Learned multiphysics inversion with differentiable programming and machine learning

Mathias Louboutin[1*], Ziyi Yin[2*], Rafael Orozco[3], Thomas J. Grady II[3], Ali Siahkoohi[3], Gabrio Rizzuti[4], Philipp A. Witte[5], Olav Møyner[6], Gerard J. Gorman[7], and Felix J. Herrmann[1]

https://doi.org/10.1190/tle42070474.1

Abstract

We present the Seismic Laboratory for Imaging and Modeling/Monitoring open-source software framework for computational geophysics and, more generally, inverse problems involving the wave equation (e.g., seismic and medical ultrasound), regularization with learned priors, and learned neural surrogates for multiphase flow simulations. By integrating multiple layers of abstraction, the software is designed to be both readable and scalable, allowing researchers to easily formulate problems in an abstract fashion while exploiting the latest developments in high-performance computing. The design principles and their benefits are illustrated and demonstrated by means of building a scalable prototype for permeability inversion from time-lapse crosswell seismic data, which, aside from coupling of wave physics and multiphase flow, involves machine learning.

Motivation

Advancements in high-performance computing techniques have led to giant leaps in computational (exploration) geophysics over the past decades. These developments have led, for instance, to the adoption of wave-equation-based inversion technologies such as full-waveform inversion (FWI) and reverse time migration (RTM) that, due to their adherence to wave physics, have resulted in superior imaging in complex geologies. While these techniques rank among the most sophisticated imaging technologies, their implementation relies with few exceptions — most notably iWave++ (Sun and Symes, 2010), Julia Devito Inversion framework (JUDI.jl) of the Seismic Laboratory for Imaging and Modeling (SLIM) (Witte et al., 2019a; Louboutin et al., 2023), and Chevron's COFII (Washbourne et al., 2021) — on monolithic low-level (C/Fortran) implementations. As a consequence, due to their lack of abstraction and modern programming constructs, these low-level implementations are difficult and costly to maintain, especially when performance considerations prevail over best software practices. A noteworthy attempt at modernizing wave-equation inversion frameworks is Deepwave (Richardson, 2018), which implements FWI using PyTorch (Paszke et al., 2019). Despite state-of-the-art examples and applications for 2D inversion, this work is limited by the aforementioned pitfalls as it relies on handwritten low-level C/Cuda code, reducing the flexibility and extensibility to new physics and three-dimensional problems. It also does not integrate machine learning with FWI as advocated in this work. While these implementation design choices lead to performant code for specific problems, such as FWI, they often hinder the implementation of new algorithms, e.g., based on different objective functions or constraints, as well as coupling existing code bases with external software libraries. For instance, combining wave-equation-based inversion with machine learning frameworks or coupling wave physics with multiphase fluid-flow solvers is considered challenging and costly. Thus, our industry runs the risk of losing its ability to innovate, a situation exacerbated by the challenges we face due to the energy transition.

In this work, we present a flexible and agile software framework that aims to resolve these challenges and is designed to be scalable, differentiable, and interoperable. We first introduce the design principles of our software framework, followed by a concrete usage scenario for time-lapse seismic monitoring of geologic carbon storage. This illustrative and didactic example involves the integration of multiple software modules for different types of physics with machine learning techniques such as learned deep priors and neural surrogates. For each module, we explain the choices we made and how these modules are connected through software abstractions and overarching high-level programming language constructs. The advocacy of our proposed framework is demonstrated on a preliminary 2D case study involving the realistic Compass model (Jones et al., 2012). We conclude by discussing remaining challenges and future work directions.

Design principles

To address the shortcomings of current software implementations that impede progress, we have embarked on the development of a performant software framework. For instance, our wave propagators, implemented in Devito (Louboutin et al., 2019; Luporini et al., 2020), are used in production by contractors and oil and gas majors while enabling rapid, low-cost, scalable, and interoperable algorithm development for multiphysics and machine learning problems that run on a variety of chipsets (e.g., ARM, Intel, POWER) and graphics accelerators (e.g., NVIDIA, AMD,

[*]The first and second authors contributed equally to this work.
[1]Georgia Institute of Technology, School of Earth and Atmospheric Sciences, Atlanta, Georgia, USA. E-mail: mlouboutin3@gatech.edu; felix.herrmann@gatech.edu.
[2]Georgia Insitute of Technology, School of Computational Science and Engineering, Atlanta, Georgia, USA. E-mail: ziyi.yin@gatech.edu.
[3]Georgia Institute of Technology, College of Computing, Atlanta, Georgia, USA. E-mail: rorozco@gatech.edu; tgrady@gatech.edu; ali.siahkoohi@gmail.com.
[4]Utrecht University, Utrecht, Netherlands. E-mail: g.rizzuti@uu.nl.
[5]Microsoft Corp., Redmond, Washington, USA. E-mail: pwitte@microsoft.com.
[6]SINTEF, Trondheim, Norway. E-mail: olav.moyner@sintef.no.
[7]Imperial College London, Department of Earth Science and Engineering, London, UK. E-mail: g.gormam@imperial.ac.uk.

Intel). To achieve this, we adopt contemporary software design practices that include high-level abstractions, software design principles, and utilization of modern programming languages such as Python (Rossum and Drake, 2009) and Julia (Bezanson et al., 2017). We also make use of abstractions provided by domain-specific languages (DSLs) such as the Rice Vector Library (Padula et al., 2009) and the Unified Form Language (Alnaes et al., 2015; Rathgeber et al., 2016) and adopt reproducible research practices introduced by the trailblazing open-source initiative Madagascar (Fomel et al., 2013), which made use of version control and an abstraction based on the software construction tool SCons.

To meet the challenges of modern software design in a performance-critical environment, we adhere to three key principles — in addition to the fundamental principle of separation of concerns. First, we adopt mathematical language to inform our abstractions. Mathematics is concise, unambiguous, well understood, and leads to natural abstractions for the

- wave physics, through partial differential equations as put to practice by Devito, which relies on Symbolic Python (SymPy) (Meurer et al., 2017) to define partial differential equations. Given the symbolic expressions, Devito automatically generates highly optimized, possibly domain-decomposed, parallel C code that targets the available hardware with near-optimal performance for 3D acoustic, tilted-transverse-isotropic, or elastic wave equations;
- linear algebra, through matrix-free linear operators, as in JUDI.jl (Witte et al., 2019a; Louboutin et al., 2023) — a high-level linear algebra DSL for wave-equation-based modeling and inversion. These ideas date back to SPOT (van den Berg and Friedlander, 2009) with more recent implementations JOLI.jl (Modzelewski et al., 2023) in Julia and PyLops in Python (Ravasi and Vasconcelos, 2020); and
- optimization, through definition of objective functions, also known as loss functions, that need to be minimized — via SlimOptim.jl (Louboutin et al., 2022c) — subject to mathematical constraints, which can be imposed through SetIntersectionProjection.jl (Peters and Herrmann, 2019; Peters et al., 2022).

Second, we exploit hierarchy within wave-equation-based inversion problems that naturally leads to a separation of concerns. At the highest level, we deal with linear operators, specifically matrix-free Jacobians of wave-based inversion, with JUDI.jl and parallel file input/output with SegyIO.jl (Lensink et al., 2023) on premise, or in the cloud (Azure) via JUDI4Cloud.jl (Louboutin et al., 2022b) and CloudSegyIO.jl (Modzelewski and Louboutin, 2022). At the intermediate and lower level, we make extensive use of Devito (Louboutin et al., 2019; Luporini et al., 2020) — a just-in-time compiler for stencil-based time-domain finite-difference calculations, the development of which SLIM has been involved in over the years.

Third, we build on the principles of differentiable programming as advocated by Innes et al. (2019) and intrusive automatic differentiation introduced by D. Li et al. (2020) to integrate wave physics with machine learning frameworks and multiphase flow. Specifically, we employ automatic differentiation (AD) through the use of the chain rule, including abstractions that allow the user to add derivative rules, as in ChainRules.jl (White et al., 2022, 2023).

During the Federal University of Rio Grande do Norte's inaugural FWI workshop in 2015, we at SLIM started articulating these design principles (Lin and Herrmann, 2015), which over the years cumulated in scalable parallel software frameworks for time-harmonic FWI (Silva and Herrmann, 2019), for time-domain RTM and FWI (Witte et al., 2018, 2019a; Louboutin et al., 2023), and for abstracted FWI (Louboutin et al., 2022a) allowing for connections with machine learning. Aside from developing software for wave-equation-based inversion, we have been involved more recently in the development of scalable machine learning solutions, including the Julia package InvertibleNetworks.jl (Witte et al., 2023), which implements memory-efficient invertible deep neural networks such as (conditional) normalizing flows (NFs) (Rezende and Mohamed, 2015), and scalable distributed Fourier neural operators (FNOs) (Z. Li et al., 2020) in the dfno software package (Grady et al., 2022a, 2022b). All of these will be described in more detail later in this paper.

To illustrate how these design principles can lead to solutions of complex learned coupled inversions, we consider in the ensuing

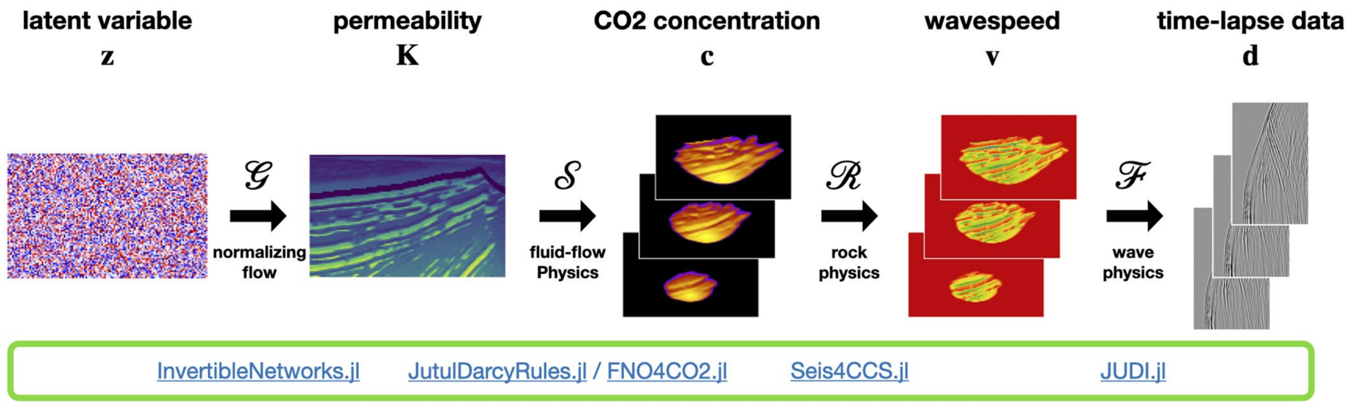

Figure 1. The multiphysics forward model. The permeability, **K**, is generated from Gaussian noise with a pretrained NF, $\mathscr{G}$, followed by two-phase flow simulations through $\mathscr{S}$, rock physics denoted by $\mathscr{R}$, and time-lapse seismic data simulations via wave physics, $\mathscr{F}$.

sections end-to-end inversion of time-lapse seismic data for the spatial permeability distribution (D. Li et al., 2020). As can be seen from Figure 1, this inversion problem is rather complex, and its solution arguably benefits from our three design principles listed earlier. In this formulation, the latent representation for the permeability is taken via a series of nonlinear operations all the way to the time-lapse seismic data. In the remainder of this exposition, we will detail how the different components in this learned inversion problem are implemented so that the coupled inversion can be carried out. The results presented are preliminary, representing a snapshot on how research is conducted according to the design principles.

Learned time-lapse end-to-end permeability inversion

Combating climate change and dealing with the energy transition call for solutions to problems of increasing complexity. Building seismic monitoring systems for geologic CO_2 and/or H_2 storage falls in this category. To demonstrate how math-inspired abstractions can help, we consider inversion of permeability from crosswell time-lapse data (see Figure 2 for experimental setup) involving (1) coupling of wave physics with two-phase (brine/CO_2) flow using Jutul.jl (Møyner et al., 2023) state-of-the-art reservoir modeling software in Julia; (2) learned regularization with NFs with InvertibleNetworks.jl; and (3) learned surrogates for the fluid-flow simulations with FNOs. This type of inversion problem is especially challenging because it involves different types of physics to estimate the past, current, and future saturation and pressure distributions of CO_2 plumes from crosswell data in saline aquifers. In the subsequent sections, we demonstrate how we invert time-lapse data using the separate software packages listed in Figure 1.

Wave-equation-based inversion. Due to its unmatched ability to resolve CO_2 plumes, active-source time-lapse seismic is arguably the preferred imaging modality when monitoring geologic

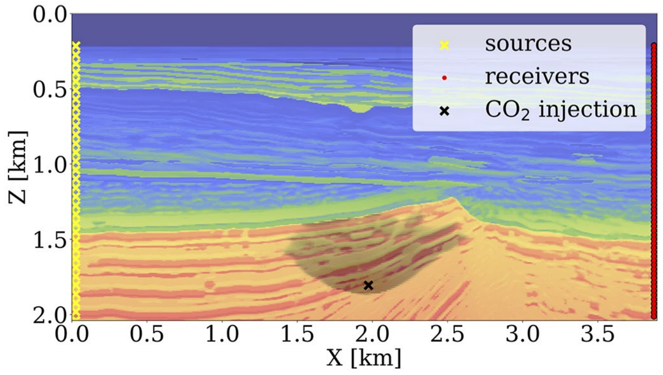

Figure 2. Experimental setup. The black X symbol in the middle of the model indicates the CO_2 injection location. The seismic sources are on the left-hand side of the model (shown as yellow X symbols) and receivers are on the right-hand side of the model (shown as red dots). Overlaid in gray is the compressional wavespeed with simulated CO_2 saturation modeled for 18 years.

storage (Ringrose, 2020). In its simplest form for a single time-lapse vintage, FWI involves minimizing the ℓ_2-norm misfit/loss function between observed and synthetic data — i.e., we have

$$\underset{\mathbf{m}}{\text{minimize}} \ \ \frac{1}{2}\|\mathbf{F}(\mathbf{m})\mathbf{q}-\mathbf{d}\|_2^2 \ \ \text{where} \ \ \mathbf{F}(\mathbf{m}) = \mathbf{P}_r\mathbf{A}(\mathbf{m})^{-1}\mathbf{P}_s^\top. \qquad (1)$$

In this formulation, the symbol $\mathbf{F}(\mathbf{m})$ represents the forward modeling operator (wave physics), parameterized by the squared slowness $\mathbf{m}$. This forward operator acting on the sources consists of the composition of source injection operator $\mathbf{P}_s^\top$, with $^\top$ denoting the transpose operator, solution of the discretized wave equation via $\mathbf{A}(\mathbf{m})^{-1}$, and restriction to the receivers via the linear operator $\mathbf{P}_r$. The vector $\mathbf{q}$ represents the seismic sources, and the vector $\mathbf{d}$ contains single-vintage seismic data collected at the receiver locations. Thanks to our adherence to the math, the corresponding Julia code to invert for the unknown squared slowness $\mathbf{m}$ with JUDI.jl reads

```julia
# Forward modeling to generate seismic data.
Pr = judiProjection(recGeometry)  # setup receiver
Ps = judiProjection(srcGeometry)  # setup sources
Ainv = judiModeling(model)        # setup wave-equation solver
F = Pr * Ainv * Ps'               # forward modeling operator
d = F(m_true) * q                 # generate observed data
# Gradient descent to invert for the unknown squared slowness.
for it = 1:maxiter
    d0 = F(m) * q                 # generate synthetic data
    J = judiJacobian(F(m), q)     # setup the Jacobian operator of F
    g = J' * (d0 - d)             # gradient w.r.t. squared slowness
    m = m - t * g                 # gradient descent with steplength t
end
```

To obtain this concise and abstract formulation for FWI, we utilized hierarchical abstractions for propagators in Devito and linear algebra tools in JUDI.jl, including matrix-free implementations for `F` and its Jacobian `J`. While the preceding stand-alone implementation allows for (sparsity-promoting) seismic (Louboutin and Herrmann, 2017; Louboutin et al., 2018; Herrmann et al., 2019; Witte et al., 2019b; Rizzuti et al., 2020, 2021; Siahkoohi et al., 2020a, 2020b, 2020c; Yang et al., 2020; Yin et al., 2021, 2023) and medical (Yin et al., 2020; Orozco et al., 2021, 2023a, 2023b) inversions, it relies on hand-derived implementations for the adjoint of the Jacobian `J'` and for the derivative of the loss function. Although this approach is viable, relying solely on hand-derived derivatives can become cumbersome when we want to utilize machine learning models or when we need to couple the wave equation to the multiphase flow equations.

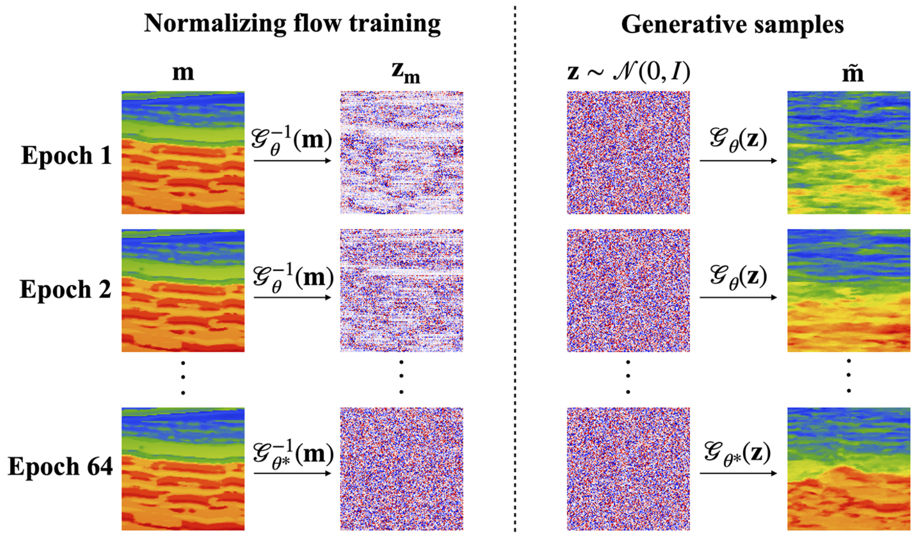
Normalizing flow training
Generative samples
$\mathbf{m}$
$\mathbf{z_m}$
$\mathbf{z} \sim \mathcal{N}(0, I)$
$\tilde{\mathbf{m}}$
Epoch 1
Epoch 2
Epoch 64
$\mathcal{G}_\theta^{-1}(\mathbf{m})$
$\mathcal{G}_{\theta^*}^{-1}(\mathbf{m})$
$\mathcal{G}_\theta(\mathbf{z})$
$\mathcal{G}_{\theta^*}(\mathbf{z})$

Figure 3. Demonstration of Gaussianization of Compass slices during training of an NF. The data used for this didactic example are openly available and this figure is in the InvertibleNetworks.jl repository.

To allow for this situation, we make use of Julia's differentiable programming ecosystem that includes tools to use AD and to add differentiation rules via ChainRules.jl. Using this tool, the AD system can be taught how to differentiate JUDI.jl via the following differentiation rule for the forward propagator:

```julia
# Custom AD rule for wave modeling operator.
function rrule(::typeof(*), F::judiModeling, q)
    y = F * q                      # forward modeling
    # The pullback function for gradient calculations.
    pullback(dy) = NoTangent(), judiJacobian(F, q)' * dy, F' * dy
    return y, pullback
end
```

In this rule, the `pullback` function takes as input the data residual, `dy`, and outputs the gradient of `F * q` with respect to the operator `*` (no gradient), the model parameters, and the source distribution. With this differentiation rule, the above gradient descent algorithm can be implemented as follows:

```julia
# Define the loss function.
loss(m) = .5f0 * norm(F(m) * q - d)^2f0
# Gradient descent to invert for the squared slowness.
for it = 1:maxiter
    g = gradient(loss, m)[1]       # gradient computation via AD
    m = m - t * g                  # gradient descent with steplength t
end
```

Compared to the original implementation, this code only needs `F(m)` and the function `loss(m)`. With the help of the above `rrule`, Julia's AD system[8] is capable of computing the gradients (line 5). Aside from remaining performant — i.e., we still make use of the adjoint-state method to compute the gradients — the advantage of this approach is that it allows for much more flexibility, e.g., in situations where the squared slowness is parameterized in terms of a pretrained neural network or in terms of the output of multiphase flow simulations. In the next section, we show how trained NFs can serve as priors to improve the quality of FWI.

[8]In this case, we used reverse AD provided by Zygote.jl, the AD system provided by Julia machine learning package Flux.jl. Because ChainRules.jl is AD system agnostic, another choice could have been made.

Deep priors and NFs. NFs are generative models that take advantage of invertible deep neural network architectures to learn complex distributions from training examples (Dinh et al., 2016). The term "flow" refers to the transformation of data from a complex distribution to a simple one. The term "normalizing" refers to the standard Gaussian (normal) target distribution that the network learns to map images to. For example, in seismic inversion applications, we are interested in approximating the distribution of earth models to use as priors in downstream tasks. NFs learn to map samples from the target distribution (i.e., earth models) to zero-mean unit standard deviation Gaussian noise using a sequence of trainable nonlinear invertible layers. Once trained, one can resample new Gaussian noise and pass it through the inverse sequence of layers to obtain new generative samples from the target distribution. NFs are an attractive choice for generative models in seismic applications (Zhang and Curtis, 2020, 2021; Siahkoohi and Herrmann, 2021; Siahkoohi et al., 2021, 2022, 2023; Zhao et al., 2021) because they provide fast sampling and allow for memory-efficient training due to their intrinsic invertibility, which eliminates the need to store intermediate activations during backpropagation. Memory efficiency is particularly important for seismic applications due to the 3D volumetric nature of the seismic models. Thus, our methods need to scale well in this regime.

To illustrate the practical use of NFs as priors in seismic inverse problems, we trained an NF on slices from the Compass model (Jones et al., 2012). The training of an NF is laid out in Figure 5 where, for illustrative purposes, we demonstrate a training run on small (64 × 64) slices of the Compass model. Each row shows the normalization (image $\mathbf{m}$ transformed to $\mathbf{Z_m}$ intended to be white zero-mean standard deviation one Gaussian noise) during training and its generative inverse (white noise $\mathbf{z} \sim \mathcal{N}(0,1)$ to image $\tilde{\mathbf{m}}$) during each epoch. From Figure 5, we clearly observe the intended behavior. As the training proceeds, the NFs transform the true model better toward white noise while its inverse progressively generates more realistic looking generative velocity models. To perform a comparison with traditional FWI, we train an NF on full model size slices (512 × 256 grid points). In Figure 5, we compare generative samples from the NF with the slices used to train the model shown in Figure 4. Although there are still irregularities, the model has learned important qualitative aspects of the model that will be useful in inverse problems. To demonstrate this usefulness, we test our prior on an FWI inverse problem. Because our NF prior is trained independently, it is flexible and can be plugged into different inverse problems easily.

Our FWI experiment includes ocean-bottom nodes, Ricker wavelet with no energy below 4 Hz, and additive colored Gaussian noise that has the same bandwidth as the noise-free data. For FWI with our learned prior, we minimize

$$\underset{\mathbf{z}}{\text{minimize}} \quad \frac{1}{2}\|\mathbf{F}(\mathscr{G}_{\theta^*}(\mathbf{z}))\mathbf{q} - \mathbf{d}\|_2^2 + \frac{\lambda}{2}\|\mathbf{z}\|_2^2 \,, \tag{2}$$

where $\mathscr{G}_{\theta^*}$ is a pretrained NF with weights θ^*. After training, the inverse of the NF maps realistic Compass-like earth samples to white noise — i.e., $\mathscr{G}_{\theta^*}^{-1}(\mathbf{m}) = \mathbf{z} \sim \mathcal{N}(0, I)$. Because the NFs are designed to be invertible, the action of the pretrained NF, $\mathscr{G}_{\theta^*}$, on Gaussian noise $\mathbf{z}$ produces realistic samples of earth models (see Figure 5). We use this capability in equation 2 where the unknown model parameters in $\mathbf{m}$ are reparameterized by $\mathscr{G}_{\theta^*}(\mathbf{z})$. The regularization term, $\frac{\lambda}{2}\|\mathbf{z}\|_2^2$, penalizes the latent variable $\mathbf{z}$ with large ℓ_2-norm, where λ balances the misfit and regularization terms. Consequently, this learned regularizer encourages FWI results that are more likely to be realistic earth models (Asim et al., 2020). However, notice that the optimization routine now requires differentiation through both the physical operator (wave physics, $\mathbf{F}$) and the pretrained NF ($\mathscr{G}_{\theta^*}$), and only a true invertible implementation like ours, with minimal memory imprint for both training and inference, can provide scalability.

Due to the JUDI.jl's `rrule` for `F` and InvertibleNetworks.jl's `rrule` for `G`, integration of machine learning with FWI becomes straightforward involving replacement of `m` by `G(z)` on line 6. Minimizing the objective function in equation 2 now translates to

```
# Load the pretrained NF and weights.
G = NetworkGlow(nc, nc_hidden, depth, nscales)
set_params!(G, θ)
# Set up the ADAM optimizer.
opt = ADAM()
# Define the reparameterized loss function including penalty term.
loss(z) = .5f0 * norm(F(G(z)) * q - d)^2f0 + .5f0 * λ * norm(z)^2f0
# ADAM iterations.
for it = 1:maxiter
    g = gradient(loss, z)[1]        # gradient computation with AD
    update!(opt, z, g)              # update z with ADAM
end
# Convert latent variable to squared slowness.
m = G(z)
```

In Figure 6, we compare the results of FWI with our learned prior against unregularized FWI. Because our prior regularizes the solution toward realistic models, we obtain a velocity estimate that is closer to the ground truth. To measure the performance of our method, we use peak signal-to-noise ratio (PS/N) and see an increase from 12.98 dB with traditional FWI to 14.77 dB with the learned prior.

Through this simple example, we demonstrated the ability to easily integrate our state-of-the-art wave-equation propagators with Julia's differentiable programming system. By applying these design principles to other components of the end-to-end inversion, we design a seismic monitoring framework for real-world applications in subsurface reservoirs.

Fluid-flow simulation and permeability inversion. As stated earlier, our goal is to estimate the permeability from time-lapse crosswell monitoring data collected at a CO_2 injection site (cf. Figure 2). Compared to conventional seismic imaging, time-lapse monitoring of geologic storage differs because it aims to image time-lapse changes in the CO_2 plume while obtaining estimates for the reservoir's fluid-flow properties. This involves coupling wave modeling operators to fluid-flow physics to track the CO_2 plumes underground. The fluid-flow physics models the slow process of CO_2 partly replacing brine in the pore space of the reservoir, which involves solving the multiphase flow equations. For this purpose, we need access to reservoir simulation software capable of modeling two-phase

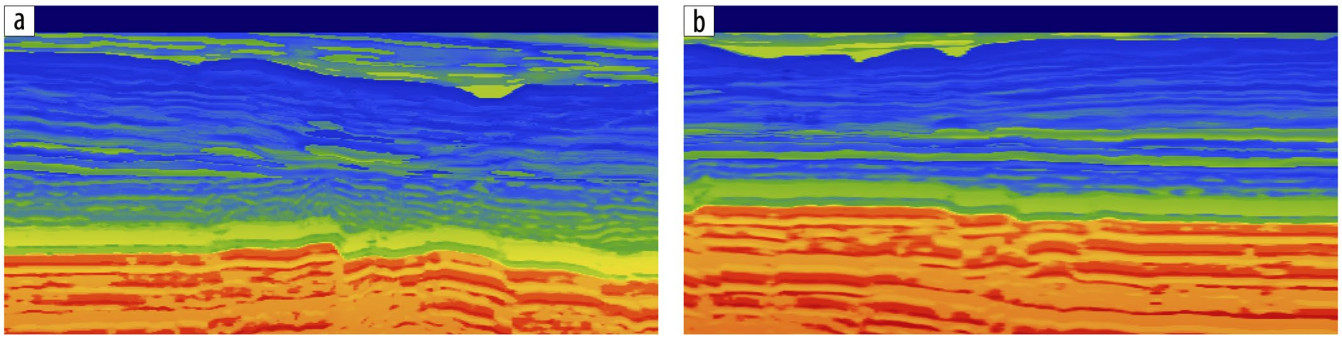

Figure 4. Examples of Compass 2D slices used to train an NF prior.

(brine/CO_2) flow. While several proprietary and open-source reservoir simulators exist, including MRST (Lie and Møyner, 2021), GEOSX (Settgast et al., 2022), and Open Porous Media (Rasmussen et al., 2021), few support differentiation of the simulator's output (CO_2 saturation) with respect to its input (the spatial permeability distribution **K** in Figure 1). We use the recently developed external Julia package JutulDarcy.jl that supports Darcy flow and serves as a front-end to Jutul.jl (Møyner et al., 2023), which provides accurate Jacobians with respect to **K**. Jutul.jl is an implicit solver for finite-volume discretizations that internally uses AD to calculate the Jacobian. It has a performance and feature set comparable to commercial multiphase flow simulators and accounts for realistic effects (e.g., dissolution, interphase mass exchange, compressibility, capillary effects) and residual trapping mechanisms. It also provides accurate sensitivities through an adjoint formulation of the subsurface multiphase flow equations. To integrate the Jacobian of this software package into Julia's differentiable programming system, we wrote the light "wrapper package" JutulDarcyRules.jl (Yin and Louboutin, 2023) that adds an `rrule` for the nonlinear operator $\mathcal{S}(\mathbf{K})$, which maps the permeability distribution, **K**, to the spatially varying CO_2 concentration snapshots, $\mathbf{c} = \{\mathbf{c}^i\}_{i=1}^{n_v}$, over n_v monitoring time steps (cf. Figure 1). Addition of this `rrule` allows these packages to interoperate with other packages in Julia's AD ecosystem. The following shows a basic example where the ADAM algorithm is used to invert for subsurface permeability given the full history of CO_2 concentration snapshots:

```
# Generate CO2 concentration.
c = S(K_true)
# Set up ADAM optimizer.
opt = ADAM()
# Define the loss function.
loss(K) = .5f0 * norm(S(K) - c)^2f0
# ADAM iterations.
for it = 1:maxiter
    g = gradient(loss, K)[1]      # gradient computed with AD
    update!(opt, K, g)            # update K with ADAM
end
```

During each iteration of the preceding loop, Julia's machine learning package Flux.jl (Innes et al., 2018; Innes, 2018b) uses the custom gradient defined by the aforementioned `rrule`, calling the high-performance adjoint code from JutulDarcy.jl. Our adaptable software framework also facilitates effortless substitution of deep learning models in lieu of the numerical fluid-flow simulator. In the next section, we introduce distributed FNOs and discuss how this neural surrogate contributes to our inversion framework.

Fourier neural operator surrogates. While the integration of multiphase flow modeling into the Julia differentiable programming ecosystem opens the way to carry out end-to-end inversions (as explained later), fluid-flow simulations are computationally expensive — a notion compounded by the fact that these simulations have to be done many times during inversion. For this reason, we switch to a data-driven approach where a neural operator is first trained on simulation examples, pairs $\{\mathbf{K}, \mathcal{S}(\mathbf{K})\}$, to learn the mapping from permeability models, **K**, to the corresponding CO_2 snapshots, $\mathbf{c} := \mathcal{S}(\mathbf{K})$. After incurring initial offline training costs, this neural surrogate provides a fast alternative to numerical solvers with acceptable accuracy. FNOs (Z. Li et al., 2020), a neural network architecture based on spectral convolutions that capture the long-range correlations rather than localized spatial convolutions, have been introduced recently as a surrogate for elliptic partial differential equations such as the Darcy or Burgers equation. This spectral architecture has been applied successfully to simulate two-phase flow during geologic CO_2 storage projects (Wen et al., 2022). Independently, Yin et al. (2022) used a trained FNO to replace the fluid-flow simulations as part of end-to-end inversion and showed that AD of Julia's machine learning package can be used to compute gradients with respect to the permeability using Flux.jl's reverse-mode AD system Zygote.jl (Innes, 2018a). After training, the above permeability inversion from concentration snapshots, **c**, is carried out by simply replacing $\mathcal{S}$ by $\mathcal{S}_{\mathbf{w}^*}$ with $\mathbf{w}^*$ being the weights of the pretrained FNO. Thanks to the AD system, the gradient with respect to **K** is computed automatically. Thus, after loading the trained FNO and redefining the operator $\mathcal{S}$, the aforementioned code remains exactly the same. For implementation details on the FNO and its training, we refer to Yin et al. (2022) and Grady et al. (2022b).

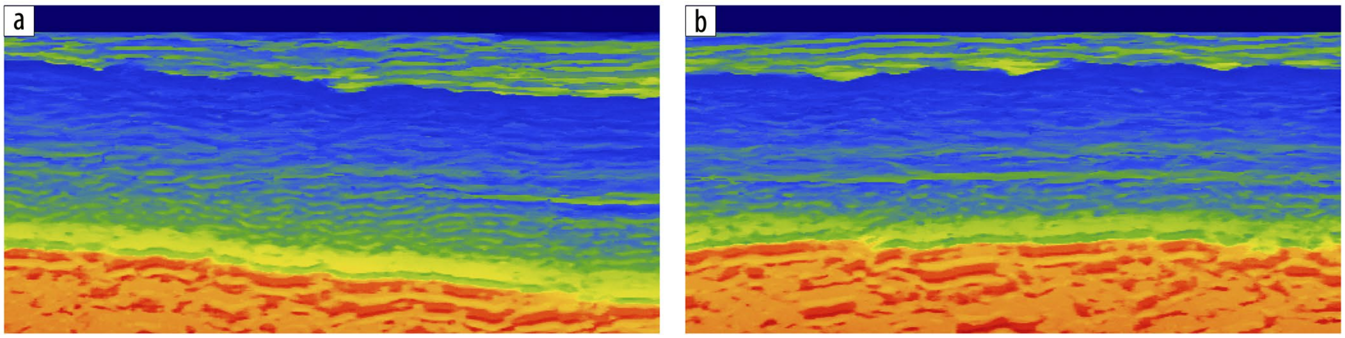
Figure 5. Generative samples from our trained prior. Their similarity to the training samples in Figure 4 suggests that our NF has learned a useful prior.

Putting it all together

As a final step in our end-to-end permeability inversion, we introduce a nonlinear rock-physics model, denoted by $\mathscr{R}$. Based on the patchy saturation model (Avseth et al., 2010), this model nonlinearly maps the time-lapse CO_2 saturations to decreases in the seismic properties (compressional wavespeeds, $\mathbf{v} = \{\mathbf{v}^i\}_{i=1}^{n_v}$) within the reservoir with the Julia code

```
# Patchy saturation function.
# Input: CO2 saturation, velocity, density, porosity.
# Optional: bulk modulus of mineral, brine, CO2; density of CO2, brine.
# Output: velocity, density.
function Patchy(sw, vp, rho, phi;
    bulk_min=36.6f9, bulk_fl1=2.735f9, bulk_fl2=0.125f9,
    ρw=7f2, ρo=1f3) where T
    # Relate vp to vs, set modulus properties.
    vs = vp ./ sqrt(3f0)
    bulk_sat1 = rho .* (vp.^2f0 .- 4f0/3f0 .* vs.^2f0)
    shear_sat1 = rho .* (vs.^2f0)
    # Calculate bulk modulus if filled with 100% CO2.
    patch_temp = bulk_sat1 ./ (bulk_min .- bulk_sat1)
        .- bulk_fl1 ./ phi ./ (bulk_min .- bulk_fl1)
        .+ bulk_fl2 ./ phi ./ (bulk_min .- bulk_fl2)
    bulk_sat2 = bulk_min ./ (1f0 ./ patch_temp .+ 1f0)
    # Calculate new bulk modulus as weighted harmonic average.
    bulk_new = 1f0 / ((1f0 .- sw) ./ (bulk_sat1 .+ 4f0/3f0 * shear_sat1)
    + sw ./ (bulk_sat2 + 4f0/3f0 * shear_sat1)) - 4f0/3f0 * shear_sat1
    # Calculate new density and velocity.
    rho_new = rho + phi .* sw * (ρw - ρo)
    vp_new = sqrt.((bulk_new .+ 4f0/3f0 * shear_sat1) ./ rho_new)
    return vp_new, rho_new
end
```

We map the changes in the wavespeeds to time-lapse seismic data, $\mathbf{d} = \{\mathbf{d}^i\}_{i=1}^{n_v}$, via the blockdiagonal seismic modeling[9] operator $\mathscr{F}(\mathbf{v}) = \mathrm{diag}\big(\{\mathbf{F}^i(\mathbf{v}^i)\mathbf{q}^i\}_{i=1}^{n_v}\big)$. In this formulation, the single-vintage forward operators $\mathbf{F}^i$ and corresponding sources, $\mathbf{q}^i$, are allowed to vary between vintages.

With the fluid-flow (surrogate) solver, $\mathscr{S}$, the rock-physics module, $\mathscr{R}$, and wave-physics module, $\mathscr{F}$, in place, along with regularization via reparametrization using $\mathscr{G}_{\theta^*}$, we are now in a position to formulate the desired end-to-end inversion problem as

$$\underset{\mathbf{z}}{\mathrm{minimize}}\ \frac{1}{2}\|\mathscr{F}\circ\mathscr{R}\circ\mathscr{S}\big(\mathscr{G}_{\theta^*}(\mathbf{z})\big)-\mathbf{d}\|_2^2+\frac{\lambda}{2}\|\mathbf{z}\|_2^2\,, \tag{3}$$

where the inverted permeability can be calculated by $\mathbf{K}^* = \mathscr{G}_{\theta^*}(\mathbf{z}^*)$ with z^* the latent space minimizer of equation 3. As illustrated in Figure 1, we obtain the nonlinear end-to-end map by composing the fluid-flow, rock, and wave physics, according to $\mathscr{F}\circ\mathscr{R}\circ\mathscr{S}$. The corresponding Julia code reads

```
# Set up ADAM optimizer.
opt = ADAM()
# Define the reparameterized loss function including penalty term.
loss(z) = .5f0 * norm(F ∘ R ∘ S(G(z)) - d)^2f0 + .5f0 * λ * norm(z)^2f0
# ADAM iterations.
for it = 1:maxiter
    g = gradient(loss, z)[1]      # gradient computed by AD
    update!(opt, z, g)            # update z by ADAM
end
# Convert latent variable to permeability.
K = G(z)
```

[9]Note, we parameterized this forward modeling in terms of the compressional wavespeed.

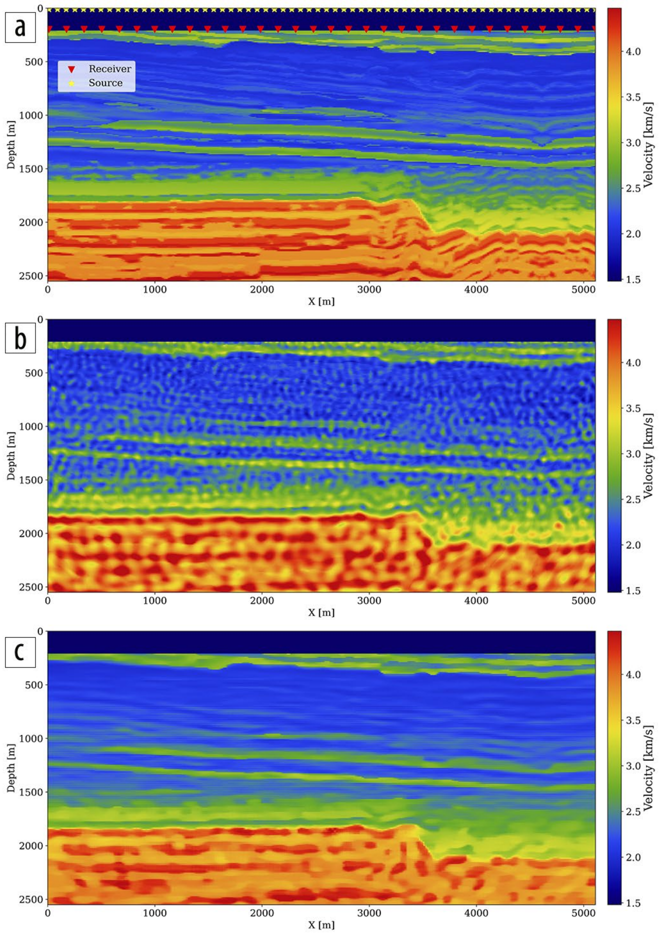

Figure 6. (a) Ground truth. (b) Traditional FWI without prior resulting in 12.98 dB PS/N. (c) Our FWI result with learned prior resulting in 14.77 dB PS/N.

This end-to-end inversion procedure, which utilizes a learned deep prior and a pretrained FNO surrogate, was successfully employed by Yin et al. (2022) on a simple stylistic blocky high-low permeability model. The procedure involves using AD, with `rrule` for the wave and fluid physics, in combination with innate AD capabilities to compute the gradient of the objective in equation 3, which incorporates fluid-flow, rock, and wave physics. To follow, we share early results from applying the proposed end-to-end inversion in a more realistic setting derived from real data (cf. Figure 2).

Preliminary inversion results

While initial results by Yin et al. (2022) were encouraging and showed strong benefits from the learned prior, the permeability model and fluid-flow simulations considered in their study were too simplistic. To evaluate the proposed end-to-end inversion methodology in a more realistic setting, we consider the permeability model plotted in Figure 7a, which we derived from a slice of the Compass model (Jones et al., 2012) shown in Figure 2. To generate realistic CO_2 plumes in this model, we generate immiscible and compressible two-phase flow simulations with JutulDarcy.jl over a period of 18 years with five snapshots plotted at years 10, 15, 16, 17, and 18. These CO_2 snapshots are shown in the first row of Figure 8. Next, given the fluid-flow simulation, we use the patchy saturation model (Avseth et al., 2010) to convert each CO_2 concentration snapshot, $\mathbf{c}^i$, $i = 1 \dots n_v$ to corresponding wavespeed model, v^i, $i = 1 \dots n_v$ with $\mathbf{v} = \mathscr{R}(\mathbf{c})$. We then use JUDI.jl to generate synthetic time-lapse data, $\mathbf{d}^i$, i = 1 … n_v, for each vintage.

During the inversion, the first 15 years of time-lapse data, $\mathbf{d}^i$, i = 1 … 15, from the aforementioned synthetic experiment are

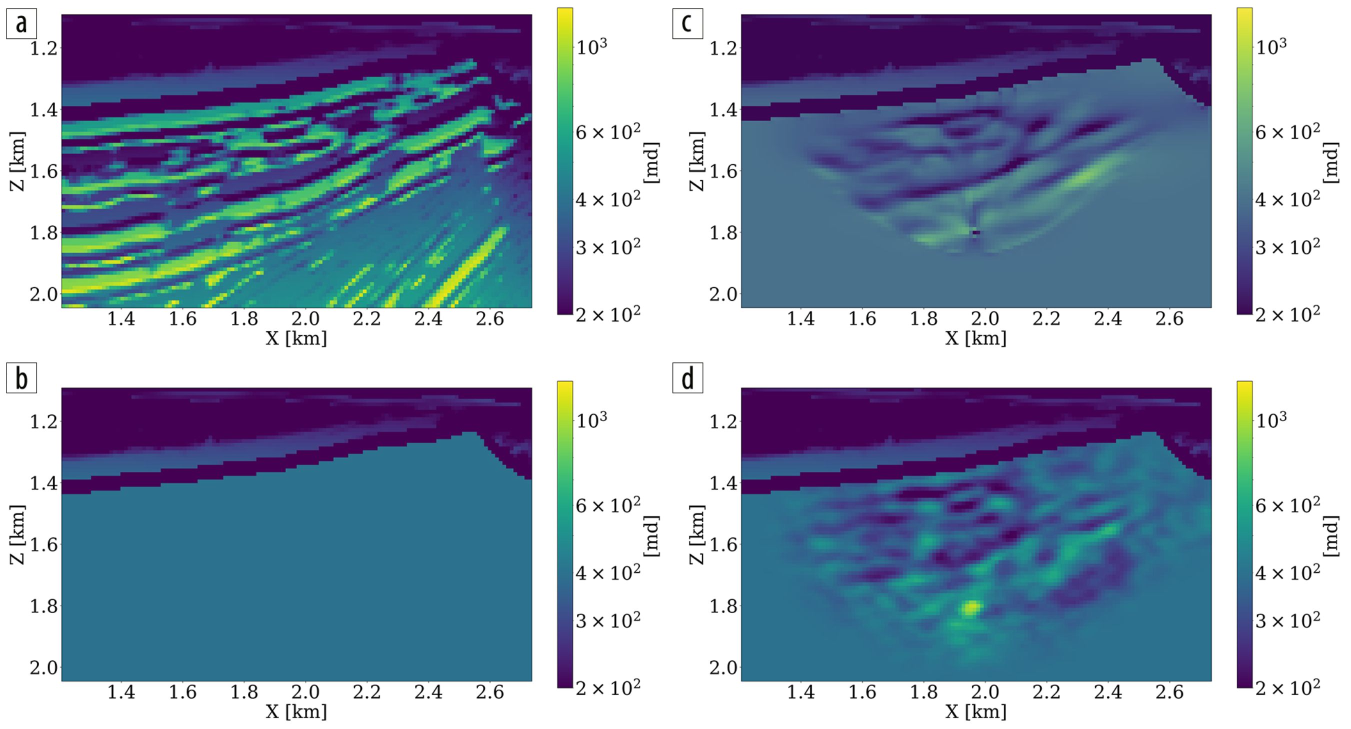

Figure 7. (a) Ground truth permeability. (b) Initial permeability with homogeneous values in the reservoir, with a 7.06 dB S/N. (c) Inverted permeability from physics-based inversion, with a 7.26 dB S/N. (d) Inverted permeability with neural surrogate approximation, with a 7.10 dB S/N.

inverted with permeabilities within the reservoir initialized by a single reasonable value as shown in Figure 7b. Inversion results obtained after 25 passes through the data for the physics-driven two-phase flow solver and its learned neural surrogate approximation are included in Figure 7c and Figure 7d, respectively. Both results were obtained with 200 iterations of the preceding code block. Each time-lapse vintage consist of 960 receivers and 32 shots. To limit the number of wave-equation solves, gradients were calculated for only four randomly selected shots with replacement per iteration. While these results obtained without learned regularization are somewhat preliminary, they lead to the following observations. First, both inversion results for the permeability follow the inverted cone shape of the CO_2. This is to be expected because permeability can only be inverted where CO_2 has flown over the first 15 years. Second, the inverted permeability follows trends of this strongly heterogeneous model. Third, as expected, details and continuity of the results obtained with the two-phase flow solver are better. In part, this can be explained by the fact that there are no guarantees that the model iterations remain with the statistical distribution on which the FNO was trained. Fourth, the implementation of this workflow benefited greatly from the aforementioned software design principles. For instance, the use of abstractions made it trivial to replace physics-driven two-phase flow solvers with their learned counterparts.

Despite being preliminary, the inversion results show that this framework is conducive to producing current CO_2 plume estimates and near-future forecasts. As described by Yin et al. (2022), these capabilities can be achieved through use of the physics simulator or the trained FNO surrogate. The 18-year CO_2 simulations in both inverted permeability models are reasonable when comparing the true plume development plotted in the top row of Figure 8 with plumes simulated from the inverted models plotted in rows three and four of Figure 8. While certain details are missing in the estimates for the past, current, and predicted CO_2 concentrations, the inversion constitutes a considerable improvement compared to plumes generated in the starting model for the permeability plotted in the second row of Figure 8. An early version of the presented workflow can be found in the Julia package Seis4CCS.jl. As the project matures, updated workflows and codes will be pushed to GitHub.

Remaining challenges

We hope we have been able to convince the reader that working with abstractions has its benefits. Due to the math-inspired abstractions, which naturally lead to modularity and separation of concerns, we were able to accelerate the research and development cycle for the end-to-end inversion. As a result, we created a development environment that allowed us to include machine learning techniques. Relatively late in the development cycle, it also gave us the opportunity to swap out the original 2D reservoir simulation code for a much more powerful and fully featured industry-strength 3D code developed by a national lab. What we unfortunately have not yet been able to do is demonstrate our ability to scale this end-to-end inversion to 3D, while both the Devito-based propagators and Jutul.jl's fluid-flow simulations both have been demonstrated on industry-scale problems. Unfortunately, lack of access to large-scale computational resources makes it challenging in an academic environment to validate the proposed methodology on 4D synthetic and field data, even though the computational toolchain presented in this paper is fully differentiable and, in principle, capable of scale-up. Most components have been tested separately and verified on realistic 3D examples (Grady et al., 2022b; Louboutin and Herrmann, 2022, 2023; Møyner and Bruer, 2023) and efforts are underway to remove fundamental memory and other bottlenecks.

Scale-up NFs. Generative models, and NFs included, call for relatively large training sets and large computational resources for training. While efforts have been made to create training sets for more traditional machine learning tasks, no public-domain training set exists that contains realistic 3D examples. A positive is that NFs (Rezende and Mohamed, 2015) have a small memory footprint compared to diffusion models

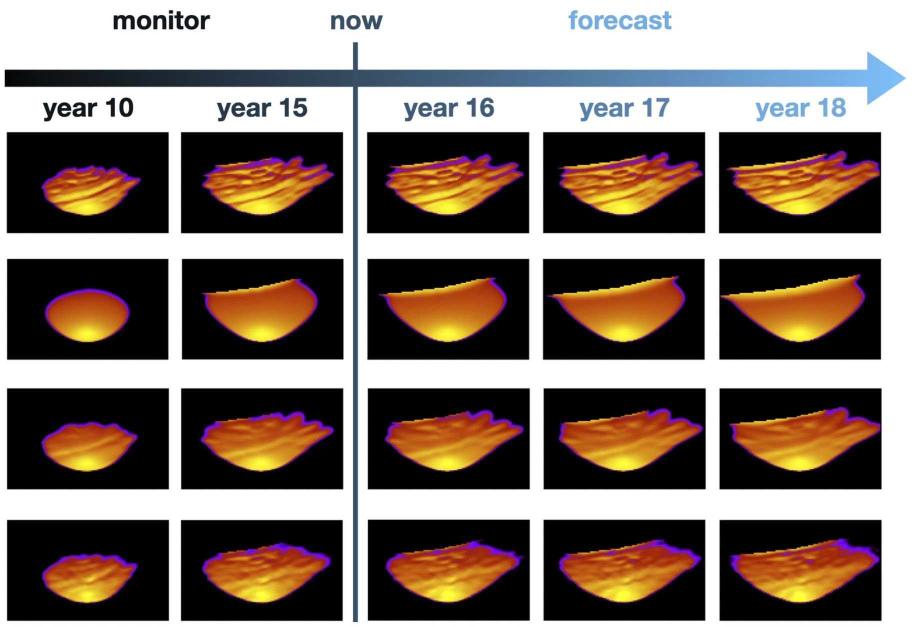

Figure 8. CO_2 plume estimation and prediction. The first two columns are the CO_2 concentration snapshots at year 10 and year 15 of the first 15 years of simulation monitored seismically. The last three columns are forecasted snapshots at years 16, 17, and 18, where no seismic data are available. The first row corresponds to the ground truth CO_2 plume simulated by the unseen ground truth permeability model. The second row contains plume simulations in the starting model, with a 10.99 dB S/N on the first 15 years of CO_2 snapshots and 8.51 dB on the last 3 years. Rows three and four contain estimated and predicted CO_2 plumes for the physics-based and surrogate-based permeability inversion. The S/N values of the first 15 years of the estimated CO_2 plume are 17.72 and 16.17 dB for the physics-based inversion and the surrogate-based inversion, respectively. The S/N values for the CO_2 plume forecasts for the last 3 years are 15.69 and 14.05 dB for the physics-based inversion and the surrogate-based inversion, respectively.

(Song et al., 2020), so training this type of network will be feasible when training sets and compute become available. In our laboratory, we were already able to successfully train and evaluate NFs on 256 × 256 × 64 models. In some cases where geophysicists might not have enough samples for velocity/permeability models, one could use in-house legacy models to train the NFs as a preparation step for inverting the seismic data. We leave the potential investigation to future studies.

Software mentioned in this article (in order of first mention)

JUDI.jl	https://github.com/slimgroup/JUDI.jl
COFII	https://github.com/ChevronETC/Examples
Devito	https://github.com/devitocodes/devito
Julia	https://julialang.org/
SymPy	https://www.sympy.org/en/index.html
SPOT	https://github.com/mpf/spot
JOLI.jl	https://github.com/slimgroup/JOLI.jl
PyLops	https://pylops.readthedocs.io/en/stable/
SlimOptim.jl	https://github.com/slimgroup/SlimOptim.jl
SetIntersectionProjection.jl	https://github.com/slimgroup/SetIntersectionProjection.jl
SegyIO.jl	https://github.com/slimgroup/SegyIO.jl
JUDI4Cloud.jl	https://github.com/slimgroup/JUDI4Cloud.jl
CloudSegyIO.jl	https://github.com/slimgroup/CloudSegyIO.jl
ChainRules.jl	https://github.com/JuliaDiff/ChainRules.jl
InvertibleNetworks.jl	https://github.com/slimgroup/InvertibleNetworks.jl
dfno	https://github.com/slimgroup/dfno
Jutul.jl	https://github.com/sintefmath/Jutul.jl
Zygote.jl	https://github.com/FluxML/Zygote.jl
Flux.jl	https://github.com/FluxML/Flux.jl
JutulDarcy.jl	https://github.com/sintefmath/JutulDarcy.jl
Jutul.jl	https://github.com/sintefmath/Jutul.jl
JutulDarcyRules.jl	https://github.com/slimgroup/JutulDarcyRules.jl
Flux.jl	https://github.com/FluxML/Flux.jl
Seis4CCS.jl	https://github.com/slimgroup/Seis4CCS.jl
ParametricOperators.jl	https://github.com/slimgroup/ParametricOperators.jl

Table 1. Current state of SLIM's software stack. To underline collaboration and active participation in other open-source projects, we included the external software packages (denoted by *) as well as how these are integrated into our software framework.

Package	3D	GPU	AD	Parallelism
Devito*	yes	yes	no	Domain decomposition via MPI, multithreading via OpenMP
JUDI.jl	yes	yes	yes	Multithreading via OpenMP, task parallel
JUDI4Cloud.jl	yes	yes	yes	Multithreading via OpenMP, task parallel
InvertibleNetworks.jl	yes	yes	yes	Julia-native multithreading
dfno	yes	yes	yes	Domain decomposition via MPI
Jutul.jl*	yes	soon	yes	Julia-native multithreading
JutulDarcyRules.jl	yes	soon	yes	Julia-native multithreading
Seis4CCS.jl	yes	yes	yes	Julia-native multithreading
ParametricOperators.jl	yes	yes	yes	Domain decomposition via MPI, Julia-native multithreading

Scale-up neural operators. Since the seminal paper by Z. Li et al. (2020), there has been a flurry of publications on the use of FNOs as neural surrogates for expensive multiphase fluid-flow solvers used to simulate CO_2 injection as part of geologic storage projects (Wen et al., 2022, 2023). While there is good reason for this excitement, challenges remain when scaling this technique to realistic 3D problems. In that case, additional measures must be taken. For instance, by nesting FNOs Wen et al. (2023) were able to divide 3D domains into smaller hierarchical subdomains centered around the wells — an approach that is only viable when certain assumptions are met. Because of this nested decomposition, these authors avoid the large memory footprint of 3D FNOs and report a speedup of many orders of magnitude. Given the potential impact of irregular CO_2 flow, e.g., leakage, we try as much as possible to avoid making assumptions on the flow behavior and propose an accurate distributed FNO structure based on a domain decomposition of the network's input and network weights (Grady et al., 2022b). By using DistDL (Hewett and Grady II, 2020), a software package that supports "model parallelism" in machine learning, our dfno software package partitions the input data and network weights across multiple GPUs such that each partition is able to fit in the memory of a single GPU. As reported by Grady et al. (2022b), our work demonstrated validity of dfno on a realistic problem and reasonable training set (permeability/CO_2 concentration pairs) sizes for permeability models derived from the Sleipner benchmark model (Furre et al., 2017). On 16 timesteps and models of size 64 × 118 × 263, we reported from our perspective a more realistic speedup of more than 1300× compared to the simulation time on Open Porous Media (Rasmussen et al., 2021), one of the leading open-source reservoir simulators. These results confirm a similar indepedent approach advocated by Witte et al. (2022). Even though we are working with our industrial partners and Extreme Scale Solutions to further improve these numbers, we are confident that distributed FNOs are able to scale to 3D with a high degree of parallel efficiency.

Toward scalable open-source software. In addition to allowing for reproduction of published results, we are advocates of pushing out scalable open-source software to help with the energy transition and with combating climate change. As observed in other fields, most notably in machine learning, open-source software leads to accelerated rates of innovation, a feature needed in industries faced with major challenges. Despite the exposition on our experiences implementing end-to-end permeability inversion, this work constitutes a snapshot of an ongoing project. However, many of the software components listed in Table 1 are in an advanced stage of development and to a large degree are ready to be tested in 3D and ultimately on field data. For instance, all of our software supports large-scale 3D simulation and AD. In addition, we are in an advanced state of development to support GPU for all codes. For those curious about future developments, we also include the Julia package ParametricOperators.jl, which is designed to allow for high-dimensional parallel tensor manipulations in support of future Julia-native implementations of distributed FNOs.

The work presented in this paper would not have been possible without open-source efforts from other groups, most notably by researchers at the UK's Imperial College London, who spearheaded the development of Devito, and researchers at Norway's SINTEF. By integrating these packages into Julia's agile differentiable programming environment, we believe that we are well on our way to arrive at a software environment that is much more viable than the sum of its parts. We welcome readers to check https://github.com/slimgroup for the latest developments.

Conclusions

In this work, we introduced a software framework for geophysical inverse problems and machine learning that provides a scalable, portable, and interoperable environment for research and development at scale. We showed that through carefully chosen design principles, software with math-inspired abstractions can be created that naturally leads to desired modularity and separation of concerns without sacrificing performance. We achieve this by combining Devito's automatic code generation for wave propagators with Julia's modern highly performant and scalable programming capabilities, including differentiable programming. These features enabled us to quickly implement a prototype, in principle scalable to 3D, for permeability inversion from time-lapse crosswell seismic data. Aside from the use of proper abstractions, our approach to solving this relatively complex multiphysics problem relied extensively on Julia's innate algorithmic differentiation capabilities, supplemented by auxiliary performant derivatives for the wave/fluid-flow physics and for components of the machine learning. Because of these design choices, we developed an agile and relatively easy to maintain compact software stack where low-level code is hidden through a combination of math-inspired abstractions, modern programming practices, and automatic code generation. TLE

Acknowledgments

This research was carried out with the support of the Georgia Research Alliance and industrial partners of the ML4Seismic Center. The authors thank Henryk Modzelewski (University of British Columbia) and Rishi Khan (Extreme Scale Solutions) for constructive discussions. This work was supported in part by the U.S. National Science Foundation grant OAC 2203821 and Department of Energy grant no. DE-SC0021515.

Data and materials availability

Our software framework is organized into registered Julia packages, all of which can be found on the SLIM GitHub page (https://github.com/slimgroup). The software packages described in this paper are all open source and released under the MIT license for use by the community.

Corresponding author: mlouboutin3@gatech.edu

References

Alnaes, M. S., J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells, 2015, The FEniCS project version 1.5: Archive of Numerical Software, **3**, no. 100, https://doi.org/10.11588/ans.2015.100.20553.

Asim, M., M. Daniels, O. Leong, A. Ahmed, and P. Hand, 2020, Invertible generative models for inverse problems: Mitigating representation error and dataset bias: Proceedings of the 37th International Conference on Machine Learning, PMLR, http://proceedings.mlr.press/v119/asim20a.html, accessed 2 June 2023.

Avseth, P., T. Mukerji, and G. Mavko, 2010, Quantitative seismic interpretation: Applying rock physics tools to reduce interpretation risk: Cambridge University Press.

Bezanson, J., A. Edelman, S. Karpinski, and V. B. Shah, 2017, Julia: A fresh approach to numerical computing: SIAM Review, **59**, no. 1, 65–98, https://doi.org/10.1137/141000671.

Dinh, L., J. Sohl-Dickstein, and S. Bengio, 2016, Density estimation using Real NVP: arXiv preprint, https://doi.org/10.48550/arXiv.1605.08803.

Fomel, S., P. Sava, I. Vlad, Y. Liu, and V. Bashkardin, 2013, Madagascar: Open-source software project for multidimensional data analysis and reproducible computational experiments: Journal of Open Research Software, **1**, no. 1, e8, https://doi.org/10.5334/jors.ag.

Furre, A.-K., O. Eiken, H. Alnes, J. N. Vevatne, and A. F. Kiær, 2017, 20 years of monitoring CO_2-injection at Sleipner: Energy Procedia, **114**, 3916–3926, https://doi.org/10.1016/j.egypro.2017.03.1523.

Grady, T., Infinoid, and M. Louboutin, 2022a, slimgroup/dfno: Optimal comm: Zenodo, https://doi.org/10.5281/zenodo.6981516.

Grady II, T. J., R. Khan, M. Louboutin, Z. Yin, P. A. Witte, R. Chandra, R. J. Hewett, and F. J. Herrmann, 2022b, Model-parallel Fourier neural operators as learned surrogates for large-scale parametric PDEs: arXiv preprint, https://doi.org/10.48550/arXiv.2204.01205.

Herrmann, F. J., A. Siahkoohi, and G. Rizzuti, 2019, Learned imaging with constraints and uncertainty quantification: arXiv preprint, https://doi.org/10.48550/arXiv.1909.06473.

Hewett, R. J., and T. J. Grady II, 2020, A linear algebraic approach to model parallelism in deep learning: arXiv preprint, https://doi.org/10.48550/arXiv.2006.03108.

Innes, M., 2018a, Don't unroll adjoint: Differentiating SSA-form programs: arXiv preprint, https://doi.org/10.48550/arXiv.1810.07951.

Innes, M., 2018b, Flux: Elegant machine learning with Julia: Journal of Open Source Software, **3**, no. 25, 602, https://doi.org/10.21105/joss.00602.

Innes, M., A. Edelman, K. Fischer, C. Rackauckas, E. Saba, V. B. Shah, and W. Tebbutt, 2019, A differentiable programming system to bridge machine learning and scientific computing: arXiv preprint, https://doi.org/10.48550/arXiv.1907.07587.

Innes, M., E. Saba, K. Fischer, D. Gandhi, M. C. Rudilosso, N. M. Joy, T. Karmali, A. Pal, and V. Shah, 2018, Fashionable modelling with Flux: arXiv preprint, https://doi.org/10.48550/arXiv.1811.01457.

Jones, C. E., J. A. Edgar, J. I. Selvage, and H. Crook, 2012, Building complex synthetic models to evaluate acquisition geometries and velocity inversion technologies: 74th Conference and Exhibition, EAGE, Extended Abstracts, https://doi.org/10.3997/2214-4609.20148575.

Lensink, K., H. Modzelewski, M. Louboutin, yzhang3198, and Z. Yin, 2023, slimgroup/SegyIO.jl: v0.8.3: Zenodo, https://doi.org/10.5281/zenodo.7502671.

Li, D., K. Xu, J. M. Harris, and E. Darve, 2020, Coupled time-lapse full-waveform inversion for subsurface flow problems using intrusive automatic differentiation: Water Resources Research, **56**, no. 8, e2019WR027032, https://doi.org/10.1029/2019WR027032.

Li, Z., N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, 2020, Fourier neural operator for parametric partial differential equations: arXiv preprint, https://doi.org/10.48550/arXiv.2010.08895.

Lie, K.-A., and O. Møyner, eds., 2021, Advanced modelling with the MATLAB reservoir simulation toolbox: Cambridge University Press, https://doi.org/10.1017/9781009019781.

Lin, T. T. Y., and F. J. Herrmann, 2015, The student-driven HPC environment at SLIM: Presented at the Inaugural Full-Waveform Inversion Workshop, https://slim.gatech.edu/Publications/Public/Conferences/IIPFWI/lin2015IIPFWIsdh/lin2015IIPFWIsdh_pres.pdf, accessed 2 June 2023.

Louboutin, M., and F. J. Herrmann, 2017, Extending the search space of time-domain adjoint-state FWI with randomized implicit time shifts: 79th Conference and Exhibition, EAGE, Extended Abstracts, https://doi.org/10.3997/2214-4609.201700831.

Louboutin, M., and F. J. Herrmann, 2022, Enabling wave-based inversion on GPUs with randomized trace estimation: 83rd Annual conference and Exhibition, EAGE, Extended Abstracts, https://doi.org/10.3997/2214-4609.202210531.

Louboutin, M., and F. J. Herrmann, 2023, Wave-based inversion at scale on GPUs with randomized trace estimation, https://slim.gatech.edu/Publications/Public/Submitted/2023/louboutin2023rte/paper.html, accessed 2 June 2023.

Louboutin, M., M. Lange, F. Luporini, N. Kukreja, P. A. Witte, F. J. Herrmann, P. Velesko, and G. J. Gorman, 2019, Devito (v3.1.0): An embedded domain-specific language for finite differences and geophysical exploration: Geoscientific Model Development, **12**, no. 3, 1165–1187, https://doi.org/10.5194/gmd-12-1165-2019.

Louboutin, M., P. Witte, and F. J. Herrmann, 2018, Effects of wrong adjoints for RTM in TTI media: 88th Annual International Meeting, SEG, Expanded Abstracts, 331–335, https://doi.org/10.1190/segam2018-2996274.1.

Louboutin, M., P. Witte, A. Siahkoohi, G. Rizzuti, Z. Yin, R. Orozco, and F. J. Herrmann, 2022a, Accelerating innovation with software abstractions for scalable computational geophysics: Second International Meeting for Applied Geoscience & Energy, SEG/APPG, Expanded Abstracts, 1482–1486, https://doi.org/10.1190/image2022-3750561.1.

Louboutin, M., P. Witte, Z. Yin, H. Modzelewski, Kerim, C. da Costa, and P. Nogueira, 2023, slimgroup/JUDI.jl: v3.2.3: Zenodo, https://doi.org/10.5281/zenodo.7785440.

Louboutin, M., Z. Yin, and F. J. Herrmann, 2022b, slimgroup/JUDI4Cloud.jl: First public release: Zenodo, https://doi.org/10.5281/zenodo.6386831.

Louboutin, M., Z. Yin, and F. J. Herrmann, 2022c, slimgroup/SlimOptim.jl: v0.2.0: Zenodo, https://doi.org/10.5281/zenodo.7019463.

Luporini, F., M. Louboutin, M. Lange, N. Kukreja, P. Witte, J. Hückelheim, C. Yount, P. H. J. Kelly, F. J. Herrmann, and G. J. Gorman, 2020, Architecture and performance of devito, a system for automated stencil computation: ACM Transactions on Mathematical Software, **46**, no. 1, 6, https://doi.org/10.1145/3374916.

Meurer, A., C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. M. T. Kumar, et al., 2017, SymPy: Symbolic computing in Python: PeerJ Computer Science, **3**, e103, https://doi.org/10.7717/peerj-cs.103.

Modzelewski, H., and M. Louboutin, 2022, slimgroup/CloudSegyIO.jl: v1.0.1: Zenodo, https://doi.org/10.5281/zenodo.7434854.

Modzelewski, H., M. Louboutin, Z. Yin, D. Karrasch, and R. Orozco, 2023, slimgroup/JOLI.jl: v0.8.5: Zenodo, https://doi.org/10.5281/zenodo.7752660.

Møyner, O., and G. Bruer, 2023, sintefmath/JutulDarcy.jl: v0.2.2: Zenodo, https://doi.org/10.5281/zenodo.7775738.

Møyner, O., M. Johnsrud, H. M. Nilsen, X. Raynaud, K. O. Lye, and Z. Yin, 2023, sintefmath/jutul.jl: v0.2.5: Zenodo, https://doi.org/10.5281/zenodo.7775759.

Orozco, R., M. Louboutin, A. Siahkoohi, G. Rizzuti, T. van Leeuwen, and F. J. Herrmann, 2023a, Amortized normalizing flows for transcranial ultrasound with uncertainty quantification, https://openreview.net/forum?id=LoJG-lUIlk, accessed 5 June 2023.

Orozco, R., A. Siahkoohi, G. Rizzuti, T. van Leeuwen, and F. J. Herrmann, 2021, Photoacoustic imaging with conditional priors from normalizing flows, https://openreview.net/forum?id=woi1OTvROO1, accessed 2 June 2023.

Orozco, R., A. Siahkoohi, G. Rizzuti, T. van Leeuwen, and F. J. Herrmann, 2023b, Adjoint operators enable fast and amortized machine learning based Bayesian uncertainty quantification: Proceedings SPIE 12464, Medical Imaging 2023: Image Processing, 124641L, https://doi.org/10.1117/12.2651691.

Padula, A. D., S. D. Scott, and W. W. Symes, 2009, A software framework for abstract expression of coordinate-free linear algebra and optimization algorithms: ACM Transactions on Mathematical Software, **36**, no. 2, 8, https://doi.org/10.1145/1499096.1499097.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, 2019, PyTorch: An imperative style, high-performance deep learning library: Proceedings of the 33rd International Conference on Neural Information Processing Systems, 8026–8037.

Peters, B., and F. J. Herrmann, 2019, Algorithms and software for projections onto intersections of convex and non-convex sets with applications to inverse problems: arXiv preprint, https://doi.org/10.48550/arXiv.1902.09699.

Peters, B., M. Louboutin, and H. Modzelewski, 2022, slimgroup/SetIntersectionProjection.jl: v0.2.4: Zenodo, https://doi.org/10.5281/zenodo.7257913.

Rasmussen, A. F., T. H. Sandve, K. Bao, A. Lauser, J. Hove, B. Skaflestad, R. Klöfkorn, et al., 2021, The Open Porous Media Flow reservoir simulator: Computers & Mathematics with Applications, **81**, 159–185, https://doi.org/10.1016/j.camwa.2020.05.014.

Rathgeber, F., D. A. Ham, L. Mitchell, M. Lange, F. Luporini, A. T. T. Mcrae, G.-T. Bercea, G. R. Markall, and P. H. J. Kelly, 2016, Firedrake: Automating the finite element method by composing abstractions: ACM Transactions on Mathematical Software, **43**, no. 3, Article 24, https://doi.org/10.1145/2998441.

Ravasi, M., and I. Vasconcelos, 2020, PyLops — A linear-operator Python library for scalable algebra and optimization: SoftwareX, **11**, 100361, https://doi.org/10.1016/j.softx.2019.100361.

Rezende, D., and S. Mohamed, 2015, Variational inference with normalizing flows: Proceedings of the 32nd International Conference on Machine Learning, 1530–1538, http://proceedings.mlr.press/v37/rezende15.html, accessed 2 June 2023.

Richardson, A., 2018, Seismic full-waveform inversion using deep learning tools and techniques: arXiv preprint, https://doi.org/10.48550/arXiv.1801.07232.

Ringrose, P., 2020, How to store CO_2 underground: Insights from early-mover CCS projects: Springer, https://doi.org/10.1007/978-3-030-33113-9.

Rizzuti, G., M. Louboutin, R. Wang, and F. J. Herrmann, 2021, A dual formulation of wavefield reconstruction inversion for large-scale seismic inversion: Geophysics, **86**, no. 6, R879–R893, https://doi.org/10.1190/geo2020-0743.1.

Rizzuti, G., A. Siahkoohi, P. A. Witte, and F. J. Herrmann, 2020, Parameterizing uncertainty by deep invertible networks: An application to reservoir characterization: 90th Annual International Meeting, SEG, Expanded Abstracts, 1541–1545, https://doi.org/10.1190/segam2020-3428150.1.

Settgast, R., C. Sherman, B. Corbett, S. Klevtsov, F. Hamon, A. Mazuyer, A. Vargas, et al., 2022, GEOSX/GEOSX: v0.2.1-alpha: Zenodo, https://doi.org/10.5281/zenodo.7151032.

Siahkoohi, A., and F. J. Herrmann, 2021, Learning by example: Fast reliability-aware seismic imaging with normalizing flows: First International Meeting for Applied Geoscience & Energy, SEG/AAPG, Expanded Abstracts, 1580–1585, https://doi.org/10.1190/segam2021-3581836.1.

Siahkoohi, A., G. Rizzuti, and F. Herrmann, 2020a, A deep-learning based Bayesian approach to seismic imaging and uncertainty quantification: 82nd Conference and Exhibition, EAGE, Extended Abstracts, https://doi.org/10.3997/2214-4609.202010770.

Siahkoohi, A., G. Rizzuti, and F. J. Herrmann, 2020b, Uncertainty quantification in imaging and automatic horizon tracking — A Bayesian deep-prior based approach: 90th Annual International Meeting, SEG, Expanded Abstracts, 1636–1640, https://doi.org/10.1190/segam2020-3417560.1.

Siahkoohi, A., G. Rizzuti, and F. J. Herrmann, 2020c, Weak deep priors for seismic imaging: 90th Annual International Meeting, SEG, Expanded Abstracts, 2998–3002, https://doi.org/10.1190/segam2020-3417568.1.

Siahkoohi, A., G. Rizzuti, and F. J. Herrmann, 2022, Deep Bayesian inference for seismic imaging with tasks: Geophysics, **87**, no. 5, S281–S302, https://doi.org/10.1190/geo2021-0666.1.

Siahkoohi, A., G. Rizzuti, M. Louboutin, P. A. Witte, and F. J. Herrmann, 2021, Preconditioned training of normalizing flows for variational inference in inverse problems: arXiv preprint, https://doi.org/10.48550/arXiv.2101.03709.

Siahkoohi, A., G. Rizzuti, R. Orozco, and F. J. Herrmann, 2023, Reliable amortized variational inference with physics-based latent distribution correction: Geophysics, **88**, no. 3, R297–R322, https://doi.org/10.1190/geo2022-0472.1.

Silva, C. D., and F. Herrmann, 2019, A unified 2D/3D large scale software environment for nonlinear inverse problems: ACM Transactions on Mathematical Software, **45**, no. 1, 7, https://doi.org/10.1145/3291042.

Song, Y., J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, 2020, Score-based generative modeling through stochastic differential equations: arXiv preprint, https://doi.org/10.48550/arXiv.2011.13456.

Sun, D., and W. W. Symes, 2010, IWAVE implementation of adjoint state method: Technical Report 10-06, Department of Computational; Applied Mathematics: Rice University, https://pdfs.semanticscholar.org/6c17/cfe41b76f6b745c435891ea6ba6f4e2c2dbf.pdf, accessed 5 June 2023.

van den Berg, E., and M. P. Friedlander, 2009, Spot: A linear-operator toolbox for Matlab: Presented at SCAIM Seminar.

van Rossum, G., and F. L. Drake, 2009, Python 3 reference manual: CreateSpace.

Washbourne, J., S. Kaplan, M. Merino, U. Albertin, A. Sekar, C. Manuel, S. Mishra, M. Chenette, and A. Loddoch 2021, Chevron optimization framework for imaging and inversion (COFII) — An open source and cloud friendly Julia language framework for seismic modeling and inversion: First International Meeting for Applied Geoscience & Energy, SEG/AAPG, Expanded Abstracts, 792–796, https://doi.org/10.1190/segam2021-3594362.1.

Wen, G., Z. Li, K. Azizzadenesheli, A. Anandkumar, and S. M. Benson, 2022, U-FNO — An enhanced Fourier neural operator-based deep-learning model for multiphase flow: Advances in Water Resources, **163**, 104180, https://doi.org/10.1016/j.advwatres.2022.104180.

Wen, G., Z. Li, Q. Long, K. Azizzadenesheli, A. Anandkumar, and S. Benson, 2023, Real-time high-resolution CO_2 geological storage prediction using nested Fourier neural operators: Energy & Environmental Science, **16**, no. 4, 1732–1741, https://doi.org/10.1039/D2EE04204E.

White, F. C., M. Abbott, M. Zgubic, J. Revels, S. Axen, A. Arslan, S. Schaub, et al., 2023, JuliaDiff/ChainRules.jl: v1.47.0: Zenodo, https://doi.org/10.5281/zenodo.7628788.

White, F. C., M. Zgubic, M. Abbott, J. Revels, N. Robinson, A. Arslan, D. Widmann, et al., 2022, JuliaDiff/ChainRulesCore.jl: v1.15.6: Zenodo, https://doi.org/10.5281/zenodo.7107911.

Witte, P., M. Louboutin, K. Lensink, M. Lange, N. Kukreja, F. Luporini, G. Gorman, and F. J. Herrmann, 2018, Full-waveform inversion, part 3: Optimization: The Leading Edge, **37**, no. 2, 142–145, https://doi.org/10.1190/tle37020142.1.

Witte, P., M. Louboutin, R. Orozco, G. Rizzuti, A. Siahkoohi, F. Herrmann, B. Peters, P. Haraldsson, and Z. Yin, 2023, slimgroup/InvertibleNetworks.jl: v2.2.4: Zenodo, https://doi.org/10.5281/zenodo.7693048.

Witte, P. A., R. J. Hewett, K. Saurabh, A. Sojoodi, and R. Chandra, 2022, SciAI4Industry — Solving PDEs for industry-scale problems with deep learning: arXiv preprint, https://doi.org/10.48550/arXiv.2211.12709.

Witte, P. A., M. Louboutin, N. Kukreja, F. Luporini, M. Lange, G. J. Gorman, and F. J. Herrmann, 2019a, A large-scale framework for symbolic implementations of seismic inversion algorithms in Julia: Geophysics, **84**, no. 3, F57–F71, https://doi.org/10.1190/geo2018-0174.1.

Witte, P. A., M. Louboutin, F. Luporini, G. J. Gorman, and F. J. Herrmann, 2019b, Compressive least-squares migration with on-the-fly fourier transforms: Geophysics, **84**, no. 5, R655–R672, https://doi.org/10.1190/geo2018-0490.1.

Yang, M., Z. Fang, P. A. Witte, and F. J. Herrmann, 2020, Time-domain sparsity promoting least-squares reverse time migration with source estimation: Geophysical Prospecting, **68**, no. 9, 2697–2711, https://doi.org/10.1111/1365-2478.13021.

Yin, Z., H. T. Erdinc, A. P. Gahlot, M. Louboutin, and F. J. Herrmann, 2023, Derisking geologic carbon storage from high-resolution time-lapse seismic to explainable leakage detection: The Leading Edge, **42**, no. 1, 69–76, https://doi.org/10.1190/tle42010069.1.

Yin, Z., and M. Louboutin, 2023, slimgroup/JutulDarcyRules.jl: v0.2.4: Zenodo, https://doi.org/10.5281/zenodo.7762154.

Yin, Z., M. Louboutin, and F. J. Herrmann, 2021, Compressive time-lapse seismic monitoring of carbon storage and sequestration with the joint recovery model: First International Meeting for Applied Geoscience & Energy, SEG/AAPG, Expanded Abstracts, 3434–3438, https://doi.org/10.1190/segam2021-3569087.1.

Yin, Z., R. Orozco, P. A. Witte, M. Louboutin, G. Rizzuti, and F. J. Herrmann, 2020, Extended source imaging — A unifying framework for seismic and medical imaging: 90th Annual International Meeting, SEG, Expanded Abstracts, 3502–3506, https://doi.org/10.1190/segam2020-3426999.1.

Yin, Z., A. Siahkoohi, M. Louboutin, and F. J. Herrmann, 2022, Learned coupled inversion for carbon sequestration monitoring and forecasting with Fourier neural operators: Second International Meeting for Applied Geoscience & Energy, SEG/AAPG, Expanded Abstracts, 467–472, https://doi.org/10.1190/image2022-3722848.1.

Zhang, X., and A. Curtis, 2020, Seismic tomography using variational inference methods: Journal of Geophysical Research: Solid Earth, **125**, no. 4, e2019JB018589, https://doi.org/10.1029/2019JB018589.

Zhang, X., and A. Curtis, 2021, Bayesian geophysical inversion using invertible neural networks: Journal of Geophysical Research: Solid Earth, **126**, no. 7, e2021JB022320, https://doi.org/10.1029/2021JB022320.

Zhao, X., A. Curtis, and X. Zhang, 2021, Bayesian seismic tomography using normalizing flows: Geophysical Journal International, **228**, no. 1, 213–239, https://doi.org/10.1093/gji/ggab298.