian, Weitao Du, Zhi-Ming Ma, and Tie-Yan Liu. An efficient lorentz equivariant graph neural network for jet tagging. *Journal of High Energy Physics*, 2022(7):30, Jul 2022. ISSN 1029-8479. doi:10.1007/JHEP07(2022)030. arXiv:2201.08187 [hep-ex, physics:hep-ph].

J. Andrew Grant, Barry T. Pickup, and Anthony Nicholls. A smooth permittivity function for Poisson–Boltzmann solvation methods. *Journal of Computational Chemistry*, 22(6): 608–640, 2001. ISSN 1096-987X. doi:10.1002/jcc.1032. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.1032.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. August 2024. URL https://openreview.net/forum?id=tEYskw1VY2#discussion.

Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep Learning for 3D Point Clouds: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(12):4338–4364, December 2021. ISSN 1939-3539. doi:10.1109/TPAMI.2020.3005434. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

Sijia Hao and Per-Gunnar Martinsson. A direct solver for elliptic PDEs in three dimensions based on hierarchical merging of Poincaré–Steklov operators. *Journal of Computational and Applied Mathematics*, 308:419–434, December 2016. ISSN 0377-0427. doi:10.1016/j.cam.2016.05.013. URL https://www.sciencedirect.com/science/article/pii/S0377042716302308.

Kenneth L. Ho and Leslie Greengard. A fast direct solver for structured linear systems by recursive skeletonization. *SIAM Journal on Scientific Computing*, 34(5):A2507–A2532, January 2012. ISSN 1064-8275. doi:10.1137/120866683. URL https://epubs.siam.org/doi/10.1137/120866683. Publisher: Society for Industrial and Applied Mathematics.

Jason Hu, Bowen Song, Xiaojian Xu, Liyue Shen, and Jeffrey A. Fessler. Learning Image Priors through Patch-based Diffusion Models for Solving Inverse Problems, October 2024.

Yao Huang, Wenrui Hao, and Guang Lin. Hompinns: Homotopy physics-informed neural networks for learning multiple solutions of nonlinear elliptic differential equations. *Computers & Mathematics with Applications*, 121:62–73, 2022. ISSN 0898-1221. doi:https://doi.org/10.1016/j.camwa.2022.07.002.

Peter Hähner and Thorsten Hohage. New stability estimates for the inverse acoustic inhomogeneous medium problem and applications. *SIAM Journal on Mathematical Analysis*, 33(3):670–685, January 2001. ISSN 0036-1410. doi:10.1137/S0036141001383564.

Steven L Jacques. Optical properties of biological tissues: A review. *Physics in Medicine & Biology*, 58(11):R37, May 2013. ISSN 0031-9155. doi:10.1088/0031-9155/58/11/R37.

Hanyang Jiang, Yuehaw Khoo, and Haizhao Yang. Reinforced inverse scattering. *SIAM Journal on Scientific Computing*, 46(6):B884–B902, December 2024. ISSN 1064-8275. doi:10.1137/22M153207X.

Ruoxi Jiang, Xiao Zhang, Karan Jakhar, Peter Y. Lu, Pedram Hassanzadeh, Michael Maire, and Rebecca Willett. Hierarchical implicit neural emulators. (arXiv:2506.04528), June 2025. doi:10.48550/arXiv.2506.04528. URL http://arxiv.org/abs/2506.04528. arXiv:2506.04528 [cs].

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, August 2021. ISSN 1476-4687. doi:10.1038/s41586-021-03819-2.

Praneeth Kacham and David Woodruff. Sketching algorithms and lower bounds for ridge regression. In *Proceedings of the 39th International Conference on Machine Learning*, page 10539–10556. PMLR, Jun 2022. URL https://proceedings.mlr.press/v162/kacham22a.html.

Ulugbek S. Kamilov, Hassan Mansour, and Brendt Wohlberg. A Plug-and-Play Priors Approach for Solving Nonlinear Imaging Inverse Problems. *IEEE Signal Processing Letters*, 24(12):1872–1876, December 2017. ISSN 1558-2361. doi:10.1109/LSP.2017.2763583.

Gregor Kasieczka, Tilman Plehn, Anja Butter, Kyle Cranmer, Dipsikha Debnath, Barry M. Dillon, Malcolm Fairbairn, Darius A. Faroughy, Wojtek Fedorko, Christophe Gay, Loukas Gouskos, Jernej Fesel Kamenik, Patrick Komiske, Simon Leiss, Alison Lister, Sebastian

Macaluso, Eric Metodiev, Liam Moore, Benjamin Nachman, Karl Nordström, Jannicke Pearkes, Huilin Qu, Yannik Rath, Marcel Rieger, David Shih, Jennifer Thompson, and Sreedevi Varma. The Machine Learning landscape of top taggers. *SciPost Physics*, 7(1): 014, July 2019. ISSN 2542-4653. doi:10.21468/SciPostPhys.7.1.014.

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, January 2019. ISSN 1064-8275. doi:10.1137/18M1222399. Publisher: Society for Industrial and Applied Mathematics.

Patrick Kidger. *On Neural Differential Equations*. PhD Thesis, University of Oxford, 2021.

Patrick T. Komiske, Eric M. Metodiev, and Jesse Thaler. Energy flow polynomials: A complete linear basis for jet substructure. *Journal of High Energy Physics*, 2018(4):13, Apr 2018. ISSN 1029-8479. doi:10.1007/JHEP04(2018)013. arXiv:1712.07124 [hep-ex, physics:hep-ph].

Risi Kondor, Zhen Lin, and Shubhendu Trivedi. Clebsch–gordan nets: A fully fourier space spherical convolutional neural network. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 10138–10147, Red Hook, NY, USA, 2018. Curran Associates Inc.

Soren D. Konecky, George Y. Panasyuk, Kijoon Lee, Vadim Markel, Arjun G. Yodh, and John C. Schotland. Imaging complex structures with diffuse light. *Optics Express*, 16(7): 5048–5060, March 2008. ISSN 1094-4087. doi:10.1364/OE.16.005048.

David A. Kopriva. A staggered-grid multidomain spectral method for the compressible Navier–Stokes equations. *Journal of Computational Physics*, 143(1):125–158, June 1998. ISSN 0021-9991. doi:10.1006/jcph.1998.5956. URL https://www.sciencedirect.com/science/article/pii/S0021999198959563.

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.

Dávid Péter Kovács, Cas van der Oord, Jiri Kucera, Alice E. A. Allen, Daniel J. Cole, Christoph Ortner, and Gábor Csányi. Linear Atomic Cluster Expansion Force Fields for Organic Molecules: Beyond RMSE. *Journal of Chemical Theory and Computation*, 17 (12):7696–7711, December 2021. ISSN 1549-9618, 1549-9626. doi:10.1021/acs.jctc.1c00647. URL https://pubs.acs.org/doi/10.1021/acs.jctc.1c00647.

Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, page 26548–26560. Curran Associates,

Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/df438e5206f31600e6ae4af72f2725f1-Paper.pdf.

Anish Kumar and Walter Arnold. High resolution in non-destructive testing: A review. *Journal of Applied Physics*, 132(10):100901, September 2022. ISSN 0021-8979. doi:10.1063/5.0095328.

Joseph Kump, Anna Yesypenko, and Per-Gunnar Martinsson. A two-level direct solver for the hierarchical Poincaré–Steklov method. (arXiv:2503.04033), March 2025. doi:10.48550/arXiv.2503.04033. URL http://arxiv.org/abs/2503.04033. arXiv:2503.04033 [math].

Chris Leary and Todd Wang. XLA: TensorFlow, Compiled!, February 2017.

Jason D. Lee, Yuekai Sun, and Michael A. Saunders. Proximal Newton-Type Methods for Minimizing Composite Functions. *SIAM Journal on Optimization*, 24(3):1420–1443, January 2014. ISSN 1052-6234. doi:10.1137/130921428.

Kai Li, Bo Zhang, and Haiwen Zhang. Reconstruction of inhomogeneous media by an iteration algorithm with a learned projector. *Inverse Problems*, 40(7):075008, June 2024. ISSN 0266-5611. doi:10.1088/1361-6420/ad4f0b.

Matthew Li, Laurent Demanet, and Leonardo Zepeda-Núñez. Accurate and Robust Deep Learning Framework for Solving Wave-Based Inverse Problems in the Super-Resolution Regime, June 2021a. URL http://arxiv.org/abs/2106.01143. arXiv:2106.01143 [cs, math, stat].

Matthew Li, Laurent Demanet, and Leonardo Zepeda-Núñez. Wide-Band Butterfly Network: Stable and Efficient Inversion Via Multi-Frequency Neural Networks. *Multiscale Modeling & Simulation*, 20(4):1191–1227, December 2022. ISSN 1540-3459. doi:10.1137/20M1383276. Publisher: Society for Industrial and Applied Mathematics.

Xiaoye S. Li and James W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Math. Softw.*, 29(2):110–140, June 2003. ISSN 0098-3500. doi:10.1145/779359.779361. URL https://dl.acm.org/doi/10.1145/779359.779361.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021b. URL https://openreview.net/forum?id=c8P9NQVtmnO.

Hsiou-Yuan Liu, Dehong Liu, Hassan Mansour, Petros T. Boufounos, Laura Waller, and Ulugbek S. Kamilov. SEAGLE: Sparsity-Driven Image Reconstruction Under Multiple Scattering. *IEEE Transactions on Computational Imaging*, 4(1):73–86, March 2018. ISSN 2333-9403, 2334-0118. doi:10.1109/TCI.2017.2764461.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, March 2021. ISSN 2522-5839. doi:10.1038/s42256-021-00302-5.

Wenqi Lu, Jinming Duan, David Orive-Miguel, Lionel Herve, and Iain B. Styles. Graph- and finite element-based total variation models for the inverse problem in diffuse optical tomography. *Biomedical Optics Express*, 10(6):2684–2707, June 2019. ISSN 2156-7085. doi:10.1364/BOE.10.002684.

José Pablo Lucero Lorca, Natalie Beams, Damien Beecroft, and Adrianna Gillman. An iterative solver for the HPS discretization applied to three dimensional Helmholtz problems. *SIAM Journal on Scientific Computing*, 46(1):A80–A104, February 2024. ISSN 1064-8275. doi:10.1137/21M1463380. URL https://epubs.siam.org/doi/full/10.1137/21M1463380. Publisher: Society for Industrial and Applied Mathematics.

Yanting Ma, Hassan Mansour, Dehong Liu, Petros T. Boufounos, and Ulugbek S. Kamilov. Accelerated Image Reconstruction for Nonlinear Diffractive Imaging. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6473–6477, April 2018. doi:10.1109/ICASSP.2018.8462400.

P. G. Martinsson. A direct solver for variable coefficient elliptic PDEs discretized via a composite spectral collocation method. *Journal of Computational Physics*, 242:460–479, June 2013. ISSN 0021-9991. doi:10.1016/j.jcp.2013.02.019. URL https://www.sciencedirect.com/science/article/pii/S0021999113001320.

Per-Gunnar Martinsson. *Fast Direct Solvers for Elliptic PDEs*. Society for Industrial and Applied Mathematics, Philadelphia, PA, January 2019. ISBN 978-1-61197-603-8 978-1-61197-604-5. doi:10.1137/1.9781611976045. URL https://epubs.siam.org/doi/book/10.1137/1.9781611976045.

Houcine Meftahi. Uniqueness, lipschitz stability, and reconstruction for the inverse optical tomography problem. *SIAM Journal on Mathematical Analysis*, 53(6):6326–6354, January 2021. ISSN 0036-1410. doi:10.1137/20M1386955.

Song Mei, Theodor Misiakiewicz, and Andrea Montanari. Learning with invariances in random features and kernel models. In Mikhail Belkin and Samory Kpotufe, editors, *Proceedings of Thirty Fourth Conference on Learning Theory*, volume 134 of *Proceedings of Machine Learning Research*, pages 3351–3418. PMLR, 15–19 Aug 2021. URL https://proceedings.mlr.press/v134/mei21a.html.

Owen Melia, Eric M. Jonas, and Rebecca Willett. Rotation-Invariant Random Features Provide a Strong Baseline for Machine Learning on 3D Point Clouds. *Transactions on Machine Learning Research*, April 2023. ISSN 2835-8856.

Owen Melia, Daniel Fortunato, Jeremy Hoskins, and Rebecca Willett. Hardware Acceleration for HPS Algorithms in Two and Three Dimensions, March 2025a.

Owen Melia, Olivia Tsang, Vasileios Charisopoulos, Yuehaw Khoo, Jeremy Hoskins, and Rebecca Willett. Multi-frequency progressive refinement for learned inverse scattering. *Journal of Computational Physics*, 527:113809, April 2025b. ISSN 0021-9991. doi:10.1016/j.jcp.2025.113809.

Xiangrui Meng, Michael A. Saunders, and Michael W. Mahoney. LSRN: A Parallel Iterative Solver for Strongly Over- or Underdetermined Systems. *SIAM Journal on Scientific Computing*, 36(2):C95–C118, January 2014. ISSN 1064-8275, 1095-7197. doi:10.1137/120866580. URL http://epubs.siam.org/doi/10.1137/120866580.

Vishal Monga, Yuelong Li, and Yonina C Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Processing Magazine*, 38(2): 18–44, 2021.

Grégoire Montavon, Katja Hansen, Siamac Fazli, Matthias Rupp, Franziska Biegler, Andreas Ziehe, Alexandre Tkatchenko, O. Anatole von Lilienfeld, and Klaus-Robert Müller. Learning invariant representations of molecules for atomization energy prediction. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 440–448, Red Hook, NY, USA, 2012. Curran Associates Inc.

F. Natterer. *The Mathematics of Computerized Tomography*. Society for Industrial and Applied Mathematics, 2001. doi:10.1137/1.9780898719284.

Quoc Tuan Nguyen Diep, Hoang Nhut Huynh, Thanh Ven Huynh, Minh Quan Cao Dinh, Anh Tu Tran, and Trung Nghia Tran. Impact of ISTA and FISTA iterative optimization algorithms on electrical impedance tomography image reconstruction. *Journal of Electrical Bioimpedance*, 16(1):11–22, March 2025. doi:10.2478/joeb-2025-0003.

Anthony Nicholls and Barry Honig. A rapid finite difference algorithm, utilizing successive over-relaxation to solve the Poisson–Boltzmann equation. *Journal of Computational Chemistry*, 12(4):435–445, 1991. ISSN 1096-987X. doi:10.1002/jcc.540120405. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.540120405.

Shinpei Okawa and Yoko Hoshi. A Review of Image Reconstruction Algorithms for Diffuse Optical Tomography. *Applied Sciences*, 13(8):5016, January 2023. ISSN 2076-3417. doi:10.3390/app13085016.

Yong Zheng Ong, Zuowei Shen, and Haizhao Yang. Integral autoencoder network for discretization-invariant learning. *Journal of Machine Learning Research*, 23(286):1–45, 2022. ISSN 1533-7928.

Gregory Ongie, Ajil Jalal, Christopher A. Metzler, Richard G. Baraniuk, Alexandros G. Dimakis, and Rebecca Willett. Deep Learning Techniques for Inverse Problems in Imaging. *IEEE Journal on Selected Areas in Information Theory*, 1(1):39–56, May 2020. ISSN 2641-8770. doi:10.1109/JSAIT.2020.2991563. Conference Name: IEEE Journal on Selected Areas in Information Theory.

Christopher C. Paige and Michael A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software*, 8(1): 43–71, March 1982a. ISSN 0098-3500. doi:10.1145/355984.355989. URL https://dl.acm.org/doi/10.1145/355984.355989.

Christopher C. Paige and Michael A. Saunders. Algorithm 583: Lsqr: Sparse linear equations and least squares problems. *ACM Transactions on Mathematical Software*, 8(2):195–209, Jun 1982b. ISSN 0098-3500. doi:10.1145/355993.356000.

Harald P. Pfeiffer, Lawrence E. Kidder, Mark A. Scheel, and Saul A. Teukolsky. A multidomain spectral method for solving elliptic equations. *Computer Physics Communications*, 152(3):253–273, May 2003. ISSN 0010-4655. doi:10.1016/S0010-4655(02)00847-0. URL https://www.sciencedirect.com/science/article/pii/S0010465502008470.

Adrien Poulenard, Marie-Julie Rakotosaona, Yann Ponty, and Maks Ovsjanikov. Effective Rotation-Invariant Point CNN with Spherical Harmonics Kernels. In *2019 International Conference on 3D Vision (3DV)*, pages 47–56, September 2019. doi:10.1109/3DV.2019.00015. ISSN: 2475-7888.

Omri Puny, Matan Atzmon, Edward J. Smith, Ishan Misra, Aditya Grover, Heli Ben-Hamu, and Yaron Lipman. Frame Averaging for Invariant and Equivariant Network Design. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=zIUyj55nXR.

Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017a.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017b. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/d8bf84be3800d12f74d8b05e9b89836f-Paper.pdf.

Zhuoran Qiao, Matthew Welborn, Animashree Anandkumar, Frederick R. Manby, and III Miller, Thomas F. Orbnet: Deep learning for quantum chemistry using symmetry-adapted atomic-orbital features. *The Journal of Chemical Physics*, 153(12):124111, Sep 2020. ISSN 0021-9606. doi:10.1063/5.0021955.

Ali Rahimi and Benjamin Recht. Random Features for Large-Scale Kernel Machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007. URL https://proceedings.neurips.cc/paper/2007/file/013a006f03dbc5392effeb8f18fda755-Paper.pdf.

Ali Rahimi and Benjamin Recht. Weighted Sums of Random Kitchen Sinks: Replacing minimization with randomization in learning. In D. Koller, D. Schuurmans, Y. Bengio,

and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2008. URL https://proceedings.neurips.cc/paper/2008/file/0efe32849d230d7f53049ddc4a4b0c60-Paper.pdf.

Raghunathan Ramakrishnan, Pavlo O. Dral, Matthias Rupp, and O. Anatole von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1 (1):140022, December 2014. ISSN 2052-4463. doi:10.1038/sdata.2014.22. URL http://www.nature.com/articles/sdata201422.

Lars Ruddigkeit, Ruud van Deursen, Lorenz C. Blum, and Jean-Louis Reymond. Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17. *Journal of Chemical Information and Modeling*, 52(11):2864–2875, November 2012. ISSN 1549-9596, 1549-960X. doi:10.1021/ci300415d. URL https://pubs.acs.org/doi/10.1021/ci300415d.

Matthias Rupp, Alexandre Tkatchenko, Klaus-Robert Müller, and O. Anatole von Lilienfeld. Fast and Accurate Modeling of Molecular Atomization Energies with Machine Learning. *Physical Review Letters*, 108(5):058301, January 2012. ISSN 0031-9007, 1079-7114. doi:10.1103/PhysRevLett.108.058301. URL https://link.aps.org/doi/10.1103/PhysRevLett.108.058301.

Ernest K. Ryu and Wotao Yin. *Large-Scale Convex Optimization: Algorithms & Analyses via Monotone Operators*. Cambridge University Press, 1 edition, November 2022. ISBN 978-1-00-916086-5 978-1-00-916085-8. doi:10.1017/9781009160865.

Yousef Saad and Martin H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986. doi:10.1137/0907058.

Nils Schuergers, Conrad W Mullineaux, and Annegret Wilde. Cyanobacteria in motion. *Current Opinion in Plant Biology*, 37:109–115, June 2017. ISSN 1369-5266. doi:10.1016/j.pbi.2017.03.018. URL https://www.sciencedirect.com/science/article/pii/S1369526617300067.

K. T. Schütt, H. E. Sauceda, P.-J. Kindermans, A. Tkatchenko, and K.-R. Müller. SchNet – A deep learning architecture for molecules and materials. *The Journal of Chemical Physics*, 148(24):241722, June 2018. ISSN 0021-9606, 1089-7690. doi:10.1063/1.5019779. URL http://aip.scitation.org/doi/10.1063/1.5019779.

Hong Ye Tan, Subhadip Mukherjee, Junqi Tang, and Carola-Bibiane Schönlieb. Provably Convergent Plug-and-Play Quasi-Newton Methods. *SIAM Journal on Imaging Sciences*, 17(2):785–819, June 2024. doi:10.1137/23M157185X.

Jinping Tang. Nonconvex and nonsmooth total variation regularization method for diffuse optical tomography based on RTE*. *Inverse Problems*, 37(6):065001, May 2021. ISSN 0266-5611. doi:10.1088/1361-6420/abf5ed.

Jesse Thaler and Ken Van Tilburg. Identifying boosted objects with n-subjettiness. *Journal of High Energy Physics*, 2011(3):15, Mar 2011. ISSN 1029-8479. doi:10.1007/JHEP03(2011)015.

Nathaniel Thomas, Tess Smidt, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor field networks: Rotation- and translation-equivariant neural networks for 3D point clouds, May 2018. URL http://arxiv.org/abs/1802.08219. Number: arXiv:1802.08219 arXiv:1802.08219 [cs].

William J. Thompson. *Angular momentum: an illustrated guide to rotational symmetries for physical systems*. Wiley, New York, 1994. ISBN 978-0-471-55264-2.

Lloyd N. Trefethen. *Spectral Methods in MATLAB*. Software, Environments, and Tools. Society for Industrial and Applied Mathematics, January 2000. ISBN 978-0-89871-465-4. doi:10.1137/1.9780898719598. URL https://epubs-siam-org.proxy.uchicago.edu/doi/book/10.1137/1.9780898719598.

Oliver T. Unke and Markus Meuwly. PhysNet: A Neural Network for Predicting Energies, Forces, Dipole Moments, and Partial Charges. *Journal of Chemical Theory and Computation*, 15(6):3678–3693, June 2019. ISSN 1549-9618, 1549-9626. doi:10.1021/acs.jctc.9b00181. URL https://pubs.acs.org/doi/10.1021/acs.jctc.9b00181.

Singanallur V. Venkatakrishnan, Charles A. Bouman, and Brendt Wohlberg. Plug-and-play priors for model based reconstruction. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 945–948, 2013. doi:10.1109/GlobalSIP.2013.6737048.

Felipe Vico, Leslie Greengard, and Miguel Ferrando. Fast convolution with free-space Green's functions. *Journal of Computational Physics*, 323:191–203, October 2016. ISSN 0021-9991. doi:10.1016/j.jcp.2016.07.028. URL https://www.sciencedirect.com/science/article/pii/S0021999116303230.

Soledad Villar, David W Hogg, Kate Storey-Fisher, Weichi Yao, and Ben Blum-Smith. Scalars are universal: Equivariant machine learning, structured like classical physics. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=ba27-RzNaIv.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi:10.1038/s41592-019-0686-2.

Zheng-Min Wang, George Y. Panasyuk, Vadim A. Markel, and John C. Schotland. Experimental demonstration of an analytic method for image reconstruction in optical diffusion tomography with large data sets. *Optics Letters*, 30(24):3338–3340, December 2005. ISSN 1539-4794. doi:10.1364/OL.30.003338.

A. J. Wathen. Preconditioning. *Acta Numerica*, 24:329–376, May 2015. ISSN 0962-4929, 1474-0508. doi:10.1017/S0962492915000021.

Layne T. Watson and Raphael T. Haftka. Modern homotopy methods in optimization. *Computer Methods in Applied Mechanics and Engineering*, 74(3):289–305, September 1989. ISSN 0045-7825. doi:10.1016/0045-7825(89)90053-4.

Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco Cohen. 3d steerable cnns: Learning rotationally equivariant features in volumetric data. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 10402–10413, Red Hook, NY, USA, 2018. Curran Associates Inc.

Zelin Xiao, Hongxin Lin, Renjie Li, Lishuai Geng, Hongyang Chao, and Shengyong Ding. Endowing Deep 3d Models With Rotation Invariance Based On Principal Component Analysis. In *2020 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, London, United Kingdom, July 2020. IEEE. ISBN 978-1-72811-331-9. doi:10.1109/ICME46284.2020.9102947. URL https://ieeexplore.ieee.org/document/9102947/.

Tianju Xue, Shuheng Liao, Zhengtao Gan, Chanwook Park, Xiaoyu Xie, Wing Kam Liu, and Jian Cao. JAX-FEM: A differentiable GPU-accelerated 3D finite element solver for automatic inverse design and mechanistic data science. *Computer Physics Communications*, 291:108802, October 2023. ISSN 0010-4655. doi:10.1016/j.cpc.2023.108802. URL https://www.sciencedirect.com/science/article/pii/S0010465523001479.

B. Yang and J. S. Hesthaven. Multidomain pseudospectral computation of Maxwell's equations in 3-D general curvilinear coordinates. *Applied Numerical Mathematics*, 33 (1):281–289, May 2000. ISSN 0168-9274. doi:10.1016/S0168-9274(99)00094-X. URL https://www.sciencedirect.com/science/article/pii/S016892749900094X.

Anna Yesypenko. SlabLU: A Two-Level Sparse Direct Solver for Elliptic PDEs in Python, May 2024. URL https://github.com/annayesy/slabLU.

Anna Yesypenko and Per-Gunnar Martinsson. GPU optimizations for the hierarchical Poincaré-Steklov Scheme. In Zdeněk Dostál, Tomáš Kozubek, Axel Klawonn, Ulrich Langer, Luca F. Pavarino, Jakub Šístek, and Olof B. Widlund, editors, *Domain Decomposition Methods in Science and Engineering XXVII*, pages 519–528, Cham, 2024a. Springer Nature Switzerland. ISBN 978-3-031-50769-4. doi:10.1007/978-3-031-50769-4_62.

Anna Yesypenko and Per-Gunnar Martinsson. SlabLU: a two-level sparse direct solver for elliptic PDEs. *Advances in Computational Mathematics*, 50(4):90, August 2024b. ISSN

1572-9044. doi:10.1007/s10444-024-10176-x. URL https://doi.org/10.1007/s10444-024-10176-x.

Borong Zhang, Leonardo Zepeda-Núñez, and Qin Li. Solving the Wide-band Inverse Scattering Problem via Equivariant Neural Networks, October 2023. URL http://arxiv.org/abs/2212.06068. arXiv:2212.06068 [cs, math].

Borong Zhang, Martín Guerra, Qin Li, and Leonardo Zepeda-Núñez. Back-projection diffusion: Solving the wideband inverse scattering problem with diffusion models. (arXiv:2408.02866), August 2024. URL http://arxiv.org/abs/2408.02866. arXiv:2408.02866 [cs, math].

Qingqing Zhao, Yanting Ma, Petros Boufounos, Saleh Nabi, and Hassan Mansour. Deep born operator learning for reflection tomographic imaging. In *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5, 2023. doi:10.1109/ICASSP49357.2023.10095494.

Yongyi Zhao, Ankit Raghuram, Hyun K. Kim, Andreas H. Hielscher, Jacob T. Robinson, and Ashok Veeraraghavan. High Resolution, Deep Imaging Using Confocal Time-of-Flight Diffuse Optical Tomography. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(7):2206–2219, July 2021. ISSN 1939-3539. doi:10.1109/TPAMI.2021.3075366.

Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, Boston, MA, USA, June 2015. IEEE. ISBN 978-1-4673-6964-0. doi:10.1109/CVPR.2015.7298801. URL https://ieeexplore.ieee.org/document/7298801/.

Mo Zhou, Jiequn Han, Manas Rachh, and Carlos Borges. A neural network warm-start approach for the inverse acoustic obstacle scattering problem. *Journal of Computational Physics*, 490:112341, 2023. ISSN 0021-9991. doi:https://doi.org/10.1016/j.jcp.2023.112341.