PAPER

An autoencoder compression approach for accelerating large-scale inverse problems

IOP Publishing Inverse Problems

Inverse Problems **39** (2023) 115009 (26pp) https://doi.org/10.1088/1361-6420/acfbe1

An autoencoder compression approach for accelerating large-scale inverse problems

Jonathan Wittmer[1,*], **Jacob Badger**[1], **Hari Sundar**[2] **and Tan Bui-Thanh**[3]

[1] Oden Institute for Computational Science and Engineering, University of Texas at Austin, Austin, TX 78712, United States of America
[2] Department of Computer Science, University of Utah, Salt Lake City, UT 84112, United States of America
[3] Department of Aerospace Engineering and Engineering Mechanics, Oden Institute for Computational Science and Engineering, University of Texas at Austin, Austin, TX 78712, United States of America

E-mail: jonathan.wittmer@utexas.edu

Received 31 March 2023; revised 9 August 2023
Accepted for publication 17 August 2023
Published 12 October 2023

Abstract

Partial differential equation (PDE)-constrained inverse problems are some of the most challenging and computationally demanding problems in computational science today. Fine meshes required to accurately compute the PDE solution introduce an enormous number of parameters and require large-scale computing resources such as more processors and more memory to solve such systems in a reasonable time. For inverse problems constrained by time-dependent PDEs, the adjoint method often employed to compute gradients and higher order derivatives efficiently requires solving a time-reversed, so-called adjoint PDE that depends on the forward PDE solution at each timestep. This necessitates the storage of a high-dimensional forward solution vector at every timestep. Such a procedure quickly exhausts the available memory resources. Several approaches that trade additional computation for reduced memory footprint have been proposed to mitigate the memory bottleneck, including checkpointing and compression strategies. In this work, we propose a close-to-ideal scalable compression approach using autoencoders to eliminate the need for checkpointing and substantial memory storage, thereby reducing the time-to-solution and memory requirements. We compare our approach with checkpointing and an off-the-shelf compression approach on an earth-scale ill-posed seismic inverse problem. The results verify the expected close-to-ideal speedup for the gradient and Hessian-vector product using the proposed autoencoder compression approach. To highlight the usefulness of the proposed

* Author to whom any correspondence should be addressed.

approach, we combine the autoencoder compression with the data-informed active subspace (DIAS) prior showing how the DIAS method can be affordably extended to large-scale problems without the need for checkpointing and large memory.

Keywords: machine learning, autoencoder, inverse problems, high performance computing

(Some figures may appear in colour only in the online journal)

1. Introduction

Modern supercomputers are an essential tool for today's scientists and engineers, enabling them to simulate larger and more complex systems than ever before. The availability of large-scale computing resources has enabled numerous breakthroughs in scientific knowledge in areas such as quantum computing (Doi *et al* 2019, Liu *et al* 2021c, Mandrà *et al* 2021), computational chemistry (De Jong *et al* 2010, Kowalski *et al* 2021), drug discovery (Ge *et al* 2013, Schmidt and Hildebrandt 2017, Bharadwaj *et al* 2021, Sukumar *et al* 2021), biology (McFarlane and Biktasheva 2008, Bukowski *et al* 2010), and plasma physics (Bhattacharjee and Wells 2021, Fedeli *et al* 2022). While the increased number of floating-point operations per second has enabled larger and more challenging problems to be solved in reasonable amounts of time, applications are increasingly becoming bottlenecked by limited memory of computing systems—especially those applications dealing with *big data* (Imani *et al* 2019, Denis *et al* 2022).

There are at least three common approaches to mitigating this memory bottleneck: develop new algorithms that require less storage (for example, using density functional theory to approximate electron interactions rather than directly solving a many-body problem in quantum mechanics (Bickelhaupt and Baerends 2000)), utilize additional storage devices such as hard drives to increase the amount of data that can be stored, and trade extra computation for reduced storage requirements. Algorithmic improvement is always desirable, but is not always possible. Additionally, solving a closely related problem with computational or storage advantages may cause a loss in accuracy. Serializing data to disk was previously only a viable solution for problems with high compute intensity. The high cost of reading from and writing to disk can be hidden when sufficient computation is performed. However, recent advances in storage technology, such as non-volatile memory (NVM), have made serialization approaches more viable (Peng *et al* 2020). Indeed, NVM powers the large-memory nodes on Frontera, the Texas Advanced Computing Center's largest supercomputer at the time of writing. This allows traditional DRAM to be used as an extra cache level (Wu *et al* 2017, Stanzione *et al* 2020). The last mentioned approach, trading computation for storage, includes checkpointing methods (Griewank and Walther 1997, Wang *et al* 2009, Zhang and Constantinescu 2023) and sparse decomposition (Zhao *et al* 2020). More generally, this means storing the inputs required to regenerate the desired output rather than storing the output directly (Akturk and Karpuzcu 2018). In addition to saving time, it is shown in Akturk and Karpuzcu (2018) that recomputation can have energy-saving benefits over storing and retrieving data.

One type of problem that calls for advanced big data management techniques is full-waveform seismic inversion (FWI). As motivation, we give a high-level description of why advanced data management techniques are required for FWI here, with a detailed description in section 2. FWI is a partial differential equation (PDE) constrained inverse problem with

dependencies on both space and time derivatives. The dependency on time leads to the generation of large amounts of data. With spatio-temporal measurements of the velocity field, the inverse problem is to infer the underlying spatially varying material properties, namely the acoustic wave speed. Typical inverse solution methods reframe the problem as a constrained optimization problem and use gradient-based methods to minimize an objective function. Employing the adjoint approach (Giles and Pierce 2000, Fichtner *et al* 2006) to compute the gradient necessitates solving an adjoint PDE that is solved backward in time and depends on the corresponding forward solution at each timestep. As a result, *the entire solution history of the forward problem is needed in reverse order to solve the adjoint equation*. For large-scale problems with high spatial resolution and many timesteps, storing the entire forward solution history is unfeasible, thus requiring a different approach. As a quick back-of-the-envelope calculation, consider a domain with 1 billion degrees of freedom and 2000 timesteps. Using a 4th order Runge–Kutta (RK) scheme to integrate in time, there are 4 solutions to be stored per timestep. Each solution has 6 fields—3 velocity fields and 3 strain fields (assuming a velocity-strain formulation). This would require 384 TB to store in double precision. Perhaps one of the most commonly used techniques for such a problem is to employ checkpointing. Checkpointing is a technique whereby solutions are stored at only a few timesteps and the rest discarded. The solution can then be recreated at any timestep by solving the forward problem from the nearest previous checkpoint. While reducing memory requirements, checkpointing effectively increases the computation requirements by 1 PDE solve per gradient evaluation.

An alternative approach employs compression to reduce the memory requirements compared to full storage and computation requirements compared to the checkpointing case. Using compression to mitigate the effects of limited memory or storage is not new. Compression techniques have been used extensively in the audio and video processing community for decades (Blesser 1969, Lewis and Knowles 1990). There are two kinds of compression: lossless compression and lossy compression. Lossless compression is a class of techniques that are able to exactly recover the original input data. The PNG image format is an example of a format that employs lossless compression (Kaur and Choudhary 2016). The checkpointing strategy can be viewed as lossless compression where information is encoded in the checkpoints, and the physics model provides a decompression algorithm. On the other hand, lossy compression allows for errors to be made in reconstructing the input data. The JPEG protocol is an example of lossy compression in the image processing domain (Usevitch 2001).

Various compression techniques have also been proposed to aid in scientific computation. Specific to the seismic inversion community, there are two main avenues of compression: compressing the source/receiver data (Habashy *et al* 2011) and compressing the forward solution to avoid checkpointing. Source/receiver compression is useful in cases where large amounts of data are available, such as when many shots are recorded. The gradient must be evaluated at each shot; each requires the forward and adjoint PDEs solution, implying that the cost scales linearly in the number of shots (Duarte *et al* 2020). The goal of source/receiver compression is to find linear combinations of simultaneous sources and receivers that are (nearly) equivalent to the original problem. When the number of linear combinations required to closely replicate the original problem closely is small, the number of forward and adjoint solutions required to compute a single gradient is much smaller, reducing both the memory footprint and the computation time (Habashy *et al* 2011).

Several other works have already explored using compression to mitigate high adjoint-induced memory requirements (Cyr *et al* 2015, Boehm *et al* 2016, Kukreja *et al* 2019, 2022). Those works demonstrate that data can be compressed without negatively affecting the inverse

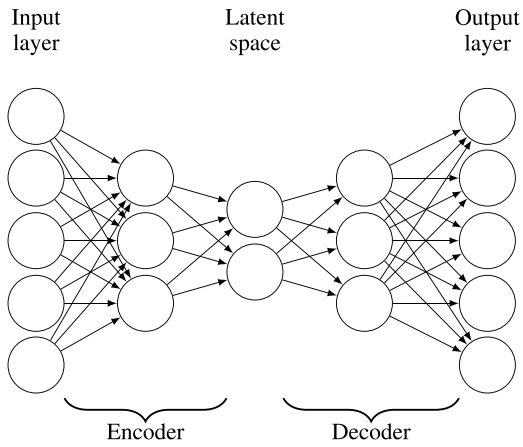


Figure 1. Autoencoder architecture with deep neural networks.

solution and that compression can be faster than forward simulation. Several techniques of compressing in space, as well as spline interpolation techniques for compression in time, were proposed in Boehm *et al* (2016). Results for a variety of off-the-shelf compression algorithms, including extensive results using the ZFP compression algorithm (Lindstrom 2014), were given in Kukreja *et al* (2019). Each paper demonstrates how different compression techniques can solve seismic inverse problems on simple domains with large amounts of data. That is, the inverse problem is not ill-posed, and regularization techniques are not required. While perturbations to the gradient due to compression may minimally affect the inverse solution in the case of large data, it is unclear whether such approaches will be viable in solving ill-posed problems with limited data. Additionally, these approaches have limited compression capabilities, limiting the maximum problem size that can be solved.

Meanwhile, the scientific and technology communities have seen an explosion in the applications of machine learning techniques. Deep learning, in particular, has found fruitful application in enabling progress in speech recognition (Kamath *et al* 2019), computer vision (Esteva *et al* 2021, Bjerge *et al* 2022), and accelerating innovation in scientific computation (Kates-Harbeck *et al* 2019, Reichstein *et al* 2019). One emerging approach is to use an autoencoder architecture to perform data compression (Hinton and Salakhutdinov 2006, Cheng *et al* 2018, Liu *et al* 2021b). An autoencoder is simply a deep neural network that approximates the identity map, as shown in figure 1. The output at some intermediate layer can be extracted as a representation of the input and is referred to as the *latent representation*, which resides in the *latent space* (Dillon *et al* 2021). Autoencoders are typically decomposed into two separate networks: an encoder, which consists of the layers from the input to the latent space and a decoder, which is a network that maps the latent space back to the original dimension. Here, we discuss only the dimension of the inputs and latent space, not the dimension or complexity of the autoencoder. To achieve a compressed representation, the latent space must have a dimension strictly smaller than the input. Compared to traditional dimension reduction methods, autoencoders can be viewed as a nonlinear version of principal component analysis (PCA) or singular value decomposition (Ladjal *et al* 2019, Kneer *et al* 2021, Phillips *et al* 2021). In the simplest case of a single-layer encoder and single-layer decoder with linear activation, the autoencoder learns the same compression as PCA. That is, the single-layer linear autoencoder

projects the input data onto the same lower dimensional subspace that PCA identifies, though the representation is different (Baldi and Hornik 1989, Ladjal *et al* 2019). The actual PCA basis vectors can be obtained from the weights of the trained encoder, showing the two are equivalent (Plaut 2018). The advantage of a deep autoencoder with nonlinear activations is that it can often achieve lower reconstruction error for the same dimension latent space compared to PCA (Liu *et al* 2021a, 2021b), enabling higher feasible compression ratios in an application.

While autoencoders have empirically demonstrated impressive compression ratios and low error rates, there is no theory that we are aware of at this time bounding the error of an autoencoder. However, a practical method of bounding the error of an autoencoder is to compute and store some form of the residual between the desired output ($\boldsymbol{y}_{\text{true}}$) and the actual output ($\boldsymbol{y}_{\text{pred}}$), i.e. $\boldsymbol{r} = \boldsymbol{y}_{\text{true}} - \boldsymbol{y}_{\text{pred}}$. Various techniques of approximately representing this residual vector have been proposed such as compressing $\boldsymbol{r}$ with an off-the-shelf algorithm when $\|\boldsymbol{r}\| > \text{tol}$ for some tolerance, tol (Liu *et al* 2021a, Lee *et al* 2022) or storing only the elements r_i of $\boldsymbol{r}$ where $|r_i| > \text{tol}$ (Abu Alsheikh *et al* 2014, Liu *et al* 2021b).

We propose a novel autoencoder approach for compressing the forward solution to address the problems of additional expensive PDE solves required by the checkpointing approach and the limited compression capability of off-the-shelf compression techniques. We show empirically that this compression method is faster than the checkpointing method while achieving comparable solution accuracy. In addition to showing that a trained autoencoder has memory and computational advantages over existing methods, we develop an efficient data generation and training procedure based on the Bayesian inversion formulation that enables our approach to maintain its advantages, even when including the cost of training the autoencoder. While variations of autoencoders are often used for generative modeling (Kingma and Welling 2013), and others assign a physical interpretation to the latent space (Goh *et al* 2019), we use the *autoencoding* capabilities of the autoencoder architecture to allow the training process to find the most effective compressed latent representation for seismic input data. We provide a mathematical description of the FWI problem and the inverse problem in section 2. Two autoencoder compression variants are proposed in section 3 along with a detailed explanation of autoencoder architecture, training data selection, and normalization. We show in section 4 that machine learning can be leveraged to accelerate the solution of seismic inverse problems in very high dimensions, even in the ill-posed case with few measurements. Through numerical experiments, we show that this approach is faster than the checkpointing approach and as fast as the state-of-the-art off-the-shelf compression approach, ZFP (Lindstrom 2014), while achieving much higher compression ratios. We then extend the data-informed active subspace (DIAS) regularization method (Nguyen *et al* 2022) to nonlinear problems and show that autoencoder compression can enable the affordable application of the DIAS prior to large-scale inverse problems.

2. Full-waveform inversion

A simplified model of the earth's seismic properties considers the acoustic wave equation. In velocity-strain form, this gives rise to a system of first-order PDEs. We choose the velocity-strain form because of its close relation to the acoustic–elastic wave equation (Wilcox *et al* 2015). This enables simultaneous derivation of gradients and Hessians for both acoustic, elastic, and acoustic-elastic formulations. Additional computational and mathematical advantages can be found in Wilcox *et al* (2015). Following the setup given in Bui-Thanh *et al* (2013), the acoustic wave equation is given by,

$$\rho \frac{\partial \boldsymbol{v}}{\partial t} - \nabla \left(\rho c^2 e\right) = \boldsymbol{g}, \tag{1a}$$

$$\frac{\partial e}{\partial t} - \nabla \cdot \boldsymbol{v} = 0, \tag{1b}$$

where $\rho = \rho(\boldsymbol{x})$ is the density, $c = c(\boldsymbol{x})$ is the acoustic wave speed for which we are inverting, $\boldsymbol{g} = \boldsymbol{g}(\boldsymbol{x}, t)$ is a forcing function, $\boldsymbol{v} = \boldsymbol{v}(\boldsymbol{x}, t)$ is the velocity vector, $e = e(\boldsymbol{x}, t)$ is the trace of the strain tensor (dilatation), and $\boldsymbol{x}$ denotes the spatial coordinate in the domain Ω. We specify initial conditions for the velocity and dilatation fields

$$\boldsymbol{v}(\boldsymbol{x}, 0) = \boldsymbol{v}_0(\boldsymbol{x}) \quad \text{and} \quad e(\boldsymbol{x}, 0) = e_0(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega. \tag{2}$$

Lastly, we impose traction-free boundary conditions

$$e(\boldsymbol{x}, t) = 0, \quad \boldsymbol{x} \in \Gamma = \partial\Omega,\ t \in (0, T)$$

with final time T. The three components of the velocity vector field and the three diagonal components of the strain tensor field, used to compute the dilatation, will be collectively referred to as the *state* variables in the rest of this paper. It is the state variables that need to be compressed and decompressed.

In this work, we consider the inverse parameter estimation problem—inferring the wave speed $c(\boldsymbol{x})$ from sparse measurements of the velocity, $\boldsymbol{d}$. Since we consider the setting with limited measurements, the inverse problem is ill-posed, and regularization is required. The inverse problem can be formulated as the optimization problem

$$\boldsymbol{u}^* := \arg\min_{\boldsymbol{u}} \frac{1}{2} \left\| \boldsymbol{\mathcal{F}}(\boldsymbol{u}) - \boldsymbol{d} \right\|^2_{\Gamma^{-1}_{\text{noise}}} + \frac{1}{2} \left\| \boldsymbol{u} - \boldsymbol{u}_0 \right\|^2_{\Gamma^{-1}_{\text{prior}}} \tag{3}$$

where $\boldsymbol{u}$ is referred to as the parameter of interest (PoI), $\boldsymbol{\mathcal{F}}$ is the parameter to observable map (PtO map), Γ_{noise} is the noise covariance matrix of the observations $\boldsymbol{d}$, $\boldsymbol{u}_0$ is the initial guess and Γ_{prior} characterizes the regularization. In the Bayesian setting (Kaipio and Somersalo 2006), $\boldsymbol{u}_0$ and Γ_{prior} are interpreted as the prior mean and prior covariance matrix, respectively. While we consider full waveform inversion in the deterministic setting, it is convenient to consider the problem in the Bayesian framework and only work toward estimating the maximum *a posteriori* (MAP) point. The statistical framing of the MAP problem allows us to choose regularization via the selection of a prior distribution rather than ad-hoc and allows for a trivial extension of our proposed compression approach to quantifying uncertainty. In our case, the PoI is the acoustic wave speed mapped to our observations, $\boldsymbol{d}$, by solving the acoustic wave equation and observing the velocity field only at the receivers. Let $\boldsymbol{u} \approx c$ be the discretized acoustic wave speed that we will numerically estimate. Then

$$\boldsymbol{\mathcal{F}} := \boldsymbol{B}\boldsymbol{\mathcal{A}}$$

where $\boldsymbol{\mathcal{A}}$ is the solution of (1) and $\boldsymbol{B}$ is an observation operator, extracting the velocity field at receiver locations.

As in Bui-Thanh *et al* (2013), we choose the prior to be the PDE-based BiLaplacian which encodes smoothness and anisotropy of the wave speed along with our certainty about our initial guess. The BiLaplacian prior term $\frac{1}{2} \left\| \boldsymbol{u} - \boldsymbol{u}_0 \right\|^2_{\Gamma^{-1}_{\text{prior}}}$ is computed by solving the following elliptic PDE:

$$-\alpha\nabla\cdot(\Theta\nabla\boldsymbol{u})+\alpha\boldsymbol{u}=s \quad \text{in } \Omega \tag{4a}$$

$$\alpha(\Theta\nabla\boldsymbol{u})\cdot\boldsymbol{n}=0 \quad \text{on } \partial\Omega \tag{4b}$$

where $\boldsymbol{n}$ is the outward-facing unit normal on the boundary, $\partial\Omega$. $\Gamma_{\text{noise}}^{-1}$ is the square of this differential operator. Properties of this prior and further discussion can be found in Bui-Thanh *et al* (2013). Finally, derivation of the discrete forward, adjoint, and gradient expressions for a discontinuous Galerkin discretization can be found in Wilcox *et al* (2015).

3. Compression methods

There are a variety of ways that the state variables can be compressed. Each state variable is a function of time and 3 space dimensions, resulting in 4 dimensions across which the state can be compressed. Additionally, some relationships between the state variables might be exploited to achieve higher compression ratios. In this work, we propose two compression techniques that are amenable for implementation with autoencoders, treating each state variable independently:

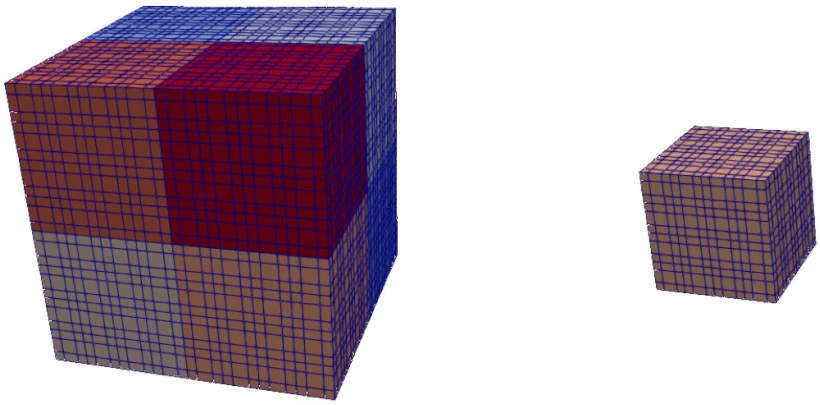
1. Compression in space, allowing the autoencoder to discover the 3D structure of the state across multiple elements.
2. Compression in time on an element-by-element basis.

As a brief justification for pursuing only these two compression setups, let us make a few notes. First, there is potential for scale mismatch between each state variable. Consider again equation (1*b*). The time derivative of the dilatation is related to the divergence of the velocity field. From this, we can see that there could be very large velocities, but no dilatation if the velocity field is divergence-free. This makes it difficult to design a method that can garner any correlation between the *value* of the velocity field at a given time step and the *value* of the strains.

Secondly, we decided against pursuing methods requiring explicit specification of the fields in three-dimensional coordinates. That is, we treat all the nodal coefficients equally, not explicitly taking advantage of the embedding of these nodal coefficients in 3-dimensional space. While it is, in principle, possible to use something like a convolutional neural network with 3-dimensional kernels (Ji *et al* 2012), we found two major difficulties: (1) convolutional neural networks in 3D are rather slow, and (2) the nodes of our DG mesh are not equally spaced. Although equally spaced nodes would enable the simple application of convolutional autoencoders, non-uniformly spaced nodal coordinates such as Gauss–Lobatto–Legendre enable more accurate integration (Quarteroni *et al* 2010) and are a natural choice for node location in a DG scheme. Sections 3.1 and 3.2, will present the implementations of the two proposed autoencoder compression approaches.

3.1. Compression on a uniform mesh

Consider a box domain where $\Omega=[0,1]^3$. Such a domain may arise in reservoir exploration, the example used in Kukreja *et al* (2019). Figure 2 shows the mesh on a box domain. While the nodes are unevenly spaced, they occur in a regular pattern. This allows us to flatten a state variable across multiple elements into a single vector for compression. Care must be taken to partition the domain evenly across MPI ranks and to ensure that each rank stores the nodes in the same spatial orientation, i.e. the elements are stacked the same way in memory on each



(a) (b)

Figure 2. (a) Coarse mesh on box domain showing the non-uniform spacing of the grid. Different colors correspond to different MPI ranks. (b) Mesh of single MPI rank consisting of 8 third-order elements.

rank. The autoencoder then has the freedom to identify the most relevant spatial correlation in order to most accurately compress the data. Note that the order in which the elements are stacked does not matter so long as it is the same across all ranks since a permutation can be implicitly learned. With the same state layout on every rank, a single autoencoder can be trained that compresses the local degrees of freedom on a single rank. Once trained, each rank stores its own copy of the autoencoder, eliminating the need for any communication. The fact that no communication is required is key to the scalability of our proposed approach.

There are a few competing factors that must be balanced in designing an autoencoder for compression. The input vector should be:

1. Large enough so that a high compression ratio can be attained with low reconstruction error.
2. Small enough so that the autoencoder runs quickly.

The second item poses a particular challenge in the present case as the DG solver used to solve the forward acoustic wave equation is highly tuned. Special care must be taken to develop a method faster than checkpointing while maintaining sufficient accuracy. We found empirically that an input dimension of 4096 adequately balanced these two constraints. This corresponds to 8 elements per MPI rank with 3rd-order polynomials, resulting in 64 nodes per element (4^3).

3.2. Compression on non-uniform mesh

While using an autoencoder for spatial compression is intuitive and simple to implement, there are severe limitations to the practical applications of such an approach. First, each MPI rank must have the same number of degrees of freedom arranged in exactly the same manner to avoid padding or communicating ghost elements. This is because autoencoders that employ dense hidden layers have a fixed input size. Unless extreme care is taken, the mesh must be

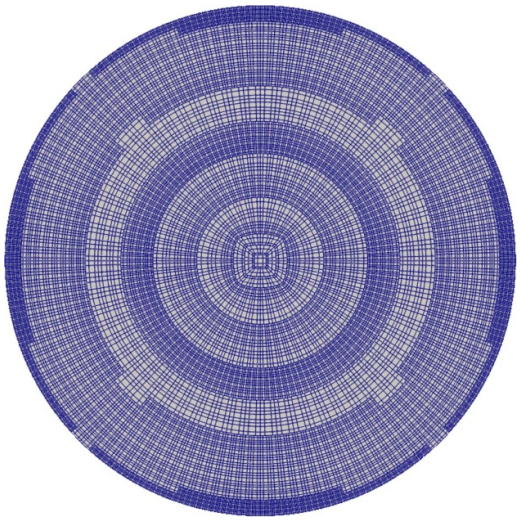


Figure 3. A cross-section of the earth domain with h-adaptive mesh refinement. Notice the non-uniform character of the mesh, with very small elements near the crust required to resolve the seismic waves.

uniformly refined and have no hanging nodes. Such a restriction poses major challenges for problems with a non-rectangular domain or when non-hexahedral elements are desired. To mitigate these limitations, we propose compressing along the time dimension, treating each element as an independent state. This method works seamlessly with h-adaptive mesh refinement (refining the mesh geometry), though p-adaptivity (changing the polynomial order of the element) remains challenging. For this work, we only consider algorithms with h-adaptivity.

To demonstrate the success of this approach, we consider solving a seismic inverse problem on a spherical domain. We employ h-adaptive mesh refinement to ensure the seismic waves can be resolved on each element with at least 3 nodes per wavelength. Figure 3 shows a cross-section of the mesh. Hanging nodes that result from local mesh refinement pose no problem for our proposed compression strategy since each node is considered independently.

The same constraints as discussed in section 3.1 apply in this scenario. Similar to the box domain case, we found that an input dimension to the autoencoder of 4096 worked well since we use a 4th-order RK explicit time stepping scheme (Quarteroni *et al* 2010), a 4096-dimensional input vector results from using 3rd-order polynomials (64 nodes per element) and 16 timesteps, each with 4 RK stages.

3.3. Autoencoder architecture and training details

Any discussion of results obtained using deep learning would be incomplete without discussing the architecture used to obtain the results. While the order of the data differs between space compression and time compression methods, the same autoencoder architecture can be used in either case since the input and output dimensions are the same. The difference between the trained autoencoders for each compression method lies solely in the training data. Table 1 shows the architecture design of both the encoder and the decoder. All layers are ‘dense’ layers, corresponding to an affine transformation followed by a nonlinear activation. We found the

Table 1. Autoencoder architecture and hyperparameters used to train network.

	Encoder	Decoder
Input dimension	4096	64
Output dimension	64	4096
# hidden layers	7	7
Neurons per hidden layer	[512, 256, 256, 256, 128, 64, 64]	[128, 128, 256, 256, 256, 512, 4096]
Activation	ELU	ELU

Table 2. Training hyperparameters used to fit autoencoder weights and biases to data.

Precision	float32
Optimizer	LAMB (You *et al* 2019)
Initial learning rate	10^{-3}
Learning rate decay	$lr = 0.5lr$ applied every 5 epochs
Epochs	20
Batch size	512
# GPUs	2 × 2080Ti

exponential linear unit (ELU) to perform better than the rectified linear unit, hyperbolic tangent (tanh), and sigmoid activations. The ELU activation function is given by

$$\mathrm{ELU}(y) := \begin{cases} e^y - 1 & \text{if } y < 0 \\ y & \text{else.} \end{cases}$$

Table 2 shows the relevant hyperparameters of the training process. Training took approximately 3 h using two Nvidia 2080Ti GPUs. The network hyperparameters reported here were tuned by hand, though Bayesian hyperparameter optimization using the *Keras tuner API* gave similar results. In terms of testing accuracy, we found the network to be insensitive to choice of hyperparameters, with the dimension of the latent space (compression ratio) being most important. It is intuitive that higher compression ratios lead to higher error.

As mentioned in section 3.1, it is difficult to design an autoencoder compression system that both performs well in terms of accuracy and compression ratio while also being fast. Recall that the evaluation of a single dense layer requires the computation of

$$\sigma(\boldsymbol{y}\boldsymbol{W} + \boldsymbol{b}). \tag{5}$$

Considering the first layer of the encoder, $\boldsymbol{W}_1 \in \mathbb{R}^{4096\times512}$, $\boldsymbol{y} \in \mathbb{R}^{4096}$ and $\boldsymbol{b}_1 \in R^{512}$. There are 2097 152 entries in the matrix $\boldsymbol{W}_1$. Compare this to the second layer which has a weight matrix $\boldsymbol{W}_2 \in \mathbb{R}^{512\times256}$ and 131 072 entries. The first layer then has 16 times more entries in the weight matrix than the second layer. Further layers of the encoder have even smaller weight matrices. It is clear that the dominant cost in compressing a vector is the computation of the first matrix-vector multiply, $\boldsymbol{y}\boldsymbol{W}_1$. To mitigate this cost, we develop a hybrid sparse-dense architecture whereby the first layer of the encoder and the last layer of the decoder are trained to be 95% sparse. Sparsity is achieved via the *pruning API* of the Tensorflow model optimization package. In short, the specified layers are made progressively more sparse by permanently setting the smallest magnitude weights to zero. The remaining weights of the network are then

Table 3. Comparison of timings of fully dense architectures vs. hybrid sparse-dense architecture for encoder and decoder for compressing/decompressing 864 vectors on a single core of an Intel Xeon Platinum 8280 CPU.

	Encoder	Decoder
Fully dense	102 ms	140 ms
Hybrid sparse-dense	40 ms	79 ms

Table 4. Comparison of timings of fully dense architectures vs. hybrid sparse-dense architecture for encoder and decoder for compressing/decompressing 864 vectors on 56 cores of 2 Intel Xeon Platinum 8280 CPUs on a single node of the Frontera supercomputer.

	Encoder	Decoder
Fully dense	110 ms	160 ms
Hybrid sparse-dense	120 ms	117 ms

fine-tuned to account for the missing weights. This process is carried out iteratively until the specified sparsity level is achieved. Since the magnitude of the weights are determined through a stochastic gradient descent process, there is no guaranteed structure in the resulting sparsity pattern. Indeed, we observe that the sparsity pattern appears random.

As an example, to show the improvement of the sparse-dense architecture over the fully dense architecture, consider compressing a *batch* of 864 vectors, each with a size of 4096. Since evaluating the neural network output requires nothing more than matrix-vector multiplications, vector additions, and element-wise application of an activation function, all 864 vectors can be processed simultaneously by stacking into rows. Then the input $\boldsymbol{y}$ has shape $(864, 4096)$. Timing results for both the encoder and decoder on a single core of an Intel Xeon Platinum 8280 CPU on Frontera are shown in table 3. We would then expect approximately a 50% speedup with the sparse-dense architecture over the fully dense architecture.

Unfortunately, the application setting is slightly different since all cores are used rather than just a single core. Consider the same test case replicated on each CPU core simultaneously. There are 56 cores across 2 sockets on a single node of Frontera. Since each task executes independently with no communication, we might expect perfect scaling and for the test to complete in the same amount of time. However, there are now more cores contending for data with limited cache and memory bandwidth. Indeed, in the first case, 864 single precision input vectors require only 14.2 MB to store. This easily fits in the 38.5 MB of L3 cache of a single Intel Xeon Platinum 8280 (int). Furthermore, the weights of the fully dense encoder require only 17.8 MB to store. The input data and the entire encoder can fit in the L3 cache! Since the L3 cache is shared among all cores of this CPU, bytes must be streamed from the main memory when all cores are running simultaneously. Table 4 shows the timings for the encoder and decoder with the fully dense and hybrid sparse-dense architectures when running on all cores.

Although there is still a significant advantage in using the hybrid architecture for the decoder, the encoder evaluation proves to be faster using the fully dense architecture than the hybrid architecture when all 56 cores are being utilized. We are unsure why this is the case at the time of writing and further optimization is a topic for future investigation. However, measuring the performance showed that we could achieve faster compression by using a fully dense matrix for the first layer's weights while optimizing the decoder's evaluation time using

a sparse matrix for the final layer. Because of the strict speed requirements, we do not utilize any error bounding methods that require the computation and potential storage of a residual vector (Abu Alsheikh *et al* 2014, Liu *et al* 2021a, 2021b, Lee *et al* 2022).

3.3.1. Training data. One common criticism of deep learning methods is that large training datasets are often required for the DNN to perform well in practice. In the case of compression, other algorithms, such as those implemented in the ZFP package (Lindstrom 2014), do not require training data. Even when deep learning methods may perform well on training data, they may perform poorly in practice when the training dataset is not sufficiently large, a concept referred to as *generalization error*. While this is true, we argue that training an autoencoder is a one-time upfront cost. Further, data generation and training require considerably fewer compute node hours than solving the inverse problem. In this section, we detail the data generation process.

Consider a statistical interpretation where the training data $\boldsymbol{y} \in \mathbb{R}^n$, is drawn from some distribution, $\pi(\boldsymbol{y})$. Suppose a dataset of N samples $\mathcal{D}^N = \{\boldsymbol{y}^1, \ldots, \boldsymbol{y}^N\}$ where $\boldsymbol{y}^i \sim \pi(\boldsymbol{y})$. Let $\Psi_{\mathcal{D}^N}(\boldsymbol{y})$ denote the output of the machine learning algorithm after being trained with dataset $\mathcal{D}^N$. A common metric for evaluating the performance of any machine learning algorithm is the *generalization error*. Given some loss function denoted $\mathcal{L}(\Psi(\boldsymbol{y}), \boldsymbol{y})$, such as mean squared error,

$$\mathcal{L}_{\text{MSE}}(\Psi(\boldsymbol{y}_{\text{true}}), \boldsymbol{y}_{\text{true}}) := \frac{1}{n} \|\boldsymbol{y}_{\text{true}} - \Psi(\boldsymbol{y}_{\text{true}})\|_2^2,$$

the generalization error for an autoencoder is $\mathbb{E}_{\pi(\boldsymbol{y})}[\mathcal{L}(\Psi(\boldsymbol{y}), \boldsymbol{y})]$ (Nadeau and Bengio 1999). The generalization error is the average loss over the entire distribution $\pi(\boldsymbol{y})$.

Generalization error plays an important role in designing a data generation process. There are two ways one can reduce the generalization error:

1. Increase the training dataset size by drawing more samples from $\pi(\boldsymbol{y})$.
2. Choose $\pi(\boldsymbol{y})$ so that it can be well-approximated with fewer samples.

The second method can be interpreted as choosing the most narrow distribution appropriate for the application. We will explore how this option can be used to generate data for seismic inversion. For typical machine learning applications, this may not be possible as we do not know $\pi(\boldsymbol{y})$. The Bayesian inversion setting makes this possible for our application, as we discuss next.

In order to compress seismic data, we need to generate examples similar to the states that will be seen in the application. One way to do this is to solve the acoustic wave equation and save some of the data for training. However, it is still unclear which wave equation solutions should be included in the training dataset. Should we try to train the autoencoder to compress and decompress all possible solutions to the wave equation on a given domain? There are clearly wave equation solutions that are not relevant to our application. Mathematically, the challenge becomes defining the relevant distribution $\pi(\boldsymbol{y})$. The Bayesian framework (Kaipio and Somersalo 2006) helps us to choose a good distribution for generating data. Bayes' formula tells us

$$\pi_{\text{post}}(\boldsymbol{u}|\boldsymbol{d}) \propto \pi_{\text{like}}(\boldsymbol{d}|\boldsymbol{u})\, \pi_{\text{prior}}(\boldsymbol{u}). \tag{6}$$

This can be interpreted as the likelihood, $\pi_{\text{like}}(\boldsymbol{d}|\boldsymbol{u})$, updating the prior density, $\pi_{\text{prior}}(\boldsymbol{u})$, based on observed data, $\boldsymbol{d}$, to produce the posterior density, $\pi_{\text{post}}(\boldsymbol{u}|\boldsymbol{d})$, (Scales and Tenorio 2001).

The observations $\boldsymbol{d}$ are related to the states that will be compressed, denoted $\boldsymbol{y}$ here, through an observation operator, i.e. $\boldsymbol{d} = \mathcal{B}\boldsymbol{y}$. Since the posterior density is the product of the likelihood and the prior, the posterior density is non-zero only in locations where the prior density is non-zero. That is, the posterior lies within the support of the prior. Therefore, we are only interested in whether the autoencoder performs well in compressing states, $\boldsymbol{y}$, that can be generated by samples from the prior. We can generate relevant examples of possible states by solving the acoustic wave equation with the PoI (the wave speed) set to samples from the prior. The autoencoder does not need to perform well on every possible state generated from the wave equation. It only needs to work well on states generated by solving the wave equation with parameters sampled from the prior. By solving the inverse problem using the Bayesian formulation and training the autoencoder to compress states generated by samples from the prior, there is no need to consider out-of-distribution errors in the autoencoder compression. The optimizer, initialized to the prior mean, evolves the solution estimate toward the MAP point, remaining within the support of the prior, which results in states that are in-distribution for the autoencoder.

Algorithm 1. Data generation algorithm for training compression autoencoder

Input: number of samples, n; final time, T

```
for i = 1,...,n do
  c = random draw from prior
  for t = 1,...,T do
    Solve acoustic wave equation for one timestep
    Consolidate states, v, e, into compression data structure
    if Data structure is full then
      Write data structure to file
    end if
  end for
end if
```

With motivation from statistical learning theory and Bayesian inverse problems, we now detail an algorithm for generating training data. The compression data structure is a vector that stores the state variables in the order corresponding to the compression scheme. This setup allows any order to be used and for training data to automatically have the correct ordering—reducing the effort to set up the training pipeline. Algorithm 1 can generate large amounts of data very quickly. Several terabytes can be generated in only a few minutes for the earth-scale problem with an 11.2 million degree of freedom (DoF) mesh. We randomly chose whether to write or discard the data structure to reduce the amount of data written to file. On 32 nodes of Frontera, solving the wave equation and writing to file takes on the order of 1 min. With 10 draws from the prior, keeping only 0.1% of the data, we generated 50 GB of training data in less than 1 h. The total cost of data generation is then $\mathcal{O}(10)$ node hours.

Finally, we discuss the normalization of the training data. The states have scales that vary between 10^{-16} and 10^{5} over the solution of (1). It would be difficult for an autoencoder to accurately compress and decompress states with such widely varying scales, especially in single precision. To reduce the burden on the autoencoder of learning the data scale, we normalize the data between 0 and 1. This enables us to store the scale and offset of the data in double precision while capturing the relative variation of the data in single precision. Consider an arbitrary state vector that will be compressed, denoted $\boldsymbol{y}$, following the notation in (5). Let

$\boldsymbol{y}_{\text{true}}$ be a consolidated state vector that will be compressed (state DoFs consolidated either in space or in time). The input vector to the encoder, $\boldsymbol{y}$, is then given by

$$r = \max\left(\max\left(\boldsymbol{y}_{\text{true}}\right) - \min\left(\boldsymbol{y}_{\text{true}}\right), \beta\right)$$
$$\boldsymbol{y} = \left(\boldsymbol{y}_{\text{true}} - \min\left(\boldsymbol{y}_{\text{true}}\right)\right)/r.$$

In order to avoid division by zero, r is set to be at least some tolerance, β. We choose $\beta = 10^{-7}$.

4. Numerical results

In this section, we present numerical results showing the efficacy of our proposed autoencoder compression approach for both spatial compression on the box domain problem (section 3.1) and temporal compression on a spherical domain (section 3.2). We solve the inverse problem for both box and earth domains using a Newton conjugate gradient method (Epanomeritakis *et al* 2008, Yang 2009). In each conjugate gradient iteration, the action of the Hessian on a vector is required, which entails solving two additional linearized PDEs, referred to as the incremental forward problem and incremental adjoint problem. Expressions for these PDEs in the case of the acoustic wave equation can be found in Bui-Thanh *et al* (2012). The incremental forward problem relies on the solution of the forward problem at each timestep and so requires decompression as well. The incremental adjoint problem requires the solution of both the forward and incremental forward problems, and thus requires another decompression of the forward states. Empirically, we find that compression costs mostly offset the gains of decompression. However, the states only need to be compressed once per Newton iteration and are potentially decompressed many times. The cost of compression is amortized over every evaluation of the adjoint, incremental forward, and incremental adjoint problems within a single Newton iteration. While it is also possible to compress and decompress the incremental forward problem, this paper does not explore it since each incremental forward solution is only re-used once, though such an approach may find use in extremely memory-limited environments.

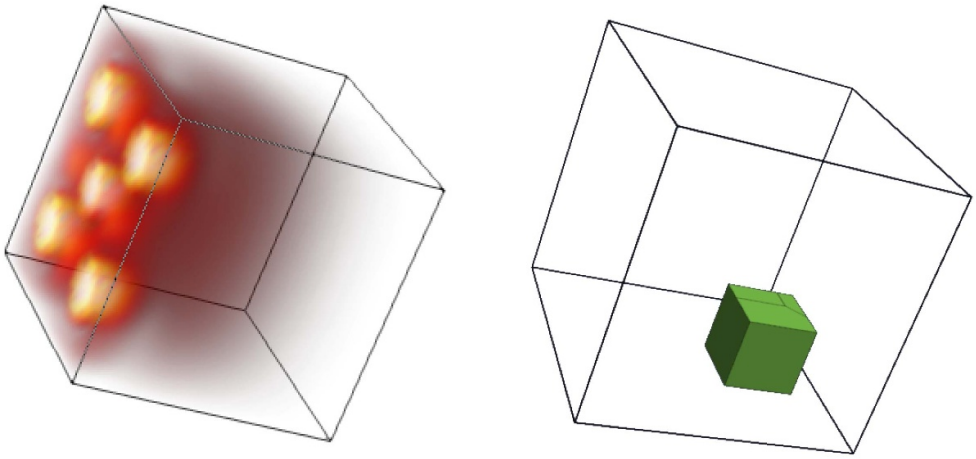
4.1. Results: compression in space

We demonstrate the compression in space approach on a cube domain with all sides of length 1 km. Five sources are located at 1 m depth and 1089 receivers are evenly spaced at the surface, observing the velocity field for 3 s. The sources are Ricker wavelets in time, smoothed by convolving with a narrow Gaussian in space. The source term for a source at $\boldsymbol{x}_0$ is then given by

$$\boldsymbol{g}\left(\boldsymbol{x}, t; \boldsymbol{x}_0\right) = \rho\left(\boldsymbol{x}\right) \frac{1}{\sqrt{2\pi}\sigma_{\boldsymbol{x}}} e^{-\frac{\|x-x_0\|_2^2}{2\sigma_{\boldsymbol{x}}^2}} \left(1 - \frac{\left(t-t_c\right)^2}{\sigma_t^2}\right) e^{-\frac{(t-t_c)^2}{2\sigma_t^2}} \left[0, 0, -1\right]^T.$$

In this case, we consider a wavelet centered at $t_c = 0.6$ s with $\sigma_t = \frac{1}{\pi}$ and $\sigma_{\boldsymbol{x}} = 0.05$.

A snapshot of the velocity magnitude is shown in figure 4(a). Following the setup in section 3.1, we show both speedup and absolute error results compared to the checkpointing solution. While the true synthetic solution is our target, our compression strategy aims to replace checkpointing to managing limited memory, not to change the inverse solution. Therefore, the appropriate comparison is with the inverse solution computed using checkpoints rather than the true solution. Similar to the earth problem, we invert for a variation in



(a) (b)

Figure 4. (a) Snapshot of velocity magnitude at 1.2 s (b) inclusion in background wave speed.

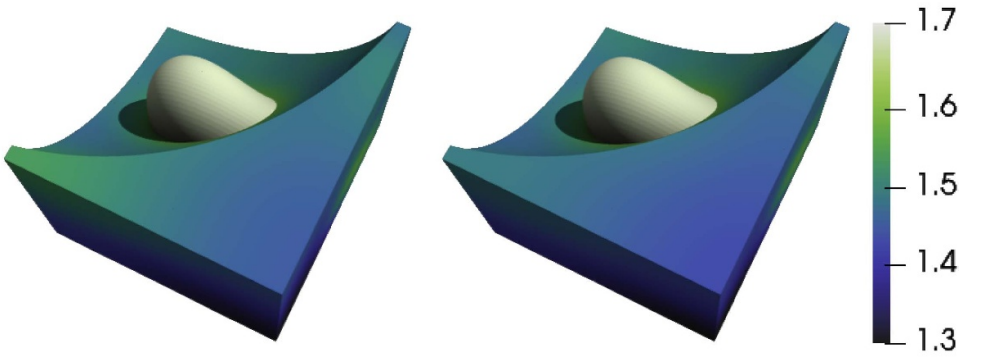
Table 5. Speedup of the autoencoder compression algorithm compared to the checkpointing solution. ∇ represents the gradient, and $\mathcal{H}\boldsymbol{v}$ represents a Hessian-vector product. The speedup approaches the ideal speedup as the number of DoFs increases while the relative error decreases.

Mesh DoFs	Cores	∇ Speedup	$\mathcal{H}\boldsymbol{v}$ Speedup	Relative l_2 Error, %
4096	1	1.06	1.20	1.7
32 768	8	1.19	1.26	0.7
262 144	64	1.17	1.22	0.6
2097 152	512	1.17	1.23	0.7
16 777 216	4096	1.22	1.27	0.4

the background wave speed field. A box-shaped inclusion is introduced into the domain, shown in figure 4(b), and is referred to as the anomaly.

To demonstrate the weak scaling of this method, we show results for several refinement levels, each refinement increasing the total number of degrees of freedom by 8 times. Table 5 shows the speedup compared to the checkpointing solution. While the overall speedup may not be very impressive, it is not the only objective of our proposed compression approach. As we have discussed, the other objectives are reducing memory footprint and eradicating checkpointing. Furthermore, there is a maximum speedup of $1.3\times$ for the gradient since the times to solve the forward problem, the adjoint problem, and evaluate the gradient are approximately equal. That is, the cost of computing the gradient in the ideal case is 3 PDE solves, while the checkpointing case requires 4 PDE solves. Thus, the speedup of 1.22 for the largest mesh in table 5 is close to ideal.

Similarly, the Hessian-vector product computation has a maximum speedup of 1.5 (requiring 4 PDE solves in the ideal case vs. 6 PDE solves with checkpointing), and the speedup of 1.27, without compressing the incremental forward solve, for our compression approach is already an achievement. Table 5 shows that the speedup approaches the ideal one, and the



(a) Checkpoints (b) Autoencoder compression (c) c, km/s

Figure 5. Inverse solutions on box domain using (a) checkpoints and (b) autoencoder compression in space for the problem with 16 777 216 DoF mesh. The solutions have the same color scale with the inclusion rendered as an isosurface at 1.67 km s^{-1}.

relative error decreases as the number of DoFs increases. A cutaway of the inverse solution with the largest mesh for both the checkpointing case and the autoencoder compression case is shown in figure 5. The fact that the anomaly reconstructions are visibly indistinguishable is due to a 0.4% difference.

4.2. Results: compression in time

While it is intuitive that seismic waves exhibit strong spatial redundancy and thus have high compressibility, it is not immediately obvious that temporal compression schemes will yield acceptable results. We test the temporal compression scheme presented in section 3.2 on the earth-scale seismic inverse problem. We build upon the framework established in Bui-Thanh *et al* (2012, 2013) to show that our proposed compression scheme is viable for large-scale seismic inverse problems. The results shown here follow the setup of Case *II* from Bui-Thanh *et al* (2013). There are 3 source locations in the Northern Hemisphere, one at the North Pole and two placed at 90° increments along the equator. Here, we consider sources modeled as Gaussian functions in space centered at the source location, $\boldsymbol{x}_0$, and a Gaussian in time given by

$$\boldsymbol{g}(\boldsymbol{x},t;\boldsymbol{x}_0) = \frac{1}{\sqrt{2\pi}\sigma_t}\frac{1}{(2\pi\sigma_x^2)^{3/2}} e^{-\frac{\|\boldsymbol{x}-\boldsymbol{x}_0\|_2^2}{2\sigma_x^2}} e^{-\frac{(t-t_c)^2}{2\sigma_t^2}}$$

where $\sigma_x = 36$ km, $t_c = 60$ s, and $\sigma_t = 20$ s. We place 130 receivers spaced at 7.5° increments throughout North America, as shown in figure 6, measuring the velocity field. Additive noise with a standard deviation of 0.002 is introduced into the data. The earth is modeled as a sphere with a radius of 6371 km. Both the sources and receivers are buried at 10 km depth.

Coarse meshes cannot resolve the sufficiently high-frequency waves required to image the earth effectively and thus may not be used. We found a maximum frequency resolution of 0.05 Hz sufficient to obtain good inversion results. Thus, the smallest domain size we consider has a mesh with 11.2 million degrees of freedom. This corresponds to a discontinuous Galerkin discretization with 174 481 elements using 3rd-order polynomials. We solve this problem on 64 nodes (3584 CPU cores) of Frontera, running for 10 h. Speedup results compared to the checkpointing case are shown in table 6. Here, we report the average relative l_2 error of the

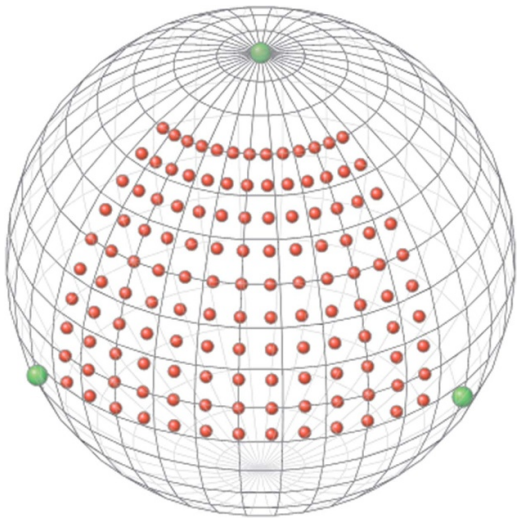


Figure 6. Three sources shown in green at the North Pole and along the equator. 130 receivers shown in red distributed in North America spaced 7.5° apart. Bui-Thanh *et al* (2013), Copyright © 2013 Society for Industrial and Applied Mathematics. Reprinted with permission. All rights reserved.

Table 6. Speedup of the autoencoder compression algorithm compared to the checkpointing solution. ∇ represents the gradient, and $\mathcal{H}\boldsymbol{v}$ represents a Hessian-vector product. Timing results were obtained by running on 64 nodes (3584 CPU cores) of the Frontera supercomputer.

Mesh DoFs	∇ Speedup	$\mathcal{H}\boldsymbol{v}$ Speedup	Relative l_2 error, %
11 166 784	1.05	1.17	0.02

solution obtained using autoencoder compression to the solution obtained using checkpointing. Similar to the box domain, there is greater speedup in the $\mathcal{H}\boldsymbol{v}$ product computation than the gradient computation due to paying the cost of compression during the gradient solve. The difference in speedup between the box problem and the earth problem is likely due to variable per-process problem size induced by the non-uniform spherical mesh and related cache contention issues. As mentioned in section 3, cache effects can greatly impact the performance of the compression algorithm.

To demonstrate the scalability of the autoencoder compression approach, we solve the same problem on a refined mesh with 139 227 136 DoFs. The full state vector (3 velocities and 3 strains) then has 835 362 816 DoFs. This problem was solved on 256 nodes of Frontera (14 336 CPU cores). Table 7 compares the autoencoder compression results to the checkpoint solution. We find a slight reduction in gradient speedup and a slight increase in the speedup of the Hessian-vector product compared to the coarse mesh solution detailed in table 8. One possible reason for the decrease in gradient computation speedup is the larger problem's increased per-process work. The smaller problem with 11 million mesh DoFs has an average of 4681 mesh DoFs per process, while the larger problem has an average of 9711 mesh DoFs per process. We explored the effects of limited cache size on the timing of autoencoder compression in section 3 and found that compression is more sensitive to cache effects than decompression, at least for our implementation. It is, therefore, not surprising that the speedup

Table 7. Inverse results for the earth-scale seismic inverse problem with higher mesh refinement, solved on 14 336 CPU cores. The gains from decompression over resolving from checkpoints for computation of the gradient are completely offset by the cost of compression. However, this cost is paid once, and the Hessian-vector product has a similar speedup as seen on a smaller problem in table 8.

Mesh DoFs	∇ Speedup	$\mathcal{H}\boldsymbol{v}$ Speedup	Relative error %
139 227 136	1.00	1.18	0.06

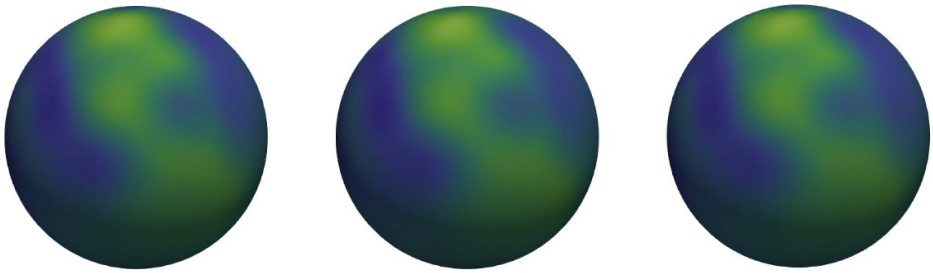
of the gradient computation, which includes the cost of compression, is reduced when each process is responsible for compressing a larger amount of data. While speedup is desired in all computations, there is an order of magnitude more Hessian-vector products than gradient computations required to compute the MAP solution, with $\mathcal{O}(100)\,\mathcal{H}\boldsymbol{v}$ products and $\mathcal{O}(10)$ gradient evaluations. The overall cost is dominated by $\mathcal{H}\boldsymbol{v}$ products, and so the observation that the gradient computation sees no speedup from compression has little impact on the total computation time.

Having shown that autoencoders are a viable compression technique for accelerating seismic inverse problems, we now compare our results to the state-of-the-art off-the-shelf compression technique proposed for seismic inversion in Kukreja *et al* (2019). While other off-the-shelf packages have been proposed, the most comprehensive results in the literature are given for the ZFP compression package (Lindstrom 2014), especially in the seismic inversion literature (Kukreja *et al* 2019, 2022). Therefore, we compare our autoencoder approach's speed, accuracy, and compression ratio to the ZFP compression package. While the ZFP package does not require normalization, we tried both with and without normalization and found that normalizing the states before compressing was necessary to obtain accurate results. Table 8 shows the inversion results using ZFP compression for several required accuracies. The relative errors for all methods are still quite small, though we can see that the autoencoder compression performs similarly to ZFP compression with error tolerance 10^{-3}. This is as expected, since the average pointwise absolute error of the trained autoencoder evaluated on the validation set (withheld from training) was on the order of 10^{-3}. As shown in figure 7, all solutions are visually similar. This replicates the findings of Kukreja *et al* (2022) that the FWI solution is relatively insensitive to compression errors. While autoencoder compression does not show a significant speed advantage over ZFP, there is a substantial difference in compression ratio, and hence, memory footprint. Considering ZFP with $\eta = 10^{-3}$ which has nearly equivalent accuracy, the compression ratio of ZFP is 11.2. On the other hand, the autoencoder compression algorithm achieved a compression ratio of 128—11 times higher than ZFP.

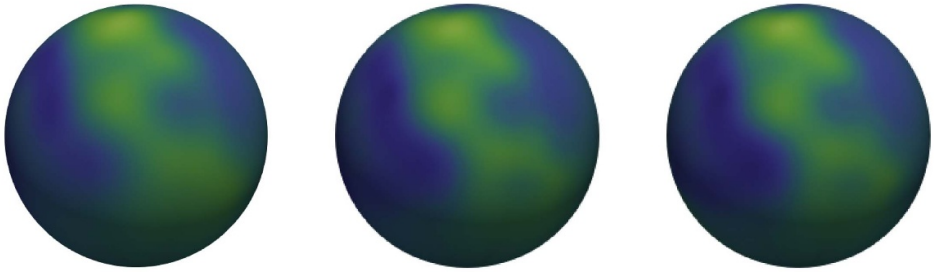
We also compute the error in the gradient for the autoencoder compression technique and compare this to the gradient errors due to using the ZFP package. Figure 8 shows the angle in degrees between the true gradient computed using checkpointing and the gradient computed using compression. As expected, the error increases with the compression tolerance for the ZFP compression algorithm. The gradient error of ZPF becomes larger than the error induced by autoencoder compression between $\eta = 10^{-3}$ and 10^{-2}—corresponding to higher relative error as seen in table 8. The autoencoder gradient has slightly higher error than the ZFP gradient with tolerance 10^{-3}, and performs similarly in terms of l_2 relative error, as seen in table 8.

4.3. DIAS regularization with autoencoder compression

As a capstone result, we show how our proposed autoencoder compression approach can be combined with the DIAS regularization (Nguyen *et al* 2022) approach on the earth-scale



(a) Checkpoint (b) Autoencoder (c) zfp, $\eta = 10^{-5}$



(d) zfp, $\eta = 10^{-4}$ (e) zfp, $\eta = 10^{-3}$ (f) zfp, $\eta = 10^{-2}$

Figure 7. Comparison of inverse solutions obtained using autoencoder and ZFP compression packages with various compression tolerances, η. Results are shown at 130 km depth on the same color scale.

seismic inverse problem. While the DIAS algorithm has the benefit of only applying regularization in the inactive subspace, speed is not one of its strengths. Simply estimating the active subspace for a nonlinear inverse problem requires numerous samples of the gradient—each evaluation of which requires 2 PDE solves. In the case of large-scale seismic inversion, each gradient evaluation can take several minutes on dozens of compute nodes. We propose to combine the DIAS regularization algorithm with autoencoder compression to alleviate the high cost and memory footprint of checkpointing in estimating the active subspace. Since we have already discussed the speedup and memory footprint gained by the autoencoder compression at length above, we will focus on the solution quality using DIAS plus autoencoder compression in the following.

Let us briefly recap the DIAS regularization. We will focus on the DIAS-F (see Nguyen *et al* 2022, section 4) variant, which uses the full data misfit and only applies regularization in the inactive subspace. First, the active subspace is estimated via,

$$\begin{bmatrix} \boldsymbol{W}_1 & \boldsymbol{W}_2 \end{bmatrix} \begin{bmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{W}_1^T \\ \boldsymbol{W}_2^T \end{bmatrix} = \boldsymbol{C} := \int \nabla_{\boldsymbol{u}} f(\boldsymbol{u}) \nabla_{\boldsymbol{u}} f(\boldsymbol{u})^T \pi_{\text{prior}}(\boldsymbol{u}) \, \mathrm{d}\boldsymbol{u}, \tag{7}$$

where $f := \frac{1}{2} \left\| \boldsymbol{\mathcal{F}}(\boldsymbol{u}) - \boldsymbol{d} \right\|^2_{\Gamma^{-1}_{\text{noise}}}$, $\boldsymbol{W}_1$ are the eigenvectors of $\boldsymbol{C}$ corresponding to the active subspace, and $\boldsymbol{W}_2$ are the eigenvectors corresponding to the inactive subspace. As in section 3.3.1,

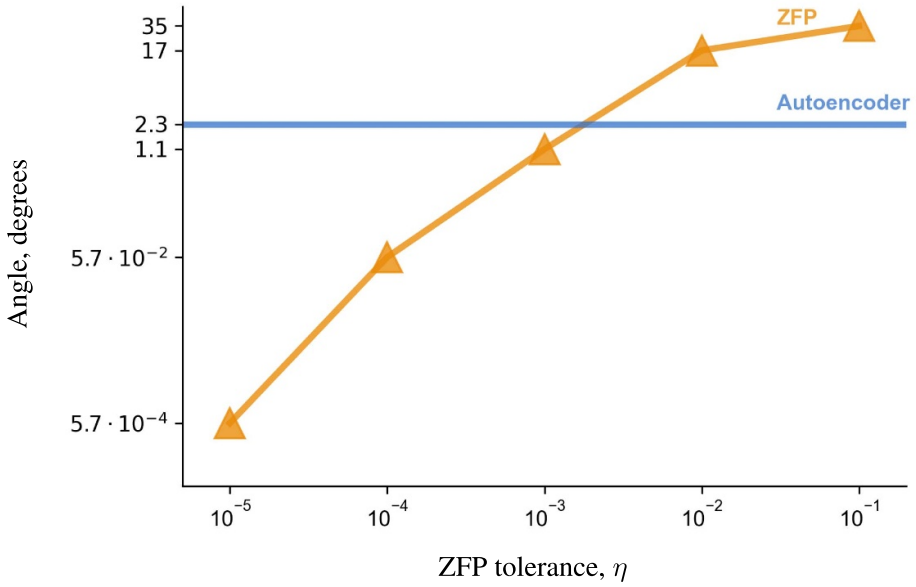


Figure 8. Angle between the true gradient and the gradient computed using compression. Between $\eta = 10^{-3}$ and 10^{-2}, the ZFP gradients become worse than the gradients computed using autoencoder compression.

Table 8. Comparison of ZFP and autoencoder compression for various compression tolerances. There is no significant difference in the speedup of the autoencoder compression vs. the ZFP compression. However, the autoencoder can achieve a much higher compression ratio (and hence, a much smaller memory footprint) while maintaining solution accuracy. Relative l_2 error is computed with respect to the checkpoint solution.

Method	∇ Speedup	$\mathcal{H}\boldsymbol{v}$ Speedup	Relative l_2 error, %	Compression Ratio
Autoencoder	1.05	1.17	0.02	128
ZFP, $\eta = 10^{-5}$	1.0	1.09	0.02	7.2
ZFP, $\eta = 10^{-4}$	1.05	1.16	0.02	8.4
ZFP, $\eta = 10^{-3}$	1.05	1.16	0.02	11.2
ZFP, $\eta = 10^{-2}$	1.06	1.17	0.05	17.6

$\pi_{\text{prior}}(\boldsymbol{u})$ is the prior probability density from the Bayesian formulation of the inverse problem. The DIAS solution is then defined to be

$$\boldsymbol{u}_{\text{DIAS}} = \underset{\boldsymbol{u}}{\text{argmin}} \frac{1}{2} \|\boldsymbol{\mathcal{F}}(\boldsymbol{u}) - \boldsymbol{d}\|^2_{\Gamma^{-1}_{\text{noise}}} + \frac{1}{2} \left\| \boldsymbol{W}_2^T (\boldsymbol{u} - \boldsymbol{u}_0) \right\|^2_{\left(\boldsymbol{W}_2^T \Gamma_{\text{prior}} \boldsymbol{W}_2\right)^{-1}}, \tag{8}$$

that is, it is a Tikhonov solution with regularization acting only on the inactive space.

For large-scale problems, it is infeasible to compute and store the large matrix $\boldsymbol{W}_2$ since the active subspace tends to be much smaller than the inactive subspace. To avoid explicitly computing and storing the potentially large matrix $\boldsymbol{W}_2$, it can be shown using Schott (2016), theorem 5.8 that

$$\left\|\boldsymbol{W}_2^T(\boldsymbol{u}-\boldsymbol{u}_0)\right\|^2_{\left(\boldsymbol{W}_2^T\Gamma_{\text{prior}}\boldsymbol{W}_2\right)^{-1}} = \left\|\boldsymbol{P}_2(\boldsymbol{u}-\boldsymbol{u}_0)\right\|^2_{(\boldsymbol{P}_2\Gamma_{\text{prior}}\boldsymbol{P}_2)^\dagger} \tag{9}$$

where $\boldsymbol{P}_2 := \boldsymbol{W}_2\boldsymbol{W}_2^T = \mathcal{I} - \boldsymbol{W}_1\boldsymbol{W}_1^T$ is a projection matrix onto the inactive subspace and $(\boldsymbol{P}_2\Gamma_{\text{prior}}\boldsymbol{P}_2)^\dagger$ denotes the pseudoinverse.

Lest we conclude that we have a free lunch in computing the regularization term arising from the active subspace prior via $\boldsymbol{W}_1$, one more challenge remains. Note that we need to apply not the prior covariance but its (pseudo-)inverse. That is, we need to compute $(\boldsymbol{P}_2\Gamma_{\text{prior}}\boldsymbol{P}_2)^\dagger$. There are several factors at play here.

1. Γ_{prior} is often a sparse matrix, while $\boldsymbol{P}_2\Gamma_{\text{prior}}\boldsymbol{P}_2$ is very likely a dense matrix.
2. We must invert the dense matrix $\boldsymbol{P}_2\Gamma_{\text{prior}}\boldsymbol{P}_2$ using the pseudoinverse.
3. This matrix is in the dimension of the parameter, n.

For cases where it is feasible to form and invert $\boldsymbol{P}_2\Gamma_{\text{prior}}\boldsymbol{P}_2$, it is also feasible to simply compute $\boldsymbol{W}_2$ explicitly. Then we can work with the lower dimensional form given in (8) which still involves inversion, but of a lower dimensional, full rank matrix. For cases where this is not feasible, we need to develop more sophisticated techniques for computing or estimating $\boldsymbol{P}_2\Gamma_{\text{prior}}\boldsymbol{P}_2$.

The first thing one might try is to find an identity that allows us to break the projection out of the pseudoinverse, leaving Γ_{prior} by itself. We might hope that

$$(\boldsymbol{P}_2\Gamma_{\text{prior}}\boldsymbol{P}_2)^\dagger = \boldsymbol{P}_2\Gamma_{\text{prior}}^{-1}\boldsymbol{P}_2,$$

but this is unfortunately not the case, as we show using Schur complements. Decompose the prior covariance as

$$\Gamma_{\text{prior}} = \left[\begin{array}{c|c} \boldsymbol{W}_1^T\Gamma_{\text{prior}}\boldsymbol{W}_1 & \boldsymbol{W}_1^T\Gamma_{\text{prior}}\boldsymbol{W}_2 \\ \hline \boldsymbol{W}_2^T\Gamma_{\text{prior}}\boldsymbol{W}_1 & \boldsymbol{W}_2^T\Gamma_{\text{prior}}\boldsymbol{W}_2 \end{array}\right]$$

and the inverse prior covariance as

$$\Gamma_{\text{prior}}^{-1} = \left[\begin{array}{c|c} \boldsymbol{W}_1^T\Gamma_{\text{prior}}^{-1}\boldsymbol{W}_1 & \boldsymbol{W}_1^T\Gamma_{\text{prior}}^{-1}\boldsymbol{W}_2 \\ \hline \boldsymbol{W}_2^T\Gamma_{\text{prior}}^{-1}\boldsymbol{W}_1 & \boldsymbol{W}_2^T\Gamma_{\text{prior}}^{-1}\boldsymbol{W}_2 \end{array}\right].$$

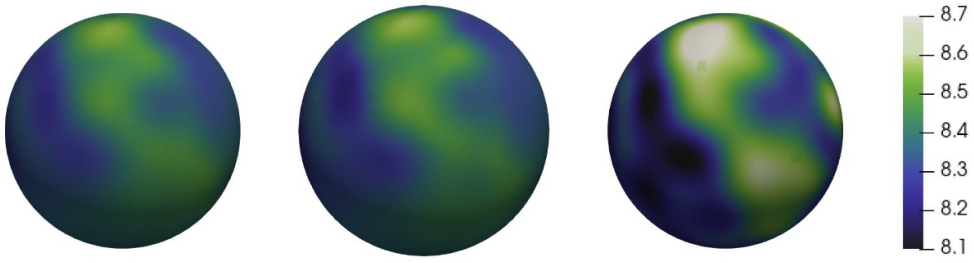
Using Schur complements gives

$$\boldsymbol{W}_2^T\Gamma_{\text{prior}}^{-1}\boldsymbol{W}_2 = \left[\boldsymbol{W}_2^T\Gamma_{\text{prior}}\boldsymbol{W}_2 - \boldsymbol{W}_2^T\Gamma_{\text{prior}}\boldsymbol{W}_1\left(\boldsymbol{W}_1^T\Gamma_{\text{prior}}\boldsymbol{W}_1\right)^{-1}\boldsymbol{W}_1^T\Gamma_{\text{prior}}\boldsymbol{W}_2\right]^{-1}. \tag{10}$$

While the inverse can be expanded using the Sherman–Morrison–Woodbury formula, the important observation is that the second term,

$$\boldsymbol{W}_2^T\Gamma_{\text{prior}}\boldsymbol{W}_1\left(\boldsymbol{W}_1^T\Gamma_{\text{prior}}\boldsymbol{W}_1\right)^{-1}\boldsymbol{W}_1^T\Gamma_{\text{prior}}\boldsymbol{W}_2$$

is not necessarily 0, except when Γ_{prior} is scaled identity. Thus, making the approximation $(\boldsymbol{P}_2\Gamma_{\text{prior}}\boldsymbol{P}_2)^\dagger = \boldsymbol{P}_2\Gamma_{\text{prior}}^{-1}\boldsymbol{P}_2$ incurs some error in the general case. Even though there is some error, it becomes necessary from a practical perspective to use the approximate form $\boldsymbol{P}_2\Gamma_{\text{prior}}^{-1}\boldsymbol{P}_2$ for large-scale problems.



(a) MAP (b) DIAS (c) True

Figure 9. The DIAS solution does not differ noticeably from the MAP solution for the seismic inverse problem. Results are shown for mesh with 11 166 784 DoFs.

Before the DIAS algorithm can be applied to the seismic inverse problem, we must first define an algorithm for applying DIAS to nonlinear inverse problems. One advantage of using the active subspace to define the data-informed subspace is that the active subspace is formed from the *global* average of the outer product of the gradient. This would indicate that the DIAS algorithm can be applied naively to nonlinear inverse problems:

$$\boldsymbol{u}_{\text{DIAS}} = \underset{\boldsymbol{u}}{\text{argmin}} \frac{1}{2} \left\| \boldsymbol{\mathcal{F}}(\boldsymbol{u}) - \boldsymbol{d} \right\|^2_{\Gamma^{-1}_{\text{noise}}} + \frac{1}{2} \left\| \boldsymbol{P}_2 (\boldsymbol{u} - \boldsymbol{u}_0) \right\|^2_{(\boldsymbol{P}_2 \Gamma_{\text{prior}} \boldsymbol{P}_2)^{-1}} . \tag{11}$$

However, the fact that the gradient samples are computed from draws of the prior biases the active subspace toward the directions in which the misfit function f is most sensitive near the prior mean, $\boldsymbol{u}_0$.

Recalling that the goal of the DIAS algorithm is to remove regularization in directions that the data are more informative and that the DIAS solutions are likely closer to the usual Bayesian solution than to the prior mean for a well-chosen prior, we propose a two-step algorithm:

1. Approximately solve the usual MAP problem for $\boldsymbol{u}_{\text{MAP}}$ using the full prior:

$$\boldsymbol{u}_{\text{MAP}} = \underset{\boldsymbol{u}}{\text{argmin}} \frac{1}{2} \left\| \boldsymbol{\mathcal{F}}(\boldsymbol{u}) - \boldsymbol{d} \right\|^2_{\Gamma^{-1}_{\text{noise}}} + \frac{1}{2} \left\| \boldsymbol{u} - \boldsymbol{u}_0 \right\|^2_{\Gamma^{-1}_{\text{prior}}} . \tag{12}$$

2. Compute the active subspace centered at $\boldsymbol{u}_{\text{MAP}}$ and, with the initial guess $\boldsymbol{u}_{\text{MAP}}$, solve the DIAS problem:

$$\boldsymbol{u}_{\text{DIAS}} = \underset{\boldsymbol{u}}{\text{argmin}} \frac{1}{2} \left\| \boldsymbol{\mathcal{F}}(\boldsymbol{u}) - \boldsymbol{d} \right\|^2_{\Gamma^{-1}_{\text{noise}}} + \frac{1}{2} \left\| \boldsymbol{P}_2 (\boldsymbol{u} - \boldsymbol{u}_0) \right\|^2_{\boldsymbol{P}_2 \Gamma^{-1}_{\text{prior}} \boldsymbol{P}_2} .$$

Even with acceleration via autoencoder compression, the cost of gradient evaluations is still high. We estimate the active subspace using 30 gradient samples and take the active subspace to have dimension 5. The difference between the DIAS and the MAP estimate is small in this case. Compared to the true solution, the MAP estimate has an average relative error of 0.3293% while the DIAS refined solution has average relative error of 0.3291%. This is reflected visually in figure 9.

5. Conclusions

In this paper, we proposed an autoencoder compression algorithm to mitigate the high storage requirements for solving large-scale time-dependent PDE-constrained inverse problems. We show the feasibility and scalability of this approach against the popular checkpointing strategy for a seismic inverse problem on both a simple box domain with uniform refinement and a complex spherical domain with adaptive mesh refinement and a nonconforming mesh. We present a data generation procedure based on the Bayesian formulation that provides a problem-focused data generation scheme to effectively train the autoencoder. We proposed two separate autoencoder compression algorithms for spatial and temporal domains. The scalability of each algorithm was shown, providing a clear close-to-ideal speed advantage on large-scale problems compared to the traditional checkpointing approach and a substantial improvement in the compression ratio of the state-of-the-art floating point compression package, ZFP. While there are clear advantages of using autoencoders to perform compression, care must be taken in their implementation for an advantage over the checkpointing approach. As an important application of the proposed autoencoder algorithm, we combined the proposed autoencoder compression approach with the DIAS prior showing how the DIAS method can be affordably extended to large-scale problems without the need for checkpointing and large memory.

Ongoing work is to apply our approach to implicit code such as SPECFEM3D Globe (Komatitsch and Tromp 2002), and in this case, we expect an order of magnitude reduction in computational time. Note that we have limited ourselves to acoustic wave inversion, but more realistic seismic inversion must solve elastic wave or coupled elastic-acoustic wave equations. Part of our work is to investigate if our approach still works in this more realistic regime, and to develop extensions when it does not. Lastly, we aim to incorporate on-the-fly fine-tuning to the machine learning model to further improve the compression capability and adaptability to new settings.

Data availability statement

The data cannot be made publicly available upon publication because they are not available in a format that is sufficiently accessible or reusable by other researchers. The data that support the findings of this study are available upon reasonable request from the authors.

Acknowledgments

We would like to thank Georg Stadler for his many correspondences and helpful advice on stably solving the forward and inverse problem in large scale runs; Jau-Uei Chen for sharing his insight on the discontinuous Galerkin method; Sheroze Sheriffdeen and Hwan Goh for their many discussions on autoencoders, compression, and HPC; and the Texas Advanced Computing Center (TACC) at the University of Texas at Austin for providing HPC resources that contributed to the results presented in this work. URL: http://www.tacc.utexas.edu. This research is partially funded by the National Science Foundation Awards NSF-OAC-2212442, NSF-2108320, NSF-1808576 and NSF-CAREER-1845799; and by the Department of Energy Awards DE-SC0018147 and DE-SC0022211.

ORCID iD

Jonathan Wittmer https://orcid.org/0000-0002-7538-5932

References

Abu Alsheikh M, Poh P K, Lin S, Tan H-P and Niyato D 2014 Efficient data compression with error bound guarantee in wireless sensor networks *Proc. 17th ACM Int. Conf. on Modeling, Analysis and Simulation of Wireless and Mobile Systems* pp 307–11

Akturk I and Karpuzcu U R 2018 Trading computation for communication: a taxonomy of data recomputation techniques *IEEE Trans. Emerg. Top. Comput.* **9** 496–506

Baldi P and Hornik K 1989 Neural networks and principal component analysis: learning from examples without local minima *Neural Netw.* **2** 53–58

Bharadwaj K K, Srivastava A, Panda M K, Singh Y D, Maharana R, Mandal K, Singh M, Singh D, Das M, Murmu D *et al* 2021 Computational intelligence in vaccine design against COVID-19 *Computational Intelligence Methods in Covid-19: Surveillance, Prevention, Prediction and Diagnosis* (Springer) pp 311–29

Bhattacharjee A and Wells J 2021 Preface to special topic: building the bridge to the exascale—applications and opportunities for plasma physics *Phys. Plasmas* **28** 090401

Bickelhaupt F M and Baerends E J 2000 Kohn-sham density functional theory: predicting and understanding chemistry *Reviews in Computational Chemistry* (Wiley) pp 1–86

Bjerge K, Mann H M R, Høye T T, Sankey T and Ahumada J 2022 Real-time insect tracking and monitoring with computer vision and deep learning *Remote Sens. Ecol. Conserv.* **8** 315–27

Blesser B 1969 Audio dynamic range compression for minimum perceived distortion *IEEE Trans. Audio Electroacoust.* **17** 22–32

Boehm C, Hanzich M, de la Puente J and Fichtner A 2016 Wavefield compression for adjoint methods in full-waveform inversion *Geophysics* **81** R385–97

Bui-Thanh T, Burstedde C, Ghattas O, Martin J, Stadler G and Wilcox L C 2012 Extreme-scale UQ for Bayesian inverse problems governed by PDEs *SC'12: Proc. Int. Conf. on High Performance Computing, Networking, Storage and Analysis* (IEEE) pp 1–11

Bui-Thanh T, Ghattas O, Martin J and Stadler G 2013 A computational framework for infinite-dimensional Bayesian inverse problems part I: the linearized case, with application to global seismic inversion *SIAM J. Sci. Comput.* **35** A2494–523

Bukowski R, Sun Q, Howard M and Pillardy J 2010 BioHPC: computational biology application suite for high performance computing *J. Biomol. Tech.* **21** S23

Cheng Z, Sun H, Takeuchi M and Katto J 2018 Deep convolutional autoencoder-based lossy image compression *2018 Picture Coding Symp. (PCS)* (IEEE) pp 253–7

Cyr E C, Shadid J and Wildey T 2015 Towards efficient backward-in-time adjoint computations using data compression techniques *Comput. Methods Appl. Mech. Eng.* **288** 24–44

De Jong W A, Bylaska E, Govind N, Janssen C L, Kowalski K, Müller T, Nielsen I M B, van Dam H J J, Veryazov V and Lindh R 2010 Utilizing high performance computing for chemistry: parallel computational chemistry *Phys. Chem. Chem. Phys.* **12** 6896–920

Denis A, Jeannot E and Swartvagher P 2022 Modeling memory contention between communications and computations in distributed HPC systems *IPDPS-2022-IEEE Int. Parallel and Distributed Processing Symp. Workshops* p 10

Dillon B, Plehn T, Sauer C and Sorrenson P 2021 Better latent spaces for better autoencoders *SciPost Phys.* **11** 061

Doi J, Takahashi H, Raymond R, Imamichi T and Horii H 2019 Quantum computing simulator on a heterogenous HPC system *Proc. 16th ACM Int. Conf. on Computing Frontiers* pp 85–93

Duarte E F, da Costa C A N, de Araújo J M, Wang Y and Rao Y 2020 Seismic shot-encoding schemes for waveform inversion *J. Geophys. Eng.* **17** 906–13

Epanomeritakis I, Akçelik V, Ghattas O and Bielak J 2008 A Newton-CG method for large-scale three-dimensional elastic full-waveform seismic inversion *Inverse Problems* **24** 034015

Esteva A, Chou K, Yeung S, Naik N, Madani A, Mottaghi A, Liu Y, Topol E, Dean J and Socher R 2021 Deep learning-enabled medical computer vision *npj Digit. Med.* **4** 1–9

Fedeli L, Huebl A, Boillod-Cerneux F, Clark T, Gott K, Hillairet C, Jaure S, Leblanc A, Lehe R, Myers A *et al* 2022 Pushing the frontier in the design of laser-based electron accelerators with

groundbreaking mesh-refined particle-in-cell simulations on exascale-class supercomputers *2022 SC22: Int. Conf. for High Performance Computing, Networking, Storage and Analysis (SC)* (IEEE Computer Society) pp 25–36
Fichtner A, Bunge H-P and Igel H 2006 The adjoint method in seismology: I. Theory *Phys. Earth Planet. Inter.* **157** 86–104
Ge H *et al* 2013 Molecular dynamics-based virtual screening: accelerating the drug discovery process by high-performance computing *J. Chem. Inf. Model.* **53** 2757–64
Giles M B and Pierce N A 2000 An introduction to the adjoint approach to design *Flow Turbul. Combust.* **65** 393–415
Goh H, Sheriffdeen S, Wittmer J and Bui-Thanh T 2019 Solving Bayesian inverse problems via variational autoencoders (arXiv:1912.04212)
Griewank A and Walther A 1997 Treeverse: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation *Technical Report IOKOMO-04-1997* (Technische Universitat of Dresden)
Habashy T, Abubakar A, Pan G and Belani A 2011 Source-receiver compression scheme for full-waveform seismic inversion *Geophysics* **76** R95–R108
Hinton G E and Salakhutdinov R R 2006 Reducing the dimensionality of data with neural networks *Science* **313** 504–7
Imani M, Gupta S, Kim Y, Zhou M and Rosing T 2019 DigitalPIM: digital-based processing in-memory for big data acceleration *Proc. 2019 on Great Lakes Symp. on VLSI* pp 429–34
Intel® xeon® platinum 8280 processor (38.5 m cache, 2.70 ghz)—product specifications
Ji S, Xu W, Yang M and Yu K 2012 3D convolutional neural networks for human action recognition *IEEE Trans. Pattern Anal. Mach. Intell.* **35** 221–31
Kaipio J and Somersalo E 2006 *Statistical and Computational Inverse Problems* vol 160 (Springer)
Kamath U, Liu J and Whitaker J 2019 *Deep Learning for NLP and Speech Recognition* vol 84 (Springer)
Kates-Harbeck J, Svyatkovskiy A and Tang W 2019 Predicting disruptive instabilities in controlled fusion plasmas through deep learning *Nature* **568** 526–31
Kaur R and Choudhary P 2016 A review of image compression techniques *Int. J. Comput. Appl* **142** 8–11
Kingma D P and Welling M 2013 Auto-encoding variational Bayes (arXiv:1312.6114)
Kneer S, Sayadi T, Sipp D, Schmid P and Rigas G 2021 Symmetry-aware autoencoders: s-PCA and s-nlPCA (arXiv:2111.02893)
Komatitsch D and Tromp J 2002 Spectral-element simulations of global seismic wave propagation—I. Validation *Geophys. J. Int.* **149** 390–412
Kowalski K *et al* 2021 From NWChem to NWChemEx: evolving with the computational chemistry landscape *Chem. Rev.* **121** 4962–98
Kukreja N, Hückelheim J, Louboutin M, Hovland P and Gorman G 2019 Combining checkpointing and data compression to accelerate adjoint-based optimization problems *European Conf. on Parallel Processing* (Springer) pp 87–100
Kukreja N, Hückelheim J, Louboutin M, Washbourne J, Kelly P H J and Gorman G J 2022 Lossy checkpoint compression in full waveform inversion: a case study with ZFPv0. 5.5 and the overthrust model *Geosci. Model Dev.* **15** 3815–29
Ladjal S, Newson A and Pham C-H 2019 A PCA-like autoencoder (arXiv:1904.01277)
Lee J, Gong Q, Choi J, Banerjee T, Klasky S, Ranka S and Rangarajan A 2022 Error-bounded learned scientific data compression with preservation of derived quantities *Appl. Sci.* **12** 6718
Lewis A and Knowles G 1990 Video compression using 3D wavelet transforms *Electron. Lett.* **26** 396–8
Lindstrom P 2014 Fixed-rate compressed floating-point arrays *IEEE Trans. Vis. Comput. Graphics* **20** 2674–83
Liu J, Di S, Zhao K, Jin S, Tao D, Liang X, Chen Z and Cappello F 2021a Exploring autoencoder-based error-bounded compression for scientific data *2021 IEEE Int. Conf. on Cluster Computing (CLUSTER)* (IEEE) pp 294–306
Liu T, Wang J, Liu Q, Alibhai S, Lu T and He X 2021b High-ratio lossy compression: exploring the autoencoder to compress scientific data *IEEE Trans. Big Data* **9** 22–36
Liu Y *et al* 2021c Closing the "quantum supremacy" gap: achieving real-time simulation of a random quantum circuit using a new sunway supercomputer *Proc. Int. Conf. for High Performance Computing, Networking, Storage and Analysis* pp 1–12

Mandrà S, Marshall J, Rieffel E G and Biswas R 2021 Hybridq: a hybrid simulator for quantum circuits *2021 IEEE/ACM 2nd Int. Workshop on Quantum Computing Software (QCS)* (IEEE) pp 99–109
McFarlane R and Biktasheva I V 2008 Beatbox-a computer simulation environment for computational biology of the heart *Visisions of Computer Science—BCS Int. Academic Conf.* (British Computer Society) pp 99–109
Nadeau C and Bengio Y 1999 Inference for the generalization error *Advances in Neural Information Processing Systems* p 12
Nguyen H, Wittmer J and Bui-Thanh T 2022 Dias: a data-informed active subspace regularization framework for inverse problems *Computation* **10** 38
Peng I, Wu K, Ren J, Li D and Gokhale M 2020 Demystifying the performance of HPC scientific applications on NVM-based memory systems *2020 IEEE Int. Parallel and Distributed Processing Symp. (IPDPS)* (IEEE) pp 916–25
Phillips T R F, Heaney C E, Smith P N and Pain C C 2021 An autoencoder-based reduced-order model for eigenvalue problems with application to neutron diffusion *Int. J. Numer. Methods Eng.* **122** 3780–811
Plaut E 2018 From principal subspaces to principal components with linear autoencoders (arXiv:1804.10253)
Quarteroni A, Sacco R and Saleri F 2010 *Numerical Mathematics* vol 37 (Springer)
Reichstein M, Camps-Valls G, Stevens B, Jung M, Denzler J, Carvalhais N and Prabhat K 2019 Deep learning and process understanding for data-driven earth system science *Nature* **566** 195–204
Scales J A and Tenorio L 2001 Prior information and uncertainty in inverse problems *Geophysics* **66** 389–97
Schmidt B and Hildebrandt A 2017 Next-generation sequencing: big data meets high performance computing *Drug Discov. Today* **22** 712–7
Schott J R 2016 *Matrix Analysis for Statistics* (Wiley)
Stanzione D, West J, Evans R T, Minyard T, Ghattas O and Panda D K 2020 Frontera: the evolution of leadership computing at the national science foundation *Practice and Experience in Advanced Research Computing* pp 106–11
Sukumar S R *et al* 2021 The convergence of HPC, AI and big data in rapid-response to the COVID-19 pandemic *Smoky Mountains Computational Sciences and Engineering Conf.* (Springer) pp 157–72
Usevitch B E 2001 A tutorial on modern lossy wavelet image compression: foundations of JPEG 2000 *IEEE Signal Process. Mag.* **18** 22–35
Wang Q, Moin P and Iaccarino G 2009 Minimal repetition dynamic checkpointing algorithm for unsteady adjoint calculation *SIAM J. Sci. Comput.* **31** 2549–67
Wilcox L C, Stadler G, Bui-Thanh T and Ghattas O 2015 Discretely exact derivatives for hyperbolic PDE-constrained optimization problems discretized by the discontinuous Galerkin method *J. Sci. Comput.* **63** 138–62
Wu K, Ober F, Hamlin S and Li D 2017 Early evaluation of intel optane non-volatile memory with HPC I/O workloads (arXiv:1708.02199)
Yang J 2009 Newton-conjugate-gradient methods for solitary wave computations *J. Comput. Phys.* **228** 7007–24
You Y, Li J, Reddi S, Hseu J, Kumar S, Bhojanapalli S, Song X, Demmel J, Keutzer K and Hsieh C-J 2019 Large batch optimization for deep learning: training bert in 76 minutes (arXiv:1904.00962)
Zhang H and Constantinescu E M 2023 Optimal checkpointing for adjoint multistage time-stepping schemes *J. Comput. Sci.* **66** 101913
Zhao Y, Chen X, Wang Y, Li C, You H, Fu Y, Xie Y, Wang Z and Lin Y 2020 Smartexchange: trading higher-cost memory storage/access for lower-cost computation *2020 ACM/IEEE 47th Annual Int. Symp. on Computer Architecture (ISCA)* (IEEE) pp 954–67