SOLVING THE WIDE-BAND INVERSE SCATTERING PROBLEM VIA EQUIVARIANT NEURAL NETWORKS

Borong Zhang
Department of Mathematics
University of Wisconsin-Madison
Madison, WI 53706
bzhang388@wisc.edu

Leonardo Zepeda-Nunez
Department of Mathematics
University of Wisconsin-Madison
Madison, WI 53706
Google Research
Mountain View, CA 94043
lzepedanunez@google.com

Qin Li
Department of Mathematics
University of Wisconsin-Madison
Madison, WI 53706
qinli@math.wisc.edu

October 11, 2023

ABSTRACT

This paper introduces a novel deep neural network architecture for solving the inverse scattering problem in frequency domain with wide-band data, by directly approximating the inverse map, thus avoiding the expensive optimization loop of classical methods. The architecture is motivated by the filtered back-projection formula in the full aperture regime and with homogeneous background, and it leverages the underlying equivariance of the problem and compressibility of the integral operator. This drastically reduces the number of training parameters, and therefore the computational and sample complexity of the method. In particular, we obtain an architecture whose number of parameters scales sub-linearly with respect to the dimension of the inputs, while its inference complexity scales super-linearly but with very small constants. We provide several numerical tests that show that the current approach results in better reconstruction than optimization-based techniques such as full-waveform inversion, but at a fraction of the cost while being competitive with state-of-the-art machine learning methods.

1 Introduction

Inverse wave scattering is a classical inverse problem that utilizes the scattered wave from an impinging probing signal (or *data*) to infer the acoustic properties of the object being probed (or the *to-be-be-reconstructed media*). The problem finds wide applications in radar imaging [1], sonar imaging [2], seismic exploration [3], geophysics exploration [4], bio-medical imaging [5] and so on.

At the theoretical level, the well studied single-frequency (or monochromatic) inverse wave scattering problem has been proved to produce a unique reconstruction of the media [6]: if the full incoming-to-scattered wave map is given for a fixed frequency, the data can uniquely determine the underlying object, when the media is assumed to be smooth. Unfortunately, this mathematical statement finds little practical use in the design of inversion algorithms. The inversion problem, when studied in the theoretical setting requires the knowledge of the full map, while the numerical implementation only exploits a data set of finite, albeit very large in size. In addition, it is known that inverse scattering problem is severely ill-posed and the quality of the reconstruction *crucially* depends on the quality of the collected data, particularly, when the data is finite. Thus, in summary, the inverse scattering problem in the numerical setting is often confronted with two issues: its sample complexity, i.e., the amount of the data required, and the lack of stability of the reconstruction.

The sample complexity is usually related to the parametrization of the medium to be reconstructed. While the mathematical PDE statement guarantees the reconstruction of a function, numerically one needs to represent this function using a finite, but often very large, number of parameters, particularly when the media is expected to have fine-grained structures.

This calls for a large amount of data for the reconstruction due to so-called diffraction limit [7]. In fact, to capture such small features, one needs to probe the medium with higher-frequency waves, with an increasingly larger number of directions, while the scattered waves need to be sampled at an increasingly finer rate. Qualitatively, in two dimensions the amount of data needed scales proportionally to the square of the inverse of the characteristic length, the length that represents the smallest feature in the media one seeks to recover.

The lack of stability has been mathematically proved [8] for monochromatic data: a small perturbation in the collected data leads to disastrous inaccuracies in the reconstruction of the media [9]. To overcome such a difficulty, within the classical computational pipelines of optimization-based reconstruction techniques, one typically uses wide-band data, i.e., using different frequencies for the probing waves. Unfortunately this brings two other issues, first, the cost of simulating the wave propagation is prohibitive, particularly at high-frequency, due to combination of the Shannon-Nyquist sampling criterion [10, 11] and a strict Courant–Friedrichs–Lewy (CFL) condition [12], and second, the problem is still highly non-convex, so non-physical local minima are abundant [13]. During the past decades, multiple numerical strategies have been taken to ease these difficulties, including fast PDE solvers [14, 15] that seek to alleviate the computational costs, and new optimization pipelines such as recursive linearization [16] and full wave inversion [17] that seek to avoid some of the spurious local minima in the optimization loop by a hierarchical processing of the data.

In a nutshell, in order to reconstruct fine-grained details one needs higher frequency data; however, as the frequency increases, the size of the data becomes larger thus rendering each step in the optimization procedure increasingly expensive, and the optimization loss becomes increasingly non-convex.

Instead of using an iterative method to reconstruct the media, one alternative is to *directly* approximate the map between the to-be-reconstructed media and the data, which would allows us to circumvent some of the issues mentioned above. Thus driven by the empirical success of machine learning in approximating high-dimensional, highly non-linear maps in a myriad of applications [18, 19, 20], we investigate whether it is possible to *efficiently* approximate this map by leveraging the power of deep learning, when the to-be-reconstructed medium is parametrized by an *unknown* but relatively low-dimensional manifold.

In our setting, the question we seek to answer in this paper is:

Are neural networks capable of efficiently approximate the map between media and data?

The answer is high-likely to be positive as neural operator networks [21, 22] have been developed by a growing number of researchers reaching different level of success. Unfortunately, both difficulties, i.e., lack of stability and sample complexity are also present when approximating the map directly, albeit they are expressed differently. The lack of stability induces a severe lack of convexity in the energy landscape of typical optimization losses, rendering the training of the network challenging, while the number of parameters for the map is often very large and increases rapidly as the frequency increases, which in return requires an increasingly larger amount of training data.

Under the theme of neural approximation of the high-dimensional maps, we seek to alleviate the complexity issues mentioned above for the inverse scattering problem. We study if one can build certain properties of media-data relation into the design of a neural architecture, so to enforce these properties and therefore reduce the computational cost, resulting in an efficient method, particularly when using wide-band data, which has been shown to stabilize the training [23]. Specifically, we look into the equivariant property and butterfly structure of the media-data relation, and we design a neural network that has these features built-in. Such architecture not only respects the underlying physics, interpreted through the PDE (Helmholtz equation), but also helps to reduce the overall computational cost.

The main contribution of this work is to construct a model that is able to reconstruct relatively fine-grained features accurately such that:

- it has asymptotically fewer training parameters, which is achieved by leveraging a detailed description of the linearized operator, the underlying equivariance of the problem, and a compression technique tailored for oscillatory phenomena; and
- empirically, it requires much smaller training datasets to achieve high-validation accuracy,

particularly when compared to existing approaches [24, 25, 23].

There have been several new developments for leveraging ML techniques for inverse problems. In [26] the authors used the recently introduced paradigm of physics informed neural networks (PINN) to solve for inverse problems in optics. Aggarwal et al. introduce a model-based image reconstruction framework [27] for MRI reconstruction. The formulation contains a novel data-consistency step that performs conjugate gradient iterations inside the unrolled algorithm. Gilton et al. proposed in [28] a novel network based on Neumann expansion series coupled with a hand-crafted pre-conditioner

for linear inverse problems, which recast an unrolled algorithm as elements in the Neumann series. In [29] Mao et al. use a deep encoder-decoder network reminiscent of U-nets [30] for image de-noising, using symmetric skip connections. Networks based on the scattering transform has been proposed [31] to take in account translational equivariance in images. In [32] the authors introduced another framework based on frames for inverse problems, which was applied to computer tomography de-noising [33]. These works pioneered the application of deep learning and explored the possibility of integrating physics of inverse scattering with novel machine learning algorithms.

In this paper we specifically focus in two features of the inverse scattering problem: the complementary low-rank property of the oscillatory integrals in the inversion formula, and the exploitation of equivariance. We point out that these features have been studied in literature although only *separately*. On the algebraical level, constructions of structures such as H-matrices [34], Fast Fourier Transforms [35, 36] or butterfly factorizations [37, 24, 38, 39, 23] have been extensively studied for reducing the computational cost. We are interested in mimicking these constructions in the design of the network. In addition, in [25], the authors proposed a network that is rotationally equivariant, which allows them to reduce the number of degrees of freedom, albeit only using monochromatic data.

Our current approach exploits both traits: our design of the neural architecture relies on equivariant formulation of the linearized operator, which we further compress it using a Butterfly-type structure [40], while processing several frequencies simultaneously.

We organize our paper as follows. In Section 2, we present the problem setup and explain the details of the media-data relation. We introduce the equivariant property and the butterfly structure of the media-data relation inherited from the PDE. The embedding of these properties into the design of NN architectures will be discussed in Section 3.1 and 3.2 respectively. The final section contains several numerical experiments showcasing the properties of the proposed method.

2 Problem setup

The inverse scattering problem, at large, is the inverse problem associated with the Helmholtz equation, one of the simplest models for time-harmonic wave propagation. Despite its simplicity, it retains the core difficulties of models with more complex physics such as seismic or electromagnetic waves, thus making it an ideal test bed for new algorithms. The Helmholtz equation is the Fourier-in-time transform of the constant-density acoustic wave equation and it reads:

$$\Delta u(\boldsymbol{x}) + \omega^2 n(\boldsymbol{x}) u(\boldsymbol{x}) = 0\,, \quad \boldsymbol{x} \in \Omega \subset \mathbb{R}^2 \tag{1}$$

where u is the total wave field, ω is the probing wave frequency, and n is the refractive index that encodes the media's wave speed. The domain of interest is denoted by $\Omega \subset \mathbb{R}^2$. We assume that the background media is homogeneous and equal to one, i.e., $n(\boldsymbol{x}) = 1$ for $\boldsymbol{x} \notin \Omega$. For simplicity we denote the perturbation $\eta(\boldsymbol{x}) = n(\boldsymbol{x}) - 1$, thus we have $\text{supp}(\eta(\boldsymbol{x})) \subset \Omega$.

For a given n (or equivalently η), the forward problem amounts to solving for the scattered wave field under the Sommerfeld radiation condition when the media is impinged by a probing wave. Due to the homogeneous background assumption we consider probing waves as plane waves:

$$u^{\text{in}} = e^{i\omega \boldsymbol{s}\cdot\boldsymbol{x}}, \tag{2}$$

where $\boldsymbol{s} \in \mathbb{S}^1$ is a unitary vector denoting the direction of the incoming plane wave. When injected into (1), it triggers the scattered wave field u^{sc}. Define

$$u^{\text{sc}} = u - u^{\text{in}}\,,$$

and utilizing the fact that u^{in} solves the Helmholtz equation with trivial media $n(\boldsymbol{x}) = 1$, it is straightforward to show that $u^{\text{sc}}(\boldsymbol{x}; \boldsymbol{s})$ solves:

$$\begin{cases} \Delta u^{\text{sc}}(\boldsymbol{x}) + \omega^2(1 + \eta(\boldsymbol{x}))u^{\text{sc}}(\boldsymbol{x}) = -\omega^2 \eta(\boldsymbol{x}) u^{\text{in}}, \\ \frac{\partial u^{\text{sc}}}{\partial |\boldsymbol{x}|} - i\omega u^{\text{sc}} = \mathcal{O}(|\boldsymbol{x}|^{-3/2}) \text{ uniformly in } \frac{\boldsymbol{x}}{|\boldsymbol{x}|} \in S^1 \text{ as } |\boldsymbol{x}| \to \infty. \end{cases} \tag{3}$$

The second row in the equation is termed the Sommerfeld radiation condition. It is imposed at the infinity to ensure the well-possedness of the equation (3), which further guarantees the uniqueness of the solution to (1).

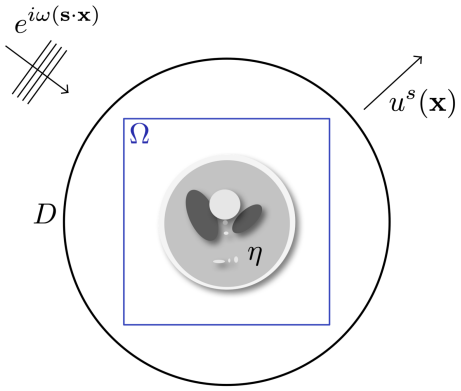

Figure 1: The setup for the inverse scattering problem. In the illustration, the media η, in the domain of interest Ω, is impinged by the probing wave with frequency ω from the direction $\mathbf{s}$. The scattered field $u^{\mathbf{s}}(\mathbf{x})$ is sampled on the disk D.

The measurements are taken at the circle R away from the origin, i.e., we measure the value of $u^{\text{sc}}(R\boldsymbol{r})$ with $R >$ radius(Ω) and $\boldsymbol{r} \in \mathbb{S}^1$. We define the continuous far-field pattern (the data) as $\Lambda^\omega(\boldsymbol{s}, \boldsymbol{r})$, namely:

$$\Lambda^\omega(\boldsymbol{s}, \boldsymbol{r}) = u^{\text{sc}}(R\boldsymbol{r}; \boldsymbol{s})\,. \tag{4}$$

Since this data is uniquely determined by the configuration of $\eta(\boldsymbol{x})$, we define the map from media η to the data Λ^ω as

$$\Lambda^\omega = \mathcal{F}^\omega[\eta]\,.$$

Problem Formulation

While the forward problem is to compute Λ^ω for the given η (or equivalently to apply $\mathcal{F}$), the inverse problem is to infer η from the measured data Λ^ω (or equivalently, to invert $\mathcal{F}$ for η), which we write formally as

$$\eta^* = \mathcal{F}^{-1}(\{\Lambda^\omega\}_{\omega\in\bar{\Omega}}), \tag{5}$$

where $\bar{\Omega}$ is a discrete set of frequencies chosen before hand. We note that even though the PDE itself is linear, $\mathcal{F}^\omega$ is a nonlinear map.

One classical way to numerically execute the inversion is to recast this problem as a PDE-constrained optimization [41]. It seeks the configuration of η so that the synthetic data generated by η (by solving the PDE (3)) minimizes the mismatch with respect to the data Λ^ω generated by the probing wave of frequency ω. Such procedure is formulated as

$$\eta^* = \mathcal{F}^{-1}(\{\Lambda^\omega\}_{\omega\in\bar{\Omega}}) := \text{argmin}_\nu \sum_{\omega\in\bar{\Omega}} \|\mathcal{F}^\omega[\nu] - \Lambda^\omega\|^2. \tag{6}$$

The problem is usually solved using highly tailored gradient-based optimization techniques, where the gradient is computed using adjoint-state methods [42].

2.1 Filtered Back-Projection

Instead of using optimization to solve (6) we seek to approximate the map directly using a neural network. As a motivation for our architecture, we briefly introduce a classical strategy that hinges on the linearization the problem using the Born-approximation [6]. In that regime we compute the impulse response for a perturbation of the input $\eta = \eta_0 + \delta\eta$, with respect to a reference η_0. The operator is then accordingly linearized to:

$$\mathcal{F}^\omega[\eta] = \mathcal{F}^\omega[\eta_0 + \delta\eta] \approx \mathcal{F}^\omega[\eta_0] + F^\omega\delta\eta,$$

where F^ω is a linear operator that maps media perturbation to data perturbation. In our case, we assume that $\eta_0 = 0$, i.e., the background media homogeneous, so η itself is the perturbation, and F can be computed explicitly using the free-space Green's function of the two-dimensional Helmholtz equation. A classical computation omitting the higher order terms in the far-field asymptotics for the Green's function yields:

$$\Lambda^\omega = \mathcal{F}^\omega[\eta] \approx F^\omega\eta\,, \quad \text{with} \quad (F^\omega\eta)\,(\boldsymbol{s}, \boldsymbol{r}) = C_{\text{nor}} \int_\Omega e^{-i\omega(\boldsymbol{r}-\boldsymbol{s})\cdot\boldsymbol{y}}\,\eta(\boldsymbol{y})\,d\boldsymbol{y}\,, \tag{7}$$

where $C_{\text{nor}} = \frac{e^{i\pi/4}}{\sqrt{8\pi\omega}}\omega^2\frac{e^{i\omega R}}{\sqrt{R}}$ is a normalization constant, which we omit from future discussions for the sake of brevity. We stress that according to the formula (7), the data can be viewed as a first-type Fredholm integration over η, the perturbation in media. In this perturbed setting, we hope to find η that minimize the error between Λ^ω and $F^\omega\eta$:

$$\min_{\eta} \|\Lambda^\omega - F^\omega\eta\|^2, \qquad \text{with} \qquad \|\Lambda^\omega - F^\omega\eta\|^2 = \int_{\mathbb{S}^1\times\mathbb{S}^1} |\Lambda^\omega(\boldsymbol{s},\boldsymbol{r}) - (F^\omega\eta)(\boldsymbol{s},\boldsymbol{r})|^2 d\boldsymbol{r}d\boldsymbol{s}\,. \tag{8}$$

Given that F^ω is a linear operator on η, the objective function has a quadratic dependence on the unknown variable η, and thus, formally, the solution is explicitly given by

$$\eta = (F^\omega)^\dagger \Lambda^\omega \quad \text{with} \quad (F^\omega)^\dagger = ((F^\omega)^* F^\omega)^{-1}(F^\omega)^*\,, \tag{9}$$

where $(F^\omega)^*$ is the adjoint operator of F^ω that maps data back to the media. This operator is typically called the back-scattering operator.

It can be proved that the operator $(F^\omega)^* F^\omega$ is compact thus its numerical inverse will be ill-conditioned. A classical technique to circumvent this issues is to apply the Tikhonov regularization, in which one adds $\epsilon\|\eta\|_2^2$ in the objective function, whose solution is given by

$$\eta^* = \min_{\eta} \|\Lambda^\omega - F^\omega\eta\|^2 + \epsilon\|\eta\|^2 \quad \Rightarrow \quad \eta^* = ((F^\omega)^* F^\omega + \epsilon I)^{-1}(F^\omega)^*\Lambda^\omega\,. \tag{10}$$

Due to this extra regularization term, this strategy is termed the filtered back-projection [43].

We should stress that one derives this formula (10) only in the linearized setting under the assumption that $\eta \sim 0$. It is not valid when η is significant, and the formula only serves as the guidance for the actual inversion. In this paper, we are interested in lifting this formula up to the nonlinear setting, and use it as a base to obtain numerical representation of the nonlinear map between data and the media. Mathematically, this is translated to replacing the linear operators (10) by nonlinear maps that will be represented by neural networks. In particular, we intend to label Λ^ω and η^* as the input and output of the neural network, and simulate their relation using a specially designed NN architecture.

Despite (10) not being directly applicable, the formula nevertheless provides insightful guidelines in designing neural architectures. It suggests that the relation between the data and the media is composed of two operations : the inner-layer operator is the adjoint operator $(F^\omega)^*$ (or the back-scattering operator), and the outer-layer operator, $(F^\omega)^* F^\omega + \epsilon I$ (or filtering operator). When one designs neural networks to represent the inverse operator (5), both operators need to be integrated. Each enjoy some unique features, which should be respected and therefore encoded in the neural architecture. In particular, the filtering operator is a pseudo-differential operator of convolution type, therefore *translational equivariant*. As a consequence, the neural ansatz is also of convolution-type and should respect this translational equivariance property. Indeed, we choose to represent it using a few two-dimensional convolutional layers as its discretization, as was studied in [23, 25].

The presentation of the back-scattering operator $(F^\omega)^*$ in the nonlinear setting is more involved. One can prove that this operator satisfies the *rotational equivariance*. This property should be respected by its neural representation. Furthermore, this operator satisfies the *complementary low-rank property*, thus it is expected to be well approximated by the butterfly factorization. Accordingly, the neural representation could incorporate the butterfly structure as well. In Section 2.4 and 2.5 respectively, we will study in depth these two properties, and investigate how to encode these features in the NN architecture to achieve both inversion accuracy and efficiency through enforcing this particular type of compressibility.

2.2 Discretization

We translate the discussion from the previous sections to the discrete setting. To streamline the notation, quantities in calligraphic fonts, such as $\mathcal{F}^\omega$, are used to denote nonlinear maps, while the ones in regular fonts, such as F^ω and Λ^ω, are used to denote the linearized version. The quantities written in serif font, such as F^ω and $\mathsf{\Lambda}^\omega$, are used to present the discretized version of the associated linear operators.

Our method relies on particular choices of the discretization, which we described in what follows. Throughout the paper, we parametrize the incoming direction and the sampling point, $\boldsymbol{s}, \boldsymbol{r} \in \mathbb{S}^1$, by their associated angles

$$\boldsymbol{s} = (\cos(s), \sin(s)) \quad \text{and} \quad \boldsymbol{r} = (\cos(r), \sin(r))\,,$$

and correspondingly, we denote the continuous normalized far-field pattern as

$$\Lambda^\omega_{\text{nor}}(r,s) := u^{\text{sc}}(R\boldsymbol{r};\boldsymbol{s})/C_{\text{nor}}\,, \tag{11}$$

and the normalized operator F^ω by F^ω_{nor}. Upon the normalization, the equation 7 becomes

$$\Lambda^\omega_{\text{nor}} = F^\omega_{\text{nor}}\eta\,, \quad \text{with} \quad \Lambda^\omega_{\text{nor}}(r,s) = \int e^{-i\omega(\boldsymbol{r}-\boldsymbol{s})\cdot\boldsymbol{y}}\eta(\boldsymbol{y})\,d\boldsymbol{y}\,. \tag{12}$$

which leads to the filtered back-projection

$$\eta^* = ((F^\omega_{\text{nor}})^* F^\omega_{\text{nor}} + \epsilon I)^{-1}(F^\omega_{\text{nor}})^*\Lambda^\omega_{\text{nor}}\,.$$

For the conciseness of the notation, we drop the subscript $\cdot_{\text{nor}}$. Numerically, the directions of sources and detectors are represented by the same uniform grid in $\mathbb{S}^1$ with n_{sc} grid points given by

$$s_j, r_j = \frac{2\pi j}{n_{\text{sc}}},\ j = 0,\dots,n_{\text{sc}}-1\,.$$

Using this setting, the discrete data $\mathsf{\Lambda}^\omega$ takes its values on the tensor product of both grids with complex values, which are decomposed in their real and imaginary parts

$$\mathsf{\Lambda}^\omega = \mathsf{\Lambda}^\omega_{\mathsf{R}} + \mathsf{i}\mathsf{\Lambda}^\omega_{\mathsf{I}} \in \mathbb{C}^{\mathsf{n}_{\text{sc}}\times\mathsf{n}_{\text{sc}}}\,. \tag{13}$$

For the discretization of Ω, we use two discretizations, each of them anchored on specific properties of the operators, which will be justified in Section 2.4. On the one hand, as it will be shown in the sequel, the back-scattering operator $(F^\omega)^*$ enjoys the rotational equivariance property (Section 2.4), and it can naturally be represented on the polar coordinates. On the other hand, the filtering operator $((F^\omega)^*F^\omega + \epsilon I)^{-1}$ is translational equivariant and it should be represented on a Cartesian grid.

Therefore, we set the physical domain to be $\Omega = [-0.5, 0.5]^2$. We use a Cartesian mesh of $n_\eta \times n_\eta$ grids. We also discretize it using polar coordinates on a slightly larger domain. In particular, we note $\Omega \subset B_{1/2}(0)$, and define

$$(\theta_j, \rho_i) = \left(\frac{2\pi j}{n_\theta}, \frac{i}{2n_\rho}\right), \quad \text{with} \quad j = 0,\dots,n_\theta - 1\,, i = 0,\dots,n_\rho - 1\,. \tag{14}$$

Thus, $\eta(\boldsymbol{x})$ is represented as a tensor: $\eta \in \mathbb{R}^{n_\eta\times n_\eta}$ with its values being $\eta(\boldsymbol{x})$ evaluated on the Cartesian mesh. We also assume $n_\theta = n_{\text{sc}}$ so that s_j, r_j and θ_j are sampled on the same mesh.

As a consequence, in the algorithmic pipeline we need to express the intermediate representations in both polar and Cartesian meshes on the physical domain, which in return requires a change of coordinates of the data between the application of the two operators. We denote the discrete intermediate field obtained upon the application of the discretized back-scattering operator $(\mathsf{F}^\omega)^*$ by α^ω:

$$\alpha^\omega := (\mathsf{F}^\omega)^*\mathsf{\Lambda}^\omega\,, \tag{15}$$

It lives in the range of $(\mathsf{F}^\omega)^*$ and is represented in polar coordinates. Thus we sample α^ω on a grid using the polar coordinates, meaning α^ω is presented as a vector of size $n_\theta \cdot n_\rho$. Then we use quadratic interpolation, to interpolate it to a Cartesian mesh before being fed to the filtering operator

$$\eta = ((\mathsf{F}^\omega)^*\mathsf{F}^\omega + \epsilon\mathsf{I})^{-1}\alpha^\omega\,. \tag{16}$$

2.3 Computation of $(F^\omega)^*$

We claim that this back-scattering operator enjoys both compressibility through the butterfly structure and the rotational equivariance property. To do so, we compute an explicit formulation for this operator in what follows. Noting that using the standard inner product, we can flip $(F^\omega)^*$ to its dual space:

$$\langle\eta, \alpha^\omega\rangle_\Omega = \langle\eta, (F^\omega)^*\Lambda^\omega\rangle_{\mathbb{S}^1\times\mathbb{S}^1}\,.$$

With straightforward calculation, recalling (7), the right hand side becomes:

$$\begin{aligned}\langle\eta, \alpha^\omega\rangle_\Omega = \langle\eta, (F^\omega)^*\Lambda^\omega\rangle_\Omega &= \int \overline{\Lambda^\omega(r,s)}\int e^{-i\omega(\boldsymbol{r}-\boldsymbol{s})\cdot\boldsymbol{y}}\eta(\boldsymbol{y})\,d\boldsymbol{y}\,ds\,dr\,,\\ &= \int \eta(\boldsymbol{y})\overline{\int e^{i\omega(\boldsymbol{r}-\boldsymbol{s})\cdot\boldsymbol{y}}\Lambda^\omega(r,s)\,ds\,dr\,d\boldsymbol{y}}\,,\end{aligned} \tag{17}$$

suggesting

$$(F^\omega)^*\Lambda^\omega(\boldsymbol{y}) = \int e^{i\omega(\boldsymbol{r}-\boldsymbol{s})\cdot\boldsymbol{y}}\Lambda^\omega(r,s)\,ds\,dr\,. \tag{18}$$

To view it in polar coordinates, we denote that $\boldsymbol{y} = (\rho\cos\theta, \rho\sin\theta)$ and have

$$i\omega(\boldsymbol{r}-\boldsymbol{s})\cdot\boldsymbol{y} = i\omega\rho(\cos(r-\theta)-\cos(s-\theta))\,.$$

Injecting it back to the formula above, the calculation becomes[1]

$$\begin{aligned}\alpha^\omega(\theta,\rho) = ((F^\omega)^*\Lambda^\omega)(\theta,\rho) &= \iint_{[0,2\pi]^2} e^{i\omega\rho\cos(r-\theta)}e^{-i\omega\rho\cos(s-\theta)}\Lambda^\omega(r,s)\,ds\,dr\,,\\ &= \iint_{[0,2\pi]^2} e^{i\omega\rho\cos(r)}e^{-i\omega\rho\cos(s)}\Lambda^\omega(r+\theta,s+\theta)\,ds\,dr\,,\\ &= \int_{[0,2\pi]} e^{i\omega\rho\cos(r)}\left(\int_{[0,2\pi]} e^{-i\omega\rho\cos(s)}\Lambda^\omega(r+\theta,s+\theta)\,ds\right)dr\,.\end{aligned} \tag{19}$$

The formula suggests that the adjoint operator $(F^\omega)^*$ can be decomposed in two-integral operators, both of which involves the integration kernel

$$K^\omega(\rho,t) := e^{-i\omega\rho\cos(t)}\,. \tag{20}$$

2.4 Equivariance

Both the back-scattering operator and the filtering operator in (10) enjoy certain equivariances and the design of NN should respect these features. We summarize the two equivalance that will be incorporated in our studies.

Rotational equivariance One key feature that $(F^\omega)^*$ enjoys is the rotational equivariance: It means that by rotating the data by a certain angle, one is expected to obtain an output that is rotated by the same angle. Recalling the definition of the data (15), and defining the following operator:

$$\mathcal{R}_{\tilde\theta}[\Lambda^\omega](r,s) = \Lambda^\omega(r-\tilde\theta, s-\tilde\theta)\,, \tag{21}$$

which rotates by $\tilde\theta$ the data. Then we expect

$$(F^\omega)^*\mathcal{R}_{\tilde\theta}[\Lambda^\omega](\theta,\rho) = \mathcal{R}_{\tilde\theta}[(F^\omega)^*\Lambda^\omega](\theta,\rho) = [(F^\omega)^*\Lambda^\omega](\theta-\tilde\theta,\rho)\,, \tag{22}$$

in which the operator only acts on the angular inputs. This property can be easily checked by examining the explicit expression (19). Considering that neither integral kernel has θ dependence and by trivially rearranging terms:

$$\begin{aligned}(F^\omega)^*\mathcal{R}_{\tilde\theta}[\Lambda^\omega](\theta,\rho) &= \int_{[0,2\pi]} e^{i\omega\rho\cos(r)}\left(\int_{[0,2\pi]} e^{-i\omega\rho\cos(s)}\Lambda^\omega((r-\tilde\theta)+\theta,(s-\tilde\theta)+\theta)\,ds\right)dr,\\ &= \int_{[0,2\pi]} e^{i\omega\rho\cos(r)}\left(\int_{[0,2\pi]} e^{-i\omega\rho\cos(s)}\Lambda^\omega(r+(\theta-\tilde\theta),s+(\theta-\tilde\theta))\,ds\right)dr,\\ &= (F^\omega)^*[\Lambda^\omega](\theta-\tilde\theta,\rho) = \mathcal{R}_{\tilde\theta}[(F^\omega)^*\Lambda^\omega](\theta,\rho)\,,\end{aligned}$$

proving (22).

Translational equivariance Using the explicit formulas of F^ω in (7) and $(F^\omega)^*$ in (18), we can also verify the translational equivariance of $((F^\omega)^*F^\omega + \epsilon I)^{-1}$. Using the Fubini's theorem, we have

$$\begin{aligned}(F^\omega)^*F^\omega(\eta)(\boldsymbol{y}) &= \int_{[0,2\pi]^2} e^{i\omega(\boldsymbol{r}-\boldsymbol{s})\cdot\boldsymbol{y}}\left(\int_\Omega e^{-i\omega(\boldsymbol{r}-\boldsymbol{s})\cdot\boldsymbol{x}}\,\eta(\boldsymbol{x})\,d\boldsymbol{x}\right)\,ds\,dr\,,\\ &= \int_{[0,2\pi]^2}\int_{\mathbb{R}^2} e^{i\omega(\boldsymbol{r}-\boldsymbol{s})\cdot\boldsymbol{y}}e^{-i\omega(\boldsymbol{r}-\boldsymbol{s})\cdot\boldsymbol{x}}\,\eta(\boldsymbol{x})\,d\boldsymbol{x}\,\,ds\,dr\,,\\ &= \int_{\mathbb{R}^2}\left(\int_{[0,2\pi]^2} e^{i\omega(\boldsymbol{r}-\boldsymbol{s})\cdot(\boldsymbol{y}-\boldsymbol{x})}\,ds\,dr\right)\,\eta(\boldsymbol{x})\,d\boldsymbol{x}\,,\\ &= \int p(\boldsymbol{y}-\boldsymbol{x})\eta(\boldsymbol{x})d\boldsymbol{x} = p*\eta(\boldsymbol{y}).\end{aligned} \tag{23}$$

where the convolution kernel is

$$p(\boldsymbol{x}) := \int_{[0,2\pi]^2} e^{i\omega(\boldsymbol{r}-\boldsymbol{s})\cdot\boldsymbol{x}}\,ds\,dr\,, \tag{24}$$

and the function η extended by zero on Ω^c. This convolution feature justifies that $(F^\omega)^*F^\omega$ is translational equivariant. The same property also holds when we add it with an identity, and take its inversion.

[1] Λ^ω is extended periodically outside the domain $(r,s)\in[0,2\pi]^2$.

2.5 Compression of the kernel K^ω

The back-scattering operator (19) presents two levels of integration against the oscillatory integral kernel $K^\omega(\rho,t) = e^{-i\omega\rho\cos(t)}$ (or its complex conjugate). A brute-force integration of such an oscillatory integrand typically calls for very fine discretization and, therefore, high cost. However, following [23], one can argue that its associated discretized form K^ω can be compressed through the butterfly factorization [44] by leveraging the complimentary low-rank property [44]. In this section we provide an overview of (and the intuition behind) the butterfly factorization algorithm.

For one-dimensional problems the complementary low-rank property can be stated as follows: any block of the matrix with constant area, i.e., the multiplication of its heights by its width, has its numerical rank upper bounded by a constant. For example, in the family of partitions depicted in Figure 2 each block has the same area, and therefore in order to satisfy the complimentary low-rank condition, each block should have the same numerical rank.

We claim that the integrating kernel $K^\omega(\rho,t) = e^{-i\omega\rho\cos(t)}$ satisfies complementary low-rank property. Indeed, the claim can be broadened to any function in the form of $e^{i\omega\phi(\rho,t)}$. To see it, we perform Taylor expansion in the neighborhood of (ρ_0,t_0). Let $|\rho-\rho_0| < d_\rho$ and $|t-t_0| < d_t$, we can approximate $\phi(\rho,t)$ up to the second order expansion as

$$\begin{aligned}\phi(\rho,t) = \phi(\rho_0,t_0) + \partial_\rho\phi(\rho_0,t_0)\cdot(\rho-\rho_0) + \partial_t\phi(\rho_0,t_0)\cdot(t-t_0) \\ + (\rho-\rho_0)^T\cdot\partial_\rho^2\phi(\rho_0,t_0)\cdot(\rho-\rho_0) + (t-t_0)^T\cdot\partial_t^2\phi(\rho_0,t_0)\cdot(t-t_0) + \mathcal{O}(d_\rho d_t)\,.\end{aligned} \tag{25}$$

Since the first five terms are separable and the reminder term is bounded by $\mathcal{O}(d_\rho d_t)$, we have

$$e^{i\omega\phi(\rho,t)} = e^{i\omega\psi(\rho)}e^{i\omega\xi(t)}e^{i\omega\mathcal{O}(d_\rho d_t)} = e^{i\omega\psi(\rho)}e^{i\omega\xi(t)}(1+\mathcal{O}(\omega d_\rho d_t))\,, \tag{26}$$

where $\psi(\rho) = \phi(\rho_0,t_0) + \partial_\rho\phi(\rho_0,t_0)\cdot(\rho-\rho_0) + (\rho-\rho_0)^T\cdot\partial_\rho^2\phi(\rho_0,t_0)\cdot(\rho-\rho_0)$ and $\xi(t) = \partial_t\phi(\rho_0,t_0)\cdot(t-t_0) + (t-t_0)^T\cdot\partial_t^2\phi(\rho_0,t_0)\cdot(t-t_0)$. Therefore, when $d_\rho d_t < \omega^{-1}$, $e^{i\omega\phi(\rho,t)}$ can be locally approximated by separable functions. Further details can be found in [23].

We note that the error term from a separable form is a product of two terms, the size of interval in ρ and the size of interval in t. If the product of these two distances is small, the integral kernel can be approximated by a separable function, hence becoming low-rank. This observation is similar to the requirement of the complementary low-rank property where the size of column indexing and that of row indexing in a given block of the matrix need to complement each other, so the total number of entries (or its ares) in this block is controlled.

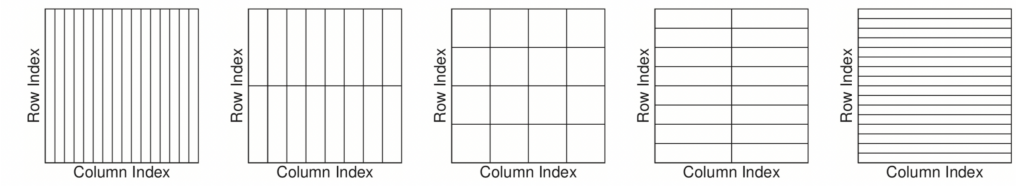

Figure 2: Sketch of a family of partitions of a matrix exhibiting the complementary low-rank property. Each sub-matrix induced by the different partition has the same numerical rank.

A matrix satisfying the complementary low-rank property can be expanded on both sides using a factorization in highly-structured sparse matrices, which can be written using smaller-sized matrix-matrix products with operations on rows/columns respectively, as illustrated in the Figure 2. This multiplication strategy is often termed *butterflying* a matrix. It allows a lower complexity matrix-vector product, reducing the cost of from N^2 to $\mathcal{O}(N\log N)$ [44]. In our case, it is K, a matrix of size $N\times N$ that is complementary low-rank and thus is butterfliable [44]. To be more specific, the butterfly algorithm should find an expansion of the matrix by a product of $L+3$ sparse matrices, each of which having only $\mathcal{O}(N)$ entries:

$$\mathsf{K} \approx \mathsf{U}^L\mathsf{G}^{L-1}\cdots\mathsf{G}^{L/2}\mathsf{M}^{L/2}(\mathsf{H}^{L/2})^*\cdots(\mathsf{H}^{L-1})^*(\mathsf{V}^L)^*\,. \tag{27}$$

This formula approximately expand K into the multiplication of a series of matrix products. Denoting L the number of "levels" in the factorization, U^L and V^L will be block diagonal matrices, and $\mathsf{M}^{L/2}$ is a weighted permutation matrix, which is usually called a switch matrix. The structures of the factors in the butterfly factorization provide acute intuition

into their interpretations. For example, when a vector is right multiplied by the matrix, U^L extracts a local representation of the vector, and then each G^l compresses two adjacent local representations. Upon the application of the switch matrix $\mathsf{M}^{L/2}$ that redistributes the representations from the previous step by permuting the vector, each H^l decompress the representation by splitting it into two, which increases the resolution of the representation. Finally, V^L converts the local representations to sampling points.

Algorithmically, this decomposition is achieved in two stages. In the first, one would perform a singular value decomposition (SVD) for submatrices at level $L/2$. This step is composed of SVDs for N submatrices, with each submatrix of size $m \times m$, where $m = \sqrt{N}$. For the moment, we unify the rank for each SVD decomposition to be r. Denoting $\mathsf{U}_{i,j}^{L/2} \in \mathbb{R}^{m\times r}$, $\mathsf{S}_{i,j}^{L/2} \in \mathbb{R}^{r\times r}$, and $\mathsf{V}_{i,j}^{L/2} \in \mathbb{R}^{m\times r}$ the left singular vectors, singular values and right singular vectors, we stack them up as

$$
\begin{aligned}
\mathsf{K} &\approx \mathsf{U}^{L/2}\mathsf{M}^{L/2}(\mathsf{V}^{L/2})^* \\
&= \begin{pmatrix}
\mathsf{U}_{0,0}^{L/2}\mathsf{S}_{0,0}^{L/2}(\mathsf{V}_{0,0}^{L/2})^* & \mathsf{U}_{0,1}^{L/2}\mathsf{S}_{0,1}^{L/2}(\mathsf{V}_{1,0}^{L/2})^* & \cdots & \mathsf{U}_{0,m-1}^{L/2}\mathsf{S}_{0,m-1}^{L/2}(\mathsf{V}_{m-1,0}^{L/2})^* \\
\mathsf{U}_{1,0}^{L/2}\mathsf{S}_{1,0}^{L/2}(\mathsf{V}_{0,1}^{L/2})^* & \mathsf{U}_{1,1}^{L/2}\mathsf{S}_{1,1}^{L/2}(\mathsf{V}_{1,1}^{L/2})^* & & \mathsf{U}_{1,m-1}^{L/2}\mathsf{S}_{1,m-1}^{L/2}(\mathsf{V}_{m-1,1}^{L/2})^* \\
\vdots & & \ddots & \\
\mathsf{U}_{m-1,0}^{L/2}\mathsf{S}_{m-1,0}^{L/2}(\mathsf{V}_{0,m-1}^{L/2})^* & \mathsf{U}_{m-1,1}^{L/2}\mathsf{S}_{m-1,1}^{L/2}(\mathsf{V}_{1,m-1}^{L/2})^* & \cdots & \mathsf{U}_{m-1,m-1}^{L/2}\mathsf{S}_{m-1,m-1}^{L/2}(\mathsf{V}_{m-1,m-1}^{L/2})^*
\end{pmatrix}.
\end{aligned} \tag{28}
$$

The approximation sign takes into account that SVD cuts off small singular values, and

$$
\mathsf{U}^{L/2} = \begin{pmatrix} \mathsf{U}_0^{L/2} & & & \\ & \mathsf{U}_1^{L/2} & & \\ & & \ddots & \\ & & & \mathsf{U}_{m-1}^{L/2} \end{pmatrix} \quad \text{and} \quad (\mathsf{V}^{L/2})^* = \begin{pmatrix} (\mathsf{V}_0^{L/2})^* & & & \\ & (\mathsf{V}_1^{L/2})^* & & \\ & & \ddots & \\ & & & (\mathsf{V}_{m-1}^{L/2})^* \end{pmatrix}. \tag{29}
$$

with

$$
\mathsf{U}_i^{L/2} = \begin{pmatrix} \mathsf{U}_{i,0}^{L/2} & \mathsf{U}_{i,1}^{L/2} & \cdots & \mathsf{U}_{i,m-1}^{L/2} \end{pmatrix} \quad \text{and} \quad \mathsf{V}_i^{L/2} = \begin{pmatrix} \mathsf{V}_{i,0}^{L/2} & \mathsf{V}_{i,1}^{L/2} & \cdots & \mathsf{V}_{i,m-1}^{L/2} \end{pmatrix}. \tag{30}
$$

Note that the diagonal blocks of U, denoted as U_i stores left (column) singular vectors for the i-th row block, and the total size of $\mathsf{U}_i^{L/2}$ is $m \times mr$. Similarly, diagonal blocks of V stores right (column) singular vectors of size m. In the second stage, one expands these vectors in U and V recursively, by partitioning them in half, while expanding them in the opposite direction to connect parallel blocks. In particular, for each diagonal block U_i, we split it in top and bottom halves

$$
\mathsf{U}_i^{L/2} = \begin{pmatrix} \mathsf{U}_i^{L/2,t} \\ \mathsf{U}_i^{L/2,b} \end{pmatrix} = \begin{pmatrix} \mathsf{U}_{i,0}^{L/2,t} & \mathsf{U}_{i,1}^{L/2,t} & \cdots & \mathsf{U}_{i,m-1}^{L/2,t} \\ \mathsf{U}_{i,0}^{L/2,b} & \mathsf{U}_{i,1}^{L/2,b} & \cdots & \mathsf{U}_{i,m-1}^{L/2,b} \end{pmatrix}.
$$

Noting that $\left(\mathsf{U}_{i,2j}^{L/2,t}, \mathsf{U}_{i,2j+1}^{L/2,t}\right)$ form the column space of $\mathsf{K}_{2i,j}^{L/2+1}$, the $(2i,j)$-block of K decomposed at $(L/2+1)$-th level for row, and $(L/2-1)$-th level for column. Similarly $\left(\mathsf{U}_{i,2j}^{L/2,b}, \mathsf{U}_{i,2j+1}^{L/2,b}\right)$ form the column space of $\mathsf{K}_{2i+1,j}^{L/2+1}$, there are translation matrix $\mathsf{G}_{2i,j}^{L/2}$ and $\mathsf{G}_{2i+1,j}^{L/2}$ so that:

$$
\left(\mathsf{U}_{i,2j}^{L/2,t}, \mathsf{U}_{i,2j+1}^{L/2,t}\right) = \mathsf{U}_{2i,j}^{L/2+1}\mathsf{G}_{2i,j}^{L/2}, \quad \left(\mathsf{U}_{i,2j}^{L/2,b}, \mathsf{U}_{i,2j+1}^{L/2,b}\right) = \mathsf{U}_{2i+1,j}^{L/2+1}\mathsf{G}_{2i+1,j}^{L/2}.
$$

Using the same definition as in (29) and (30) to stack up the matrices, we accordingly define $\mathsf{U}_i^{L/2+1}$ and $\mathsf{U}^{L/2+1}$. These definitions also allow us to write the transform in a concise form with properly stacked up $\mathsf{G}^{L/2}$:

$$
\mathsf{U}^{L/2} = \mathsf{U}^{L/2+1}\mathsf{G}^{L/2}.
$$

Viewing these transformation, it is clear that the row size of $\mathsf{U}_{2i,j}^{L/2+1}$ is that of $\mathsf{U}_{i,2j}^{L/2,t}$, which is half of that of $\mathsf{U}_{i,2j}^{L/2}$. Similarly, $\mathsf{U}_{2i,j}^{L/2+1}$ has the same rank r, and thus $\mathsf{U}_{2i,j}^{L/2+1}$ is of size $\frac{m}{2} \times \frac{mr}{2}$. As a summary, when one changes from L-th iteration to $L+1$, the number of blocks double in both row and column, while the rank of each sub-block is kept as r.

Perform this iteration recursively by $L/2$ steps, we finally arrive at the formulation of (27). At the final stage of the iteration, U^L as defined in the same way as in (29) is composed with N blocks along its diagonal, with each block being of $1 \times r$. As an illustration, a rank 1 approximation using the butterfly algorithm can shown visualized in Fig. 3.

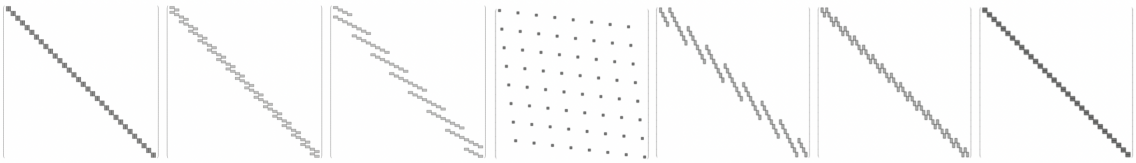

Figure 3: An illustration of the matrix factors in the butterfly factorization. In the illustration, $L = 6$, $r = 1$, and $N = 64$.

3 Neural Network Architecture

We design a neural architecture to conduct the inverse scattering in this section.

As discussed in the introduction, the implementation of $(F^\omega)^*$ and $((F^\omega)^* F^\omega + \epsilon I)^{-1}$ should honor the operators' own properties. In particular, we used convoluational NN to lift the filtering operator so it satisfies the translational equivariance. As argued before the back-scattering operator has rotational equivariance, and it is possible to compress it via a butterfly structure. We will discuss these properties in this section, and present how they get integrated in the design of NN in Section 3.1 and 3.2 respectively.

We write out architecture as

$$\eta = \Phi_\Theta(\{\Lambda^\omega\}_{\omega\in\bar{\Omega}}), \tag{31}$$

where the input $\{\Lambda^\omega\}_{\omega\in\bar{\Omega}}$ is a collection of far-field patterns as defined in (4), indexed by frequencies in $\bar{\Omega}$. Φ_Θ is the function generated by the neural network with neurons weighted using parameters Θ.

If we have only one frequency, i.e., $\bar{\Omega} = \{\omega\}$, then our architecture follows closely the filtered back-projection with a change of variables as described in Alg. 1. In practice we modify the application of F^ω by a neural network that mimics the butterfly structure, and we replace the filtering by several layers of convolutional networks.

Algorithm 1 Filtered Back-Projection.

Input: $\mathsf{\Lambda}^\omega \in \mathbb{C}^{\mathsf{n}_{\mathrm{sc}}\times\mathsf{n}_{\mathrm{sc}}}$
Output: $\eta = ((\mathsf{F}^\omega)^*\mathsf{F}^\omega + \epsilon\mathsf{I})^{-1}(\mathsf{F}^\omega)^*\mathsf{\Lambda}^\omega \in \mathbb{R}^{n_\eta\times n_\rho}$
1: Preparation of the normalized data (11);
2: Computation of back-projection $\alpha^\omega(\theta,\rho) = (\mathsf{F}^\omega)^*\mathsf{\Lambda}^\omega$ using (15);
3: Change of variables of α^ω to Cartesian coordinates $\alpha^\omega(\theta,\rho) \to \alpha^\omega(x,y)$ through polynomial interpolation;
4: Computation of the filtering operation $\eta = ((\mathsf{F}^\omega)^*\mathsf{F}^\omega + \epsilon\mathsf{I})^{-1}\alpha^\omega$ using (16).
5: **return** η

For the multiple frequencies case, we consider, as a motivation, the loss function

$$\eta^* = \operatorname{argmin} \sum_{\omega\in\bar{\Omega}} \|\Lambda^\omega - F^\omega[\eta]\|^2 + \epsilon\|\eta\|^2, \tag{32}$$

and by computing the first variation we have that

$$\eta^* = \sum_{\omega\in\bar{\Omega}} ((F^\omega)^* F^\omega + \epsilon I)^{-1}(F^\omega)^*\Lambda^\omega. \tag{33}$$

This can be easily implemented by applying Alg. 1 to each component and then average them at the end. However, as it was shown in [23] this approach does not mix the different scales efficiently. One alternative, would be to use the multilevel structure of the butterfly factorization of K^ω and merge the different scales hierarchically when applying the neural equivalent of F^ω. The implementation of this pipelines is however cumbersome and prone to errors. Instead we considered an intermediate architecture: we apply the *back-projection* independently for each frequency, we concatenate the resulting α^ω's along a new dimension, equivalent to a channel dimension for CNN. Finally, we use a *shared* filter acting in all frequencies together, so the mixing and weighting for data at different frequencies can be done progressively.

In a nutshell, for the general case, our neural network initially incorporates data of different frequencies when computing the back-projections. The α^ω for different frequencies ω are concatenated along a channel dimension, and they are merged when using a global filter operation. The method is summarized in Alg. 2.

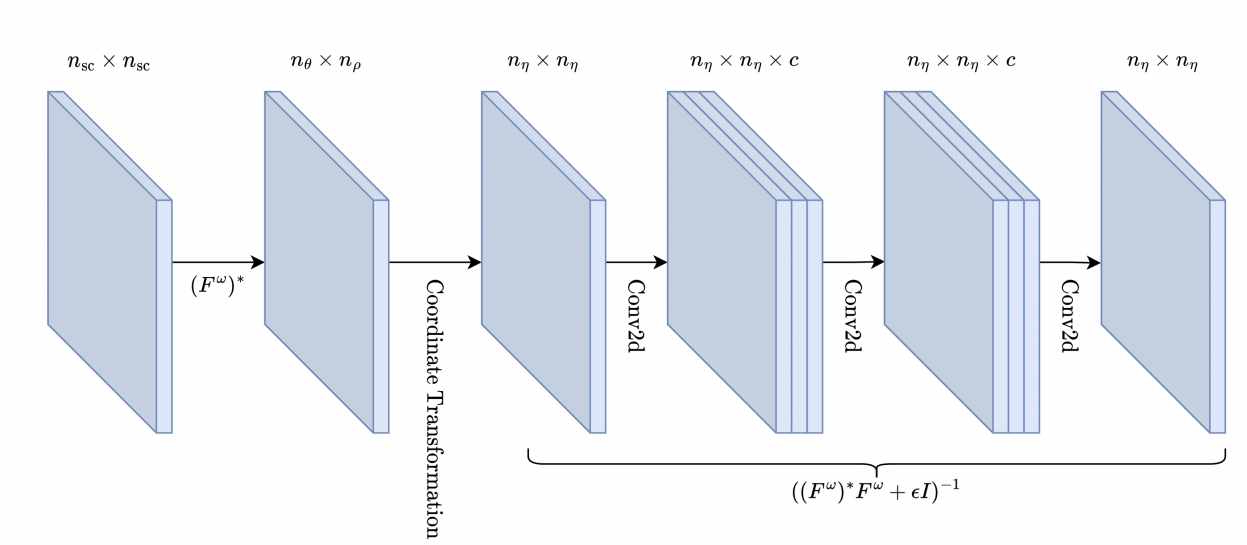

Figure 4: An illustration of the architecture of the model. The shape of each tensor at each step is labelled on the top. Convolutional NN is used to lift the filtering operator.

Algorithm 2 Wide-Band Equivariant Network.

Input: $\{\Lambda^\omega\}_{\omega\in\bar{\Omega}}$, where $\Lambda^\omega \in \mathbb{C}^{\mathsf{n}_{\text{sc}}\times \mathsf{n}_{\text{sc}}}$, for $\omega \in \bar{\Omega}$
Output: $\eta \in \mathbb{R}^{\mathsf{n}_\eta \times \mathsf{n}_\rho}$
1: **for** $\omega \in \bar{\Omega}$ **do**
2: Preparation of the normalized data (11);
3: Computation of back-projection $\alpha^\omega(\theta,\rho) = (\mathsf{F}^\omega)^*\Lambda^\omega$ using (15);
4: Change of variables of α^ω into Cartesian coordinates $\alpha^\omega(\theta,\rho) \to \alpha^\omega(x,y)$ through polynomial interpolation;
5: **end for**
6: Concatenation of $\{\alpha^\omega\}_{\omega\in\bar{\Omega}}$ into a 3d tensor $\mathbf{a}$ such that $\mathbf{a}[:,:,\omega] = \alpha^\omega$.
7: Computation of the filtering operation $\eta = ((\mathsf{F}^\omega)^*\mathsf{F}^\omega + \epsilon\mathsf{I})^{-1}\mathbf{a}$ using (16).
8: **return** η

In computation, the size and the values in the set of frequencies $\bar{\Omega}$ depends on the target resolution. We follow a dyadic partition for the frequencies. For instance, we will use data corresponding to source frequencies 2.5, 5.0, and 10.0 Hz for data of dimension $n_{\text{sc}} = 80$. Hence, the number of frequencies scale logaritmically with respect to n_{sc}.

3.1 Application of equivariance

Recall the formula (19), the backscattered data α^ω comes from two layers of integral operators, both of which involves the integration kernel $K^\omega(\rho,t) := e^{-i\omega\rho\cos(t)}$. Using the discretization introduced in Section 2.2, this function is presented using the matrix form as

$$\mathsf{K}^\omega \in \mathbb{R}^{n_{\text{sc}}\times n_\rho}, \quad \text{with} \quad \mathsf{K}^\omega_{mn} = e^{-i\omega\rho_n\cos(t_m)} = K^\omega(\rho_n,t_m)\,.$$

Recalling the discrete form of data (13), to fully implement the back-scattering operator (19), one can rewrite (19), for any θ_j 14, as:

$$\alpha^\omega(\theta_j,\cdot) = ((\mathsf{F}^\omega)^*\Lambda^\omega)(\theta_{\mathsf{j}},\cdot) = \underbrace{\text{ones}(1,n_{\text{sc}})\cdot[\overline{\mathsf{K}^\omega}\odot(\Lambda^\omega_{\theta_{\mathsf{j}}}\cdot\mathsf{K}^\omega)]}_{\text{Implementation I}} = \underbrace{\text{diag}[(\mathsf{K}^\omega)^*\cdot\Lambda^\omega_{\theta_j}\cdot\mathsf{K}^\omega]}_{\text{Implementation II}}\,. \tag{34}$$

Here $\cdot$ denotes the matrix multiplication and $\odot$ denotes the element-wise Hadamard multiplication. $\cdot^*$ is the notation for conjugate transpose. Λ^ω_θ is the discrete shifting of $\Lambda^\omega(r+\theta,s+\theta)$. Considering $\{\theta_j\} = \{r_j\}$, for the matrix, we directly shift all rows/columns by j:

$$\Lambda^\omega_{\theta_{\mathsf{j}}}(\mathsf{m},\mathsf{n}) = \Lambda^\omega(\mathsf{m}+\mathsf{j},\mathsf{n}+\mathsf{j})\,. \tag{35}$$

It should be noted that the two implementations are mathematically equivalent, although they are numerically different. The compression of K^ω drives the choice of the implementation. In particular, Implementation I replaces the first matrix-matrix-product using a Hadamard entry-wise product so it reduces the computational cost, and it is therefore preferred when there is no compression. On the other hand, this implementation is unfriendly to encode the butterfly structure. The butterfly structure requires the to-be-examined matrix to be expanded from both sides into many smaller-sized matrix products, and this is incompatible to the Hadamard product. As a consequence, when the butterfly-structure is used for compression, we turn to Implementation II.

Both implementations impose the integral kernel K^ω to be shared across different θ_j. This observation lies at the core of our algorithm. Its advantages are twofold: it automatically enforces the equivariance property encoded in (22), and it significantly reduces the computational cost.

Indeed, with this approach, the to-be-trained parameters are all encoded in K^ω, a small matrix of size $(n_{\mathrm{sc}} \times n_\rho)$. This is a significant saving compared with a brute-force computation. Starting from (19), α^ω is written as a linear operator acting on the data Λ^ω through an integral kernel. Following this more direct approach, the input, Λ^ω, will be flattened into a vector of size n_{sc}^2, and the output, α^ω, will flattened into a vector of size $n_\eta n_\rho$, therefore the integral kernel is represented by a significantly larger matrix of size $(n_\eta n_\rho) \times n_{\mathrm{sc}}^2$. This procedure not only ignores the fact that the kernel is independent of θ_j, but also requires $n_\eta n_{\mathrm{sc}}$ times more unknowns, increasing the overall computational cost.

In Figure 5, we visualize the application of the underlying equivariance on the back-scattering operator, using $n_{\mathrm{sc}} = n_\rho = n_\theta = 4$ as an example for Implementation II.

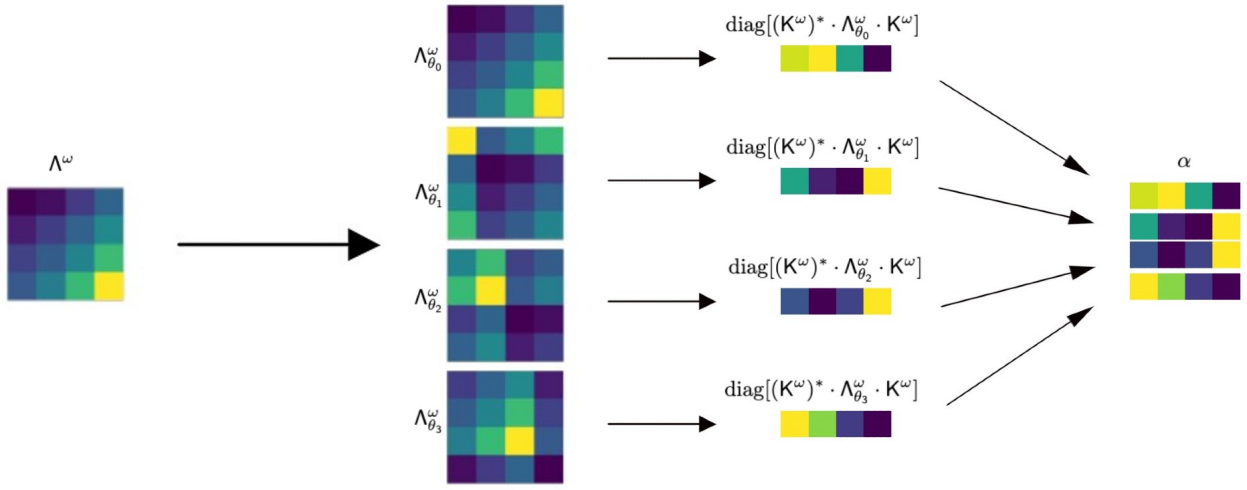

Figure 5: The visualization of the application of the underlying equivariance. In the first step, the data matrix Λ^ω is shifted to generate the other four Λ_{θ_j} for $j = 0, 1, 2, 3$. Then, the Implementation II $\mathsf{x} \mapsto \mathrm{diag}[(\mathsf{K}^\omega)^* \cdot \mathsf{x} \cdot \mathsf{K}^\omega]$ is applied to all four Λ_{θ_j}, each of which outputs a row vector. Finally, they are concatenated to form the intermediate representation α^ω.

In order to implement numerically the algorithm in Implementation I of (34), we note that the kernel can be decomposed as following

$$K^\omega = C^\omega - iS^\omega\,, \quad \text{with} \quad C^\omega(\rho, t) = \cos(\rho\omega\cos(t)) \quad \text{and} \quad S^\omega(\rho, t) = \sin(\rho\omega\cos(t))\,. \tag{36}$$

In this case, we have

$$\begin{aligned}((F^\omega)^*\Lambda^\omega)(\theta,\rho) = &\iint_{[0,2\pi]^2} C^\omega(\rho,r)C^\omega(\rho,s)\Lambda^\omega_R(r+\theta,s+\theta)\,ds\,dr\,,\\ &+\iint_{[0,2\pi]^2} S^\omega(\rho,r)S^\omega(\rho,s)\Lambda^\omega_R(r+\theta,s+\theta)\,ds\,dr\,,\\ &+\iint_{[0,2\pi]^2} C^\omega(\rho,r)S^\omega(\rho,s)\Lambda^\omega_I(r+\theta,s+\theta)\,ds\,dr\,,\\ &-\iint_{[0,2\pi]^2} S^\omega(\rho,r)C^\omega(\rho,s)\Lambda^\omega_I(r+\theta,s+\theta)\,ds\,dr\,,\end{aligned}$$

where Λ^ω has been periodically extended outside the domain $(r,s) \in [0,2\pi]^2$ torus.

As such, the training of K^ω can be translated to the training of C^ω and S^ω, which are the discrete forms of C^ω and S^ω using the same discretization as K^ω. Following the Implementation I in equation (34), we find:

$$\alpha^\omega(\theta_j,\cdot) = \mathrm{ones}(1,n_{\mathrm{sc}}) \cdot \left[\mathsf{C}^\omega \odot (\Lambda^\omega_{\theta_j,\mathsf{R}} \cdot \mathsf{C}^\omega) + \mathsf{S}^\omega \odot (\Lambda^\omega_{\theta_j,\mathsf{R}} \cdot \mathsf{S}^\omega) + \mathsf{C}^\omega \odot (\Lambda^\omega_{\theta_j,\mathsf{I}} \cdot \mathsf{S}^\omega) - \mathsf{S}^\omega \odot (\Lambda^\omega_{\theta_j,\mathsf{I}} \cdot \mathsf{C}^\omega)\right], \quad (37)$$

where $\Lambda^\omega_{\theta_j,\mathsf{R/I}}$ are shifted real/imaginary parts of the data as defined in (35). Clearly, the execution of $(\mathsf{F}^\omega)^*$ is written as the summation of four terms that share the same sequence of operations. Namely, each term is composed of one vector-matrix product, one Hadamard product, and one matrix-matrix product with the shifted data.

We summarize the implementation of (37), in which the parameters C^ω and S^ω , instead of being constant matrices given by (36), are to be trained using data. Also, we replace the constant matrix $\mathrm{ones}(1,n_{\mathrm{sc}})$ by trainable weights O^j for $j=1,2,3,4$ of the same size, so that we can get rid of the negative sign. We name the model as the uncompressed model.

Algorithm 3 The application of $(F^\omega)^*$ in the uncompressed model

Input: $\Lambda^\omega \in \mathbb{C}^{\mathsf{n_{sc}}\times \mathsf{n_{sc}}}$
Output: $(\mathsf{F}^\omega)^*\Lambda^\omega \in \mathbb{C}^{\mathsf{n}_\eta \times \mathsf{n}_\rho}$
1: Split the data in real and imaginary parts: $\Lambda^\omega = \Lambda^\omega_{\mathsf{R}} + \mathsf{i}\Lambda^\omega_{\mathsf{I}}$
2: **for** $j < n_{sc}$ **do**
3: $\quad \alpha^\omega[j,:] \leftarrow \mathsf{O}^1 \cdot (\mathsf{C} \odot (\Lambda^\omega_{\theta_j,\mathsf{R}} \cdot \mathsf{C})) + \mathsf{O}^2 \cdot (\mathsf{S} \odot (\Lambda^\omega_{\theta_j,\mathsf{R}} \cdot \mathsf{S})) + \mathsf{O}^3 \cdot (\mathsf{C} \odot (\Lambda^\omega_{\theta_j,\mathsf{I}} \cdot \mathsf{S})) + \mathsf{O}^4 \cdot (\mathsf{S} \odot (\Lambda^\omega_{\theta_j,\mathsf{I}} \cdot \mathsf{C}))$
4: **end for**
5: **return** α^ω

We note that the model presented in this section only utilizes the underlying equivariance. In the application of $(F^\omega)^*$, we represent the maps S^ω and C^ω directly as $n_{\mathrm{sc}} \times n_{\mathrm{sc}}$ matrices consisting of trainable weights and the size of $\bar{\Omega}$ scales logarithmically with respect to n_{sc} following a dyadic partitionning, which ensure that we capture high-frequency information while still taking advantage of the regularization power that low frequency data provides to the algorithm. Hence, the number of trainable weights scales as $\mathcal{O}(n_{\mathrm{sc}}^2 \log n_{\mathrm{sc}})$, and the inference complexity is dominated by the computations of $n_{\mathrm{sc}} \times n_{\mathrm{sc}}$ matrices productions for a total of $4n_{\mathrm{sc}} \log n_{\mathrm{sc}}$ times, which scales $\mathcal{O}(n_{\mathrm{sc}}^4 \log n_{\mathrm{sc}})$.

Since the filtering operator is approximated by a 2-dimensional convolutional NN with constant-sized kernel, for which the size of filters and the number of layers scale at most linearly with respect to n_{sc}, the number of trainable parameters scales $\mathcal{O}(n_{\mathrm{sc}}(\log n_{\mathrm{sc}})^2)$, and the inference complexity is $\mathcal{O}(n_{\mathrm{sc}}^3(\log n_{\mathrm{sc}})^2)$. It should be noted that the $\log n_{\mathrm{sc}}$ term comes from the dependence of the channel dimension on the number of frequencies.

Therefore, for the uncompressed model, the total number of parameters scales $\mathcal{O}(n_{\mathrm{sc}}^2 \log n_{\mathrm{sc}})$ and the inference complexity is $\mathcal{O}(n_{\mathrm{sc}}^4 \log n_{\mathrm{sc}})$.

We should note that the equivariance in our setting is hardcoded into the neural-network design. The feature of equivariance is widely observed in many other scientific domains, such as computer vision, reinforcement learning, dynamics learning, or protein folding. This triggers the studies on preserving gauge invariance structure in its very general form, see for example [45] and [46]. In the context of inverse scattering, we have found [25] that also places focuses on preserving rotational equivariance. The compression, however, was not incorporated in their study.

3.2 Application of the butterfly factorization

For the application of butterfly factorization, we adopt Implementation II in (34) of the discretized adjoint operator $(F^\omega)^*$, which results on the formula

$$\alpha^\omega(\theta_j,\cdot) = ((\mathsf{F}^\omega)^*\Lambda^\omega)(\theta_{\mathrm{j}},\cdot) = \mathrm{diag}[(\mathsf{K}^\omega)^* \cdot \Lambda^\omega_{\theta_{\mathrm{j}}} \cdot \mathsf{K}^\omega].$$

Notice that in Section 2.5 we have shown that K^ω admits a butterfly factorization, namely, it can be factorized as in (27). As a direct consequence, $(\mathsf{K}^\omega)^* \cdot \Lambda^\omega_{\theta_{\mathrm{j}}} \cdot \mathsf{K}^\omega$ is expanded as:

$$\mathsf{V}^L\mathsf{H}^{L-1}\cdots\mathsf{H}^{L/2}(\mathsf{M}^{L/2})^*(\mathsf{G}^{L/2})^*\cdots(\mathsf{G}^{L-1})^*(\mathsf{U}^L)^*\cdot\Lambda^\omega\cdot\mathsf{U}^L\mathsf{G}^{L-1}\cdots\mathsf{G}^{L/2}\mathsf{M}^{L/2}(\mathsf{H}^{L/2})^*\cdots(\mathsf{H}^{L-1})^*(\mathsf{V}^L)^*. \quad (38)$$

The form of (38) suggests the overall structure of the operator is composed of taking actions on the data $\Lambda^\omega_{\mathsf{R}}$ using $L+3$ layers:

- Layer $U_{layer}(\mathsf{x})$: In this layer, we apply the most internal action on the data: we sandwich the original data $\Lambda^{\omega}_{\mathsf{R}}$ by U^L, namely:
$$\mathsf{x} \to (\mathsf{U}^L)^* \cdot \mathsf{x} \cdot \mathsf{U}^L \,.$$
- Layer $G_{layer}(\mathsf{x}, \ell)$: there are $L/2$ $G_{layer}(\mathsf{x}, \ell)$ layers corresponding to $\ell = L/2\,, \cdots, L-1$, each of which sandwich the given data by the associated G^i:
$$\mathsf{x} \to (\mathsf{G}^\ell)^* \cdot \mathsf{x} \cdot \mathsf{G}^\ell\,, \quad \ell = L/2\,, \cdots, L-1\,.$$
- Layer SwitchResnet(x): In the middle layer, we are supposed to sandwich the given data by $\mathsf{M}^{L/2}$:
$$\mathsf{x} \to (\mathsf{M}^{L/2})^* \cdot \mathsf{x} \cdot \mathsf{M}^{L/2}\,.$$
Additionally, inspired by [23], we also lift the middle layer by integrating a Resnet structure into it. We call it a SwitchResNet. The rational will be explained in the description of the implementation later in this section.
- Layer $H_{layer}(\mathsf{x}, \ell)$: similar to the $G_{layer}(\mathsf{x}, \ell)$ layer.
- Layer $V_{layer}(\mathsf{x})$: similar to the $U_{layer}(\mathsf{x})$ layer.

We note that the application of K^{ω} assumes small perturbation around η_0 in the linearized setting, and to lift it up to deal with nonlinear inverse scattering, we still would like to honor these symmetries. In computation, we keep the linearity of every action except lifting the application of $\mathsf{M}^{L/2}$ and its adjoint $(\mathsf{M}^{L/2})^*$ to the nonlinear setting and representing them by a ResNet. This is summarized in the pseudo-code in Algorithm 4, see also [23]. We name the corresponding model as the compressed model.

Algorithm 4 The application of $(F^{\omega})^*$ in the compressed model.

Input: $\Lambda^{\omega} \in \mathbb{C}^{\mathsf{n_{sc}} \times \mathsf{n_{sc}}}$
Output: $(\mathsf{F}^{\omega})^* \Lambda^{\omega} \in \mathbb{C}^{\mathsf{n}_\eta \times \mathsf{n}_\rho}$
$\mathsf{x} \leftarrow U_{layer}(\Lambda^{\omega})$
for ℓ in $\mathrm{range}((L-1), L/2-1, -1)$ **do**:
    $\mathsf{x} \leftarrow G_{layer}(\mathsf{x}, \ell)$
end for
$\mathsf{x} \leftarrow \mathrm{SwitchResnet}(\mathsf{x})$
for ℓ in $\mathrm{range}(L/2, L+1, +1)$ **do**:
    $\mathsf{x} \leftarrow H_{layer}(\mathsf{x}, \ell)$
end for
$\mathsf{x} \leftarrow V_{layer}(\mathsf{x})$
return x

The reasoning for choosing everything linear but lifting the middle layer to SwitchResnet stems from the fact different layers are in charge of exchanging information at different levels. In particular, seen from Figure 2, U^L and V^L are block-diagonal matrices and thus are automatically in charge of local information change. G^l and H^l are in charge of exchanging information across neighboring blocks and thus the action is also local. $\mathsf{M}^{L/2}$ finally is responsible for capturing the inherent non-locality of wave scattering. Since this action is taken on the condensed representation of the measured data, we choose to represent it using a non-linearly process.

There are multiple trade-offs in the implementation of different layers, and we glean through them below. As an example, we assume $\Lambda^{\omega}_{\mathsf{R}}$ is an 80×80 matrix. As implied by the Implementation II of (34), $(\mathsf{K}^{\omega})^*$ (and similarly K^{ω}) is applied on each individual column (row) of $\Lambda^{\omega}_{\mathsf{R}}$ viewed as an independent vector, so below we describe the application of the network on one of the columns of the data matrix, which is an 80 dimensional vector. To deal with this vector, we divide it into 2^4 chunks with each chunk containing 5 entries. As a result, calling $80 = 2^4 \times 5$, we set $L = 4$ and $s = 5$.

– **Implementing the U_{layer} and V_{layer} layers:**
As shown in Figure 2, in the U_{layer} and V_{layer} layers, we sandwich the data by the block matrices U^L and V^L. In the U_{layer} layer, each block of U^L, denoted as U^L_i with $i = 1\,, \cdots 2^L = 16$ extracts a local representation of the i-th section of the vector (an s-dimensional vector) and represent it by a r-dimensional vector. Used in our setting, each of U^L_i, when applied to the $s = 5$-dimensional vector (the i-th section of the vector), produces a r-dimensional vector. In total, the procedure generates a $2^L r = 16r$-dimensional vector. The number of trainable weights in the matrix is therefore $2sr2^L$ after separating the real and complex channels. Similarly for the V_{layer} layer, each block of V transforms an r-dimensional local representation to a s-dimensional sampling points, which also requires $2sr2^L$ trainable weights.

– **Implementing the G_{layer} and H_{layer} layers:**
In the $G_{layer}(\mathsf{x}, \ell)$ layer, $(\mathsf{G}^{\ell})^*$ and G^{ℓ} are applied to x, and in the $H_{layer}(\mathsf{x}, \ell)$ layer, $(\mathsf{H}^{\ell})^*$ and H^{ℓ} are applied to x. Observing the Figure 2, we see that in the G_{layer} layer, two neighboring local representations are assimilated. Similarly, in the H_{layer} layer, information in the representations will be decompressed by being splitted in two and locally redistributed. To implement the G_{layer} and H_{layer} layers, we decompose the corresponding matrices into row or column transformations and block matrices, so that after the row or column transformations, the application of the block matrices can be implemented the same way as in U_{layer} and V_{layer} layers. The number of trainable weights in both of the layers is therefore $4r^2 2^L$.

– **Implementing the M_{layer} layer:**
In the M_{layer} layer, information are redistributed globally by the matrices $(\mathsf{M}^{L/2})^*$ and $\mathsf{M}^{L/2}$. The application of M_{layer} layer is essentially the same as the G_{layer} and H_{layer} layers, where we apply the corresponding row or column transformations following by block matrices. Inspired by [23], we also integrate a non-linear module into the model by adding a Resnet of depth n_{SR} in the M layer. Hence, we have a total of $2n_{\mathsf{SR}} r^2 2^L$ trainable weights.

Compared to the uncompressed model, the compressed model exhibits lower asymptotic scaling of the number of trainable parameters and a lower inference complexity, albeit still super-linear. As explained at the end of Section 3.1, the number of trainable parameters of the filtering operator scales $\mathcal{O}(n_{\text{sc}}(\log n_{\text{sc}})^2)$ and the inference complexity of the filtering operator is $\mathcal{O}(n_{\text{sc}}^3(\log n_{\text{sc}})^2)$. Nevertheless, for the application of $(F^{\omega})^*$ in the compressed model, the number of parameters scales $(4sr2^L + 8Lr^2 2^L + 2n_{\mathsf{SR}} r^2 2^L) \log n_{\text{sc}} = \mathcal{O}(r^2 n_{\text{sc}}(\log n_{\text{sc}})^2)$, and the inference complexity is $\mathcal{O}(r^2 n_{\text{sc}}^3(\log n_{\text{sc}})^2)$.

Therefore, for the compressed model, the complexity of the number of trainable parameters is $\mathcal{O}(r^2 n_{\text{sc}}(\log n_{\text{sc}})^2)$ and the inference complexity is $\mathcal{O}(r^2 n_{\text{sc}}^3(\log n_{\text{sc}})^2)$. In Table 1, we present the complexity of the trainable parameters and the inference time complexity for the compressed model and the uncompressed model.

Complexity	Compressed model	Uncompressed model
Parameters	$\mathcal{O}(r^2 n_{\text{sc}}(\log n_{\text{sc}})^2)$	$\mathcal{O}(n_{\text{sc}}^2 \log n_{\text{sc}})$
Inference time	$\mathcal{O}(r^2 n_{\text{sc}}^3(\log n_{\text{sc}})^2)$	$\mathcal{O}(n_{\text{sc}}^4 \log n_{\text{sc}})$

Table 1: Complexity of the number of trainable parameters and inference complexity between the uncompressed and the compressed models. The complexity of the uncompressed and compressed models was discussed in the Sections 3.1 and 3.2, respectively.

4 Numerical Results

In this section, we provide numerical evidences to showcase the capabilities of the current algorithm. In summary, what to be presented below suggest two findings:

- Uncompressed model has the equivariance built in the formulation, leading to fewer trainable parameters. The amount of data needed for the training is consequently reduced. The performance of the numerical results are competitive compared to classical FWI and other NN models (such as wide-band butterfly network and Fourier neural operator) even with smaller amount of training points.
- Compressed model further reduces the number of trainable parameters by building in the butterfly structure. The performance of the model is biased towards media that are relatively smooth: The reconstruction of smooth media is accurate while the model losses the capability of capturing sub-Nyquist features.

The training optimization formulation is presented in subsection 4.1. We then discuss data structure in subsection 4.2. These will be followed by subsection 4.3 that discusses the uncompressed models. In subsections 4.5, we systematically compare reconstruction using various of means: The uncompressed model, the compressed model, the classical FWI and two other well-accepted NN models.

The data was generated using Matlab on a ARM-based MacBook Air (M1, 2020). The script usually took about 12 hours to generate a multi-frequency datasets of dimension $n_{\text{sc}} = 80$, of size 10000, and with source frequencies 2.5, 5.0, and 10.0 Hz. The models presented in this paper were implemented using Tensorflow (2.4.1) [47] and ran on a PNY NVIDIA Quadro RTX 6000 graphics card.

4.1 Optimization

As before, we denote our neural network as Φ_Θ with parameters Θ and the set of frequencies of the data begin fed to the network by $\bar{\Omega}$, namely our network takes the form:

$$\eta = \Phi_\Theta(\{\Lambda^\omega\}_{\omega\in\bar{\Omega}})\,. \tag{39}$$

For training we use tuples of media and scattering data, $(\eta^{[s]}, \{\Lambda^{\omega,[s]}\})$, where $[s]$ is the index for the training set. We point out that different problems will have different $\bar{\Omega}$ depending of the target resolution, also the network will change depending on whether the operators are compressed.

We train the networks by minimizing the mean square error between the network produced media and the groundtruth media (the media used to generate the input data), i.e.,

$$\min_\Theta \frac{1}{N_s}\sum_{s=1}^{N_s} \|\Phi_\Theta(\{\Lambda^{\omega,[s]}\}_{\omega\in\bar{\Omega}}) - \eta^{[s]}\|^2\,. \tag{40}$$

We use Adam optimizer with learning rate chosen as 3×10^{-4}, and batch size as 16 together with an exponential scheduler. The learning scheduler was set as Tensorflow's [47] `ExponentialDecay` with a decay rate of 0.96 after every 50 plateaus steps with `staircase` option set as true. We chose the Adam optimizer [48] and we train the model after 100 epochs. The trainable weights were all randomly initialized with the glorot_uniform distribution [49]. For comparison we report the relative root mean square error (RMSE) given by

$$\frac{1}{N_t}\sum_{s=1}^{N_t} \frac{\|\Phi_\Theta(\{\Lambda^{\omega,[s]}\}_{\omega\in\bar{\Omega}}) - \eta^{[s]}\|}{\|\eta^{[s]}\|}\,, \tag{41}$$

where N_t is the size of the testing set.

4.2 Datasets

The datasets consist of media and corresponding wide-band far-field patterns at three different frequencies, which are rescaled appropriately depending on the target resolution. The far-field patterns as the data were generated by solving for equation (3) using numerical finite difference method in Matlab. The computational domain was $[-0.5, 0.5]^2$ discretized with a equispaced mesh of 80×80 points for medias of resolution 80×80 pixels ($n_\eta = 80$). The radiation boundary conditions was implemented using the perfectly matched layer (PML)[50] with order 2 and intensity 80. The wide-band data was sampled with a homogeneous background wave field at source frequencies 2.5, 5, and 10 Hz, see Section 2, for which the effective wavelength is 8 points per wavelength (PPW). In particular, we use $n_{\rm sc} = 80$ receivers and sources, where receivers geometry are aligned with the directions of sources, i.e. 80 equiangular directions. For media of different resolutions, we generate their far-field patterns by sampling at proportionally scaled frequencies.

In our experiments we use 5 different categories of media:

- The well-known Shepp-Logan phantom, which was created in 1974 by Larry Shepp and Benjamin F. Logan to represent a human head [51]. The medium has a strong discontinuity modeling an uneven skull, which produced a strong reflection, which in return renders the recovery of the interior features challenging for classical methods.
- Random smooth perturbations, which are generated by smoothing out some randomly distributed points of some random values by a Gaussian kernel. They are used to study the behavior of the algorithm in the case of diffraction.
- Media consisting of triangles of different sizes, in particular, triangles of size 3, 5 and 10 number of pixels, which are randomly located and oriented, and it is possible for them to overlap with each other. In this case we test the capacity of the algorithm to image consisting of small scatterers that are slightly below sub-Nyquist in size. For clarity we name the dataset following the sizes of the triangles, e.g., '10h triangles' are composed of triangles with side length of 10 pixels.

In Figure 6, we provide one example for each of the five category. The training and the test are all conducted within one category.

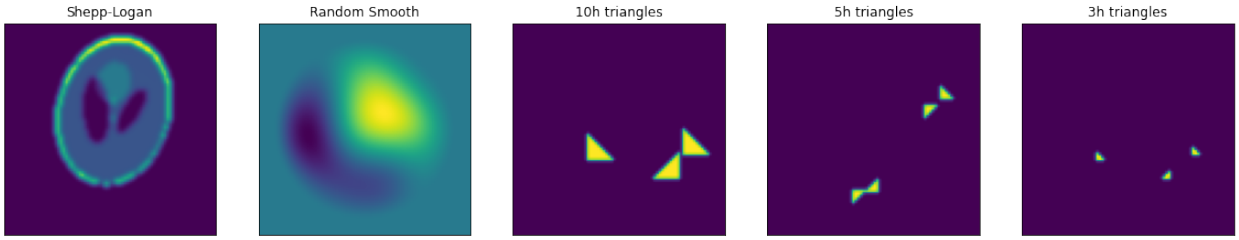

Figure 6: Examples of the five different media used to bechmarkthe model (from left to right) Shepp-Logan phantom, smooth random perturbation, and the rotating triangles of different sizes.

4.3 Uncompressed Model

The uncompressed model has the equivariance built in. As a consequence, only a small number of parameters need to be trained, which in return requires fewer numbers of samples for the training. In what follows we demonstrate a few properties of uncompressed model. In particular,

1. **Low sample complexity**: the validation error quickly saturates as we increase the number of training points;

2. **Improved stability with wide-band data**: similarly to the classical FWI method, which produces better reconstruction with wide-band data, the uncompressed model provides better reconstruction when data at multiple frequencies are provided;

3. **Improved accuracy with higher frequency data**: when the models are trained on media of finer resolution and scattered data of higher dimension that are generated by probing waves of higher frequencies, it produces results with lower relative validation error. In particular, it recovers the fine-grained details better.

In Table 2, we show how the relative validation error depends on the number of training data points, and probing wave frequency for different media. The scattered field is generated using $n_{\text{sc}} = 80$ and the media in a domain is discretized with an equispaced mesh of $n_\eta = 80$. The number of trainable parameters for reconstruction with one frequency is 46530 and that for reconstruction with three frequencies is 88186. Table 2 shows that using wide-band data consistently produces more accurate reconstructions; the equivariance property helps to drastically reduce the number of training points (the accuracy has already stagnated using a few thousands training points). Figure 7 illustrates one representative instance of the reconstruction for each of the different media mentioned above, using different numbers of training points. In this case, the model uses wide-band frequency datasets.

Moreover, we also observe in Table 2 the unstable behavior of the reconstruction for the random smooth perturbations, particularly in the diffractive regime, in the sense that the error of reconstruction increases as the frequency increases for a single frequency, which is often a sign of cycle-skipping. This is solved by using wide-band data. The model is able to super-resolve some simple geometrical figures, such as the small rotating triangles, which is often a challenging problem using classical optimization/signal processing tools. We are able to reconstruct triangles of size 3 pixels, even if the wavelength is around 8 pixels.

We also investigate how the number of trainable parameters scale according to the dimension of the data (n_{sc}). Richer data is expected to produce better reconstruction. In our setting, as the data increases its dimension (n_{sc} becomes big), we expect the model to produce better results. To do so, we first randomly generate 1024 samples of media at a native resolution with $n_\eta = 480$ and downsample them to a coarser resolution using $n_\eta = 60, 80, 120$ and 160. At each coarse level of the media, we view the media as the reference and generate the wide-band far-field patterns using $n_{\text{sc}} = n_\eta$, so that the frequencies of the probing waves scale proportionally to n_{sc}. This builds the dataset for the training. Upon validation, one projects the reconstruction back to the native resolution at $n_\eta = 480$. In Table 3, we present at different coarse resolution, the number of trainable parameters, the average relative validation error at the training resolution level, and the relative validation error when the reconstruction gets interpolated back to the finest level (termed Rel err native res). In Figure 8 we plot one example of the reconstruction at different resolution levels.

Shepp–Logan Phantom				
#Sample \ Frequency	2.5 Hz	5 Hz	10 Hz	2.5 & 5 & 10 Hz
64	12.430 %	10.119 %	15.938 %	13.236%
128	10.619 %	7.919 %	11.403 %	8.293 %
256	10.334 %	6.754 %	9.798 %	6.503 %
512	10.077 %	6.935 %	7.954 %	5.124 %
1024	10.774 %	6.721 %	8.125 %	5.306 %
Random Smooth Perturbation				
#Sample \ Frequency	2.5 Hz	5 Hz	10 Hz	2.5 & 5 & 10 Hz
128	6.446 %	10.323 %	22.811 %	7.182 %
256	5.240 %	7.649 %	21.127 %	5.456 %
512	4.490 %	7.386 %	11.681 %	4.898 %
1024	4.385 %	9.029 %	10.641 %	3.957 %
2048	5.386 %	8.237 %	11.403 %	4.102 %
10h triangles				
#Sample \ Frequency	2.5 Hz	5 Hz	10 Hz	2.5 & 5 & 10 Hz
256	47.911 %	18.432 %	14.858 %	15.701 %
512	44.270 %	14.010 %	11.066 %	9.012 %
1024	40.767 %	11.777 %	9.151 %	7.777 %
2048	41.380 %	11.551 %	8.297 %	7.068 %
5h triangles				
#Sample \ Frequency	2.5 Hz	5 Hz	10 Hz	2.5 & 5 & 10 Hz
256	59.616 %	37.625 %	11.020 %	12.611 %
512	50.114 %	22.679 %	8.680 %	8.842 %
1024	49.184 %	16.892 %	6.963 %	6.061 %
2048	47.257 %	16.036 %	6.004 %	6.230 %
3h triangles				
#Sample \ Frequency	2.5 Hz	5 Hz	10 Hz	2.5 & 5 & 10 Hz
256	59.327 %	33.156 %	12.054 %	12.774 %
512	56.135 %	39.521 %	7.499 %	10.209 %
1024	42.683 %	24.825 %	6.864 %	6.597 %
2048	43.977 %	19.458 %	6.472 %	5.902 %

Table 2: Relative root mean square test error of reconstruction given by the equivariant uncompressed model for difference media, using data at different frequencies, and with different sizes of training set. Each experiment consisted of #Sample training points as recorded in the leftmost column and was tested against an independent testing dataset with 200 points. Each table denotes a set of experiments with the type of media indicated by the table's name. Experiments on the first three column use monochromatic data sampled at source frequencies 2.5, 5, or 10 Hz, and experiments on the last column use wide-band data sampled consisting of data sampled at all three source frequencies.

Shepp-Logan				
Resolution \ Attributes	#Parameters	Av inference time (s)	Rel err train res	Rel err native res
60	70906	0.015	9.623 %	27.719 %
80	88186	0.030	7.286 %	22.854 %
120	137146	0.090	6.370 %	11.192 %
160	205306	0.264	5.586 %	7.223 %

Table 3: Statistics of the equivariant models for different resolutions of the Shepp-Logan dataset. The table shows the number of trainable parameters and the average inference time for the uncompressed model for Shepp-Logan phantom training points of resolution $n_\eta = 60, 80, 120$, and 160. The last two columns records relative errors at the training resolution and at the native resolution for each test.

It is clear that if the datasets are prepared at a finer resolution, the number of trainable parameters increases accordingly, and so does the training time. The validation errors, both the relative error on the training resolution, and the relative error when projected back to the native resolution, also decrease.

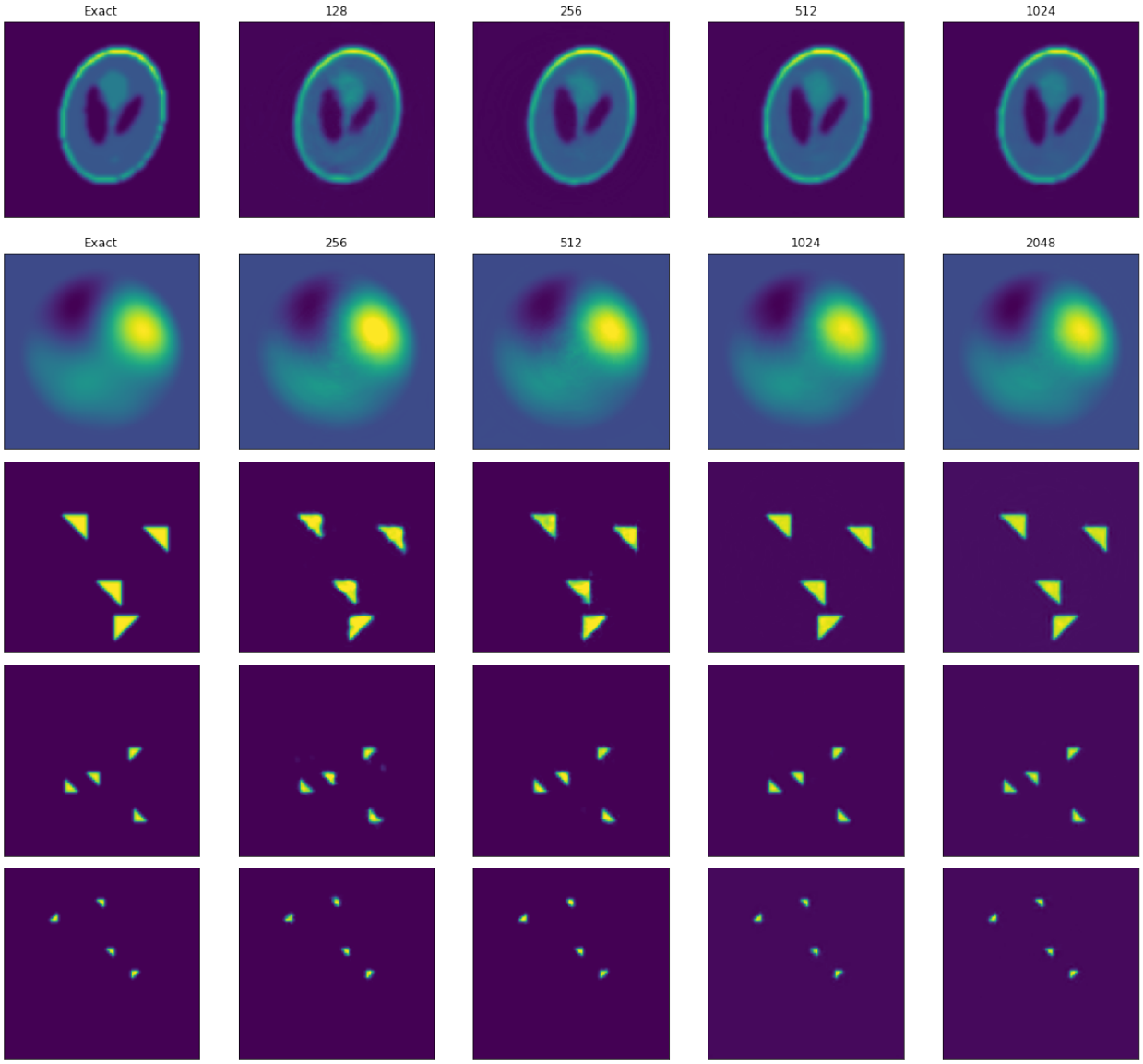

Figure 7: Illustration of the numerical performance of the Uncompressed model. Each row presents results for a different media using different numbers of training points in the training process. The exact media is shown in the left-most column. The reconstruction utilizes the wide-band frequency datasets.

4.4 Validation of the Rotational Equivariance

We demonstrate the rotational equivariance of the back-scattering operator and the translational equivariance of the filtering operator, which are hard-wired into the neural network. In particular, we create four datasets by manually rotate the entire training set by a certain degree (0, 90, 180 and 270 degrees in four experiments), we then train four models: each corresponding to each new dataset. We consider as a base training set, the set of 5h triangles containing 2048 data pairs. After training, we test each model on 500 data points for validation. Table 4, shows the average relative validation error for each rotated dataset. We can observe that regardless of the training data, the mean validation error of the reconstruction remains constant. In addition, Figure 9, shows one testing example of the reconstruction at different rotations: the reconstructed result is universally accurate across different rotations.

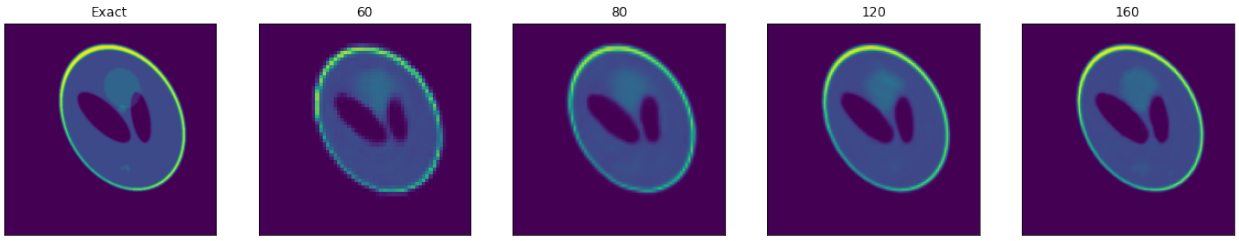

Figure 8: Instances of the reconstruction of the Shepp-Logan phantom at different resolutions using the uncompressed method. In particular, the back-scattering operation of the method has more trainable weights catering to the larger input size, and the filtering operation, approximated by the 2D convolutional NN, has the same kernel size, number of filters and depth for all experiments. The left to right: the original Shepp-Logan phantom at its native resolution $n_\eta = 480$, the other four pictures are reconstructions of the downsampled media of size: $n_\eta = 60, 80, 120$ and 160 using training points of corresponding dimension: $n_{\text{sc}} = 60, 80, 120$ and 160, repectively. The training points of higher resolutions are generated with probing wave of higher frequencies: $\frac{2.5}{80} n_{\text{sc}}$, $\frac{5}{80} n_{\text{sc}}$, and $\frac{10}{80} n_{\text{sc}}$ Hz.

Rotation (Degree)	Validation Error
0°	5.599 %
90°	5.586 %
180°	5.573 %
270°	5.588 %

Table 4: The validation errors of the neural network model for datasets rotated at 0, 90, 180, and 270 degrees.

4.5 Compressed Model

In this section we present the numerical results produced by the compressed model, for which the numerical performance of the uncompressed model in Section 4.3 established a baseline. Moreover, we also include the numerical results from two widely accepted NN models: the wide-band butterfly network (WBBN) from [23], and the Fourier neural operator (FNO) from [21] to form a comparison baseline with other ML methods. A brief presentation of WBBN and FNO can be found in the appendices. Since the compressed model calls for the construction of the butterfly factorization, the number of trainable parameters is further reduced.

Throughout our simulation, we use training points of resolution at $n_\eta = 80$. This translates to, using the notation from Section 3.2, $L = 4$ and $s = 5$. To train the uncompressed model, the compressed model, the WBBN and the FNO we use datasets consisting of far-field pattern data of dimension $n_{\text{sc}} = 80$. The training set contains 2048 data points and the test set consists 500 data points. The batch size is chosen to be 16 for all the tests.

In Figure 10 we compare the uncompressed model result, compressed model result with the classical FWI that is performed by running the optimization with the lowest frequency first and using the previous results for the optimizations with the higher frequencies iteratively. It is clear that:

1. **Out-performance over FWI:** Both the uncompressed model and the compressed model achieve higher accuracy on all five kinds of media than the classical FWI;
2. **Loss of super-resolution:** The compressed model has difficulty capturing super-resolution. It performs relatively well for 5h triangles, but lost the super-resolution for 3h triangle cases.

For the WBBN model from [23], we test the model using the same hyperparameters as chosen in the paper. The initial learning rate was set as 5×10^{-3} and the scheduler was set as Tensorflow's [47] ExponentialDecay with a decay rate of 0.95 after every 2000 plateaus steps with staircase set to true. Adam optimizer [48] is employed and we terminate training after 150 epochs. Additionally, all trainable weights were randomly initialized with glorot_uniform.

To train FNO, the initial learning rate was set to be 10^{-3} and the scheduler was set as Pytorch's [52] `StepLR` module with decay rate of 0.5 after every 100 plateaus steps with the `staircase` option set to true. Similarly, Adam optimizer [48] is employed and the training is terminated after 100 epochs.

The number of trainable parameters and the validation errors are listed in Table 5. In the case of Shepp-Logan phantom and the smooth perturbations, the performance of all four NN models is comparable even though the numbers of trainable parameters are drastically smaller for the compressed and uncompressed models, the two models proposed in the current paper.

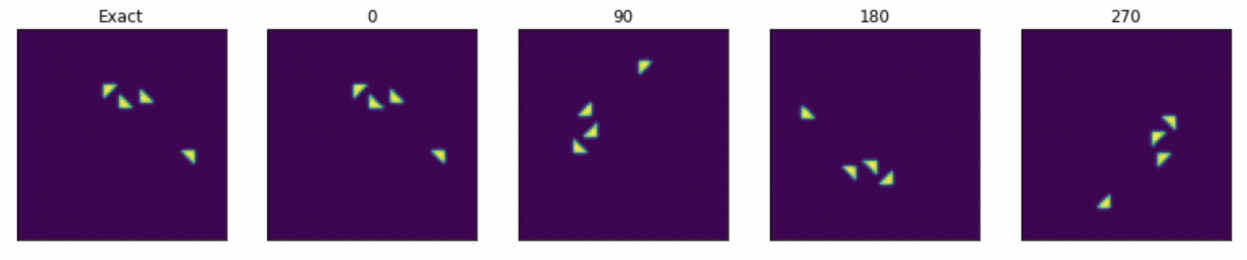

Figure 9: Instances of the reconstruction of the 5h triangles rotated at 0, 90, 180, and 270 degrees.

However, for the small triangles that are below sub-Nyquist in size, both WBBN and FNO produce considerably worse results than our proposed method. It should be noted that, for both WBBN and FNO, the numbers of trainable parameters are significantly greater than those of the uncompressed model and the compressed model, which possibly results in over-fitting of the model when trained with a dataset of size 2048. Indeed, when trained with larger datasets, the WBBN and the FNO produce much better results. In the Table 6, we list the validation errors of the WBBN and FNO with trained with 20000 data points using 10h, 5h, and 3h triangles.

Media \ Model	Uncompressed	Compressed	WBBN	FNO
#parameters	88,186	73,210	1,914,061	1,188,385
Shepp-Logan	5.124 %	8.306 %	9.819 %	7.981 %
Random smooth	3.957 %	6.866 %	8.268 %	4.289 %
10h triangles	7.068 %	15.751 %	42.545 %	42.633 %
5h triangles	6.061 %	16.144 %	Untrainable	47.692 %
3h triangles	5.902 %	38.651 %	Untrainable	49.348 %

Table 5: Comparison of the relative RMSE for various NN models in the five different media categories. The media reconstructed have resolution $n_\eta = 80$. The data is generated using probing wave of frequencies 2.5, 5, and 10 Hz, and the number of data points in the training set is 2048. Notice that the numbers of trainable parameters of the compressed model and the uncompressed model only have a small difference because of a relatively large constant in the complexity of the latter.

Media \ Model	WBBN	FNO
Random smooth	1.712 %	2.253 %
10h triangles	2.858 %	8.588 %
5h triangles	3.521 %	30.004 %
3h triangles	5.119 %	14.675 %

Table 6: The validation errors of the WBBN and FNO in the three media that contains small scattered are slightly below sub-Nyquist in size. The number of the data points is 20000.

5 Conclusion

In this manuscript, we perform numerical study on using various neuron networks to solve the 2D inverse scattering problem. In particular, we propose to build equivariance properties and the butterfly structures in the NN architecture design. By leveraging the underlying equivariance of the problem, we propose a model (termed the uncompressed model) that respects the equivariance of the back-scatter operator and reduces the number of trainable parameters, and by incorporating butterfly expansion structure we propose the second model (termed the compressed model) that further reduces the number of trainable parameters. The two models both outperform the classical FWI. For smooth media, they perform equally well as other NN architecures with fewer trainable parameters and smaller training sets, but the compressed model lost the super-resolution on sub-Nyquist features during the compression procedure.

6 Acknowledgement

The work of Q.~L. and B.~Z. is supported in part by the UW-Madison Data Initiative and the Office of Naval Research under the grant ONR-N00014-21-1-2140. The work of L.~Z.-N. is supported in part by the National Science Foundation

under the grant DMS-2012292. In addition, Q.~L. and L.~Z.-N. are supported by the NSF TRIPODS award 1740707. The views expressed in the article do not necessarily represent the views of the any funding agencies. The authors are grateful for the support.

References

[1] Brett Borden. Mathematical problems in radar inverse scattering. *Inverse Problems*, 18:R1 – R28, 2001.

[2] C. H. Greene, P. H. Wiebe, J. Burczynski, and M. J. Youngbluth. Acoustical detection of high-density krill demersal layers in the submarine canyons off georges bank. *Science*, 241(4863):359–361, 1988.

[3] Arthur B Weglein, Fernanda V Araújo, Paulo M Carvalho, Robert H Stolt, Kenneth H Matson, Richard T Coates, Dennis Corrigan, Douglas J Foster, Simon A Shaw, and Haiyan Zhang. Inverse scattering series and seismic exploration. *Inverse Problems*, 19(6):R27–R83, oct 2003.

[4] Dirk J. Verschuur and A. J. Berkhout. Estimation of multiple scattering by iterative inversion; part ii, practical aspects and examples. *Geophysics*, 62:1596–1611, 1997.

[5] Tommy Henriksson, N. Joachimowicz, Christophe Conessa, and Jean-Charles Bolomey. Quantitative microwave imaging for breast cancer detection using a planar 2.45 ghz system. *Instrumentation and Measurement, IEEE Transactions on*, 59:2691 – 2699, 11 2010.

[6] Andreas Kirsch. *An Introduction to the Mathematical Theory of Inverse Problems*. 2021.

[7] J. Garnier and G. Papanicolaou. *Passive Imaging with Ambient Noise*. Cambridge Monographs on Applied and Computational Mathematic. Cambridge University Press, 2016.

[8] 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, 2001.

[9] Shi Chen, Zhiyan Ding, Qin Li, and Leonardo Zepeda-Núñez. High-frequency limit of the inverse scattering problem: asymptotic convergence from inverse helmholtz to inverse liouville, 2022.

[10] E. T. Whittaker. Xviii.—on the functions which are represented by the expansions of the interpolation-theory. *Proceedings of the Royal Society of Edinburgh*, 35:181–194, 1915.

[11] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[12] Richard Courant, K. Friedrichs, and Hans Lewy. Über die partiellen differenzengleichungen der mathematischen physik. *Mathematische Annalen*, 100:32–74, 1928.

[13] William W. Symes, Huiyi Chen, and Susan E. Minkoff. *Full-waveform inversion by source extension: Why it works*, pages 765–769. 2020.

[14] L. Zepeda-Núñez and L. Demanet. The method of polarized traces for the 2D Helmholtz equation. *J. Comput. Phys.*, 308:347 – 388, 2016.

[15] B. Engquist and L. Ying. Sweeping preconditioner for the Helmholtz equation: moving perfectly matched layers. *Multiscale Model. Sim.*, 9(2):686–710, 2011.

[16] Carlos Borges, Adrianna Gillman, and Leslie Greengard. High resolution inverse scattering in two dimensions using recursive linearization, 2016.

[17] Yunyue Elita Li and Laurent Demanet. Full waveform inversion with extrapolated low frequency data, 2016.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[19] Alex Graves. Generating sequences with recurrent neural networks, 2013.

[20] Abdel-rahman Mohamed, George E. Dahl, and Geoffrey Hinton. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):14–22, 2012.

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

[22] 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. In *Advances in Neural Information Processing Systems*, volume 3. Nature Machine Intelligence, 2021.

[23] 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, 2022.

[24] Y. Khoo and L. Ying. SwitchNet: A neural network model for forward and inverse scattering problems. *SIAM J. Sci. Comput.*, 41(5):A3182–A3201, 2019.

[25] Y Fan and L. Ying. Solving inverse wave scattering with deep learning. *arXiv:1911.13202*, 2019.

[26] Yuyao Chen, Lu Lu, George Em Karniadakis, and Luca Dal Negro. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Opt. Express*, 28(8):11618–11633, Apr 2020.

[27] H. K. Aggarwal, M. P. Mani, and M. Jacob. MoDL: Model-based deep learning architecture for inverse problems. *IEEE Transactions on Medical Imaging*, 38(2):394–405, 2019.

[28] Davis Gilton, Greg Ongie, and Rebecca Willett. Neumann networks for inverse problems in imaging. *arXiv preprint arXiv:1901.03707*, 2019.

[29] Xiaojiao Mao, Chunhua Shen, and Yu-Bin Yang. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2802–2810. Curran Associates, Inc., 2016.

[30] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*, pages 234–241. Springer International Publishing, Cham, 2015.

[31] J. Bruna and S Mallat. Invariant scattering convolution networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1872–1886, 2013.

[32] Jong Chul Ye, Yoseob Han, and Eunju Cha. Deep convolutional framelets: A general deep learning framework for inverse problems. *SIAM Journal on Imaging Sciences*, 11(2):991–1048, 2018.

[33] E. Kang, W. Chang, J. Yoo, and J. C. Ye. Deep convolutional framelet denosing for low-dose CT via wavelet residual network. *IEEE Transactions on Medical Imaging*, 37(6):1358–1369, 2018.

[34] Y. Fan, J. Feliu-Fabà, L. Lin, L. Ying, and L. Zepeda-Núñez. A multiscale neural network based on hierarchical nested bases. *Research in the Mathematical Sciences*, 6(2):21, Mar. 2019.

[35] 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*, 2021.

[36] Yifan Peng, Lin Lin, Lexing Ying, and Leonardo Zepeda-Núñez. Efficient long-range convolutions for point clouds, 2020.

[37] Tri Dao, Albert Gu, Matthew Eichhorn, Atri Rudra, and Christopher Ré. Learning fast algorithms for linear transforms using butterfly factorizations. *Proceedings of Machine Learning Research*, 97:1517–1527, 06 2019.

[38] Zhongshu Xu, Yingzhou Li, and Xiuyuan Cheng. Butterfly-Net2: Simplified Butterfly-Net and Fourier transform initialization. In Jianfeng Lu and Rachel Ward, editors, *Proceedings of The First Mathematical and Scientific Machine Learning Conference*, volume 107 of *Proceedings of Machine Learning Research*, pages 431–450, Princeton University, Princeton, NJ, USA, 20–24 Jul. 2020. PMLR.

[39] Y. Li, X. Cheng, and J. Lu. Butterfly-Net: Optimal function representation based on convolutional neural networks. *arXiv preprint arXiv:1805.07451*, 2018.

[40] Y. Liu, X. Xing, H. Guo, E. Michielssen, and X. S. Ghysels, P. Li. Butterfly factorization via randomized matrix-vector multiplications. *arXiv:2002.03400*, 2020.

[41] Carlos Borges, Adrianna Gillman, and Leslie Greengard. High resolution inverse scattering in two dimensions using recursive linearization, 2016.

[42] R.-E. Plessix. A review of the adjoint-state method for computing the gradient of a functional with geophysical applications. *Geophysical Journal International*, 167(2):495–503, 11 2006.

[43] D. Colton and R. Kress. *Integral Equation Methods in Scattering Theory*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2013.

[44] Yingzhou Li, Haizhao Yang, Eileen R. Martin, Kenneth L. Ho, and Lexing Ying. Butterfly factorization. 13(2):714–732, jan 2015.

[45] Taco Cohen and Max Welling. Group equivariant convolutional networks. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2990–2999, New York, New York, USA, 20–22 Jun 2016. PMLR.

[46] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, 2021.

[47] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[48] Diederik Kingma and Jimmy Ba. Adam: a method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, May 2015.

[49] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[50] J.-P. Bérenger. A perfectly matched layer for the absorption of electromagnetic waves. *J. Comput. Phys.*, 114(2):185–200, 1994.

[51] L. A. Shepp and B. F. Logan. The fourier reconstruction of a head section. *IEEE Transactions on Nuclear Science*, 21(3):21–43, 1974.

[52] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

A Wide-band Butterfly Network

Wide-band Butterfly Network utilizes the butterfly factorization and Cooley-Tukey FFT algorithm to reduce the number of trainable weights so that it matches the inherent complexity of the problem. The model does not exploit the rotational equivariance property of the problem, and it directly approximates the back-scattering operator by a matrix in the discrete setting.

As we discussed in Section 2.5, the structure of the butterfly factorization is:

$$\mathsf{K} \approx \mathsf{U}^L \mathsf{G}^{L-1} \cdots \mathsf{G}^{L/2} \mathsf{M}^{L/2} (\mathsf{H}^{L/2})^* \cdots (\mathsf{H}^{L-1})^* (\mathsf{V}^L)^* \,, \tag{42}$$

where L is the level of the butterfly factorization. When the matrix is applied on the left to the data, the process can be intuitively interpreted as: V^L extracts a local representation of the vector, and then each H^l compresses two adjacent local representations. Upon the application of the switch matrix $\mathsf{M}^{L/2}$ that redistributes the representations from the previous step by permuting the vector, each G^l decompress the representation by splitting it into two, which increases the resolution of the representation. Finally, U^L converts the local representations to sampling points. In the process, we notice that the resolution of the representation decreases with each H^l being applied. That is where the idea of the Cooley-Tukey FFT algorithm come into effect. After we compress the data of higher frequency by applying H^l to it, it is natural to merge the obtained representation of lower resolution with the data of lower frequency. The assimilation of the multi-frequency data is done in the process of applying H^l. As in our model, the filtering operator is approximated by a 2D convolutional neural network.

In our experiments, we use the data of the same dimension as was in the original paper [23]. Hence, according to the paper, we choose the number of levels as 4, the leaf size as 5 and the rank as 3. In particular, the level number L determines the number of factors in the butterfly factorization, each of which is approximated by a LocallyConnected2D layer in Tensorflow [47]. The leaf size is chosen so that in any $s \times s$ pixels, the data is not oscillatory. Hence, the data can be further compressed and admits a low-rank representation with the desired rank.

B Fourier Neural Operator

Fourier Neural Operator [21] is a data-driven method that learns operators mapping between infinite dimensional spaces. In the model, the far-field data is first lifted into a higher dimensional channel space by a neural network, which is

usually done by using a shallow fully-connected network. Then, a series of Fourier layers are applied. The Fourier layer is composed of a Fourier transformation, a linear transformation, i.e. multiplication by trainable parameters, to filter out the higher modes, and a inverse Fourier transformation. Finally, the result is projected back to the target dimension.

In our experiments, we follow the training approach of the original paper [21]. The model lifts the data to 32-dimensional channel space by applying a linear transformation. Then, four Fourier layers are applied. In particular, the number of higher modes chosen in each layer is 12. Finally it projects the results from the channel space to the target dimension by applying two linear transformations with an activation function placed between them.

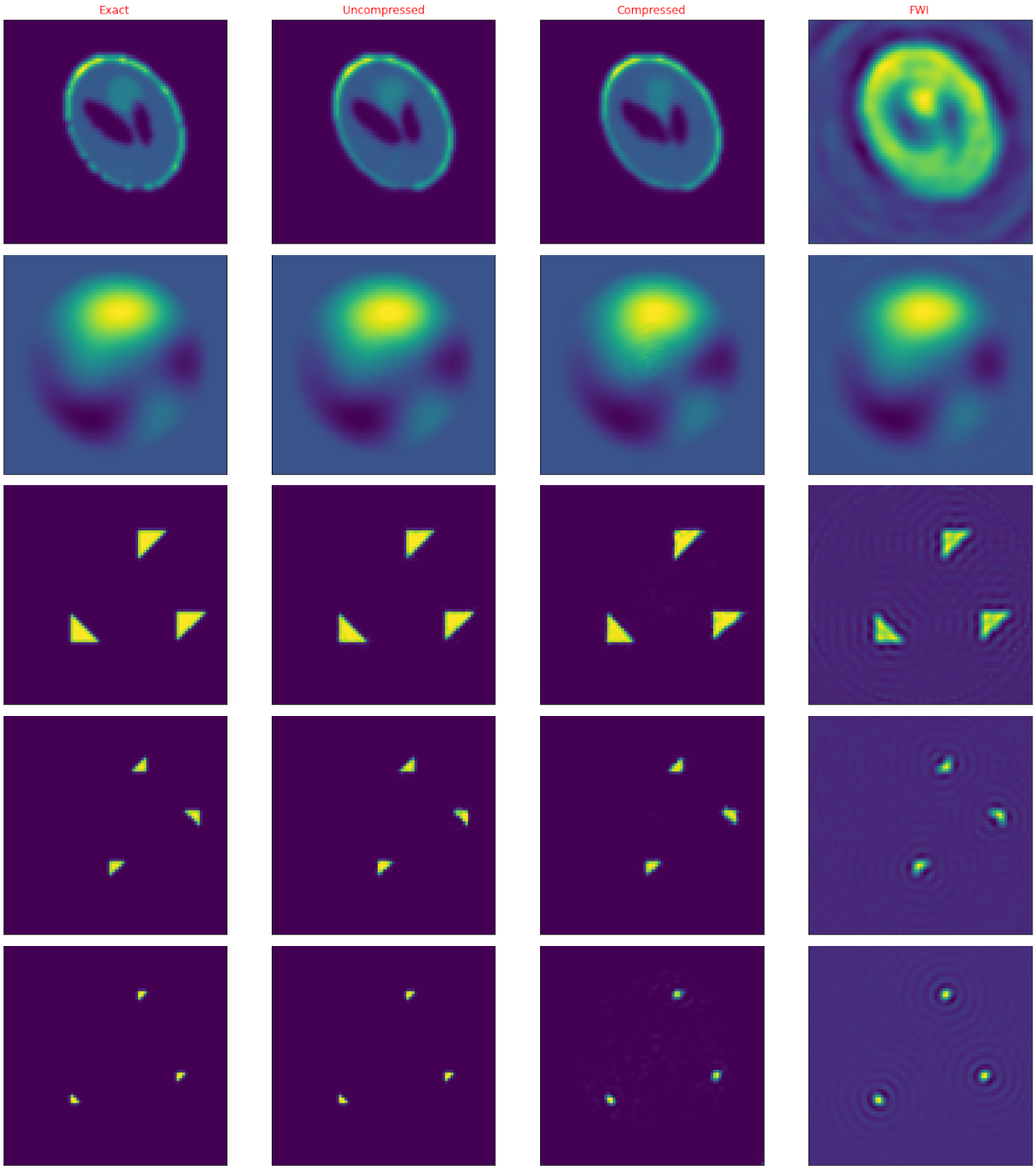

Figure 10: Comparisons of reconstructions of various media by the uncompressed model, the compressed model, and the classical FWI. The media used are the Shepp-Logan phantom (the first row), the random smooth perturbations (the second row), and rotating triangles of different sizes. For the uncompressed and the compressed models, the dimension of data is chosen as $n_{\text{sc}} = 80$, and the number of data points is 2048.