Deep Learning for Full Waveform Inversion

Exploring hybrid CNN-Transformer approach to reconstruct high-fidelity velocity models from seismic waveforms

Gaurav Khanal[0000−0001−2345−6789]

Université Côte d'Azur

Abstract. Geophysical imaging relies on reconstructing the Earth's subsurface structures from seismic waveform measurements. Full Waveform Inversion (FWI) is the leading technique for this task, yet its practical application is hampered by prohibitive computational costs, inherent ill-posedness, and susceptibility to cycle skipping. This paper introduces a deep learning approach to address these challenges, employing a hybrid Convolutional Neural Network (CNN) and Transformer architecture for data-driven FWI. Our proposed model leverages a Swin V2 Transformer encoder to capture long-range wavefield dependencies and an advanced U-Net decoder for precise velocity model reconstruction.

Keywords: Full Waveform Inversion, Deep Learning, Swin Transformer, U-Net, Geophysical Imaging, OpenFWI, Seismic Inversion

1. Introduction

1.1 The Physics of Geophysical Imaging

Geophysical imaging (also known as seismic imaging, subsurface imaging or geophysical tomography) is the process of creating a detailed picture of the Earth's interior from measurements taken at Earth's surface often with the help of or in accordance with geophysics. This practice is fundamental to a wide range of fields, from resource exploration and environmental monitoring to civil engineering and seismic hazard assessment [46, 68].

The imaging process begins by generating controlled seismic waves using an artificial source, such as vibrator trucks on land or compressed air guns in marine environments [46, 45, 3]. These waves propagate downward into the Earth. As they travel, they encounter different geological layers and structures. At the interface between materials with different physical properties—primarily contrasts in acoustic impedance (the product of density and velocity)—a portion of the wave energy is reflected back toward the surface [46, 52].

This reflected energy is captured by an array of sensitive receivers, known as geophones on land or hydrophones in water [45, 3]. Each receiver records a time

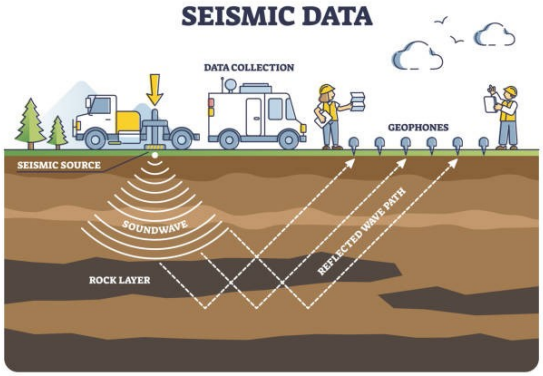

Figure 1: Subsurface Imaging. Source: https://www.kaggle.com/code/hanchenwang114/waveform-inversion-kaggle-competition-tutorial. Accessed July 2025.

series of ground motion or pressure changes, called a seismic trace. The collection of traces from a single source event forms a "shot gather," which contains a rich, albeit complex, record of the wavefield's interaction with the subsurface [3].

The central challenge of geophysical imaging is to translate these recorded waveforms into a geologically meaningful image. This is fundamentally an inverse problem: using the observed effect (the seismic data) to infer the cause (the subsurface properties) [45, 3]. The most critical property to determine is the seismic velocity, which dictates how fast the waves travel through different rock formations [60, 65, 24].

The entire process is governed by the physics of wave propagation, which is mathematically described by the wave equation [65, 61]. This second-order partial differential equation describes the evolution of the acoustic pressure wavefield, p, as a function of position, $\mathbf{x}$, and time, t. In an isotropic medium with constant density, it is expressed as:

$$\frac{1}{v(\mathbf{x})^2} \cdot \frac{\partial^2 p(\mathbf{x},t)}{\partial t^2} - \nabla^2 p(\mathbf{x},t) = s(\mathbf{x},t) \tag{1}$$

where,

- $v(\mathbf{x})$ is the spatially varying acoustic velocity
- ∇^2 is the Laplacian operator, and
- $s(\mathbf{x},t)$ is the seismic source/wave.

This equation forms the basis of the "forward model"—given a velocity model $v(\mathbf{x})$, it allows us to simulate the seismic data $s(\mathbf{x},t)$ that would be recorded at the surface at time t [57]. Hence, The goal of seismic inversion is to find the velocity model $v(\mathbf{x})$ that causes the simulated data to best match the actual

observed data. Among the various techniques developed to solve this complex inverse problem, Full Waveform Inversion (FWI) has emerged as the most powerful and comprehensive.

1.2 Full Waveform Inversion (FWI)

FWI is mathematically formulated as a large-scale, non-linear inverse problem, typically solved through iterative local optimization. The objective is to minimize a misfit functional, $J(v)$, that quantifies the discrepancy between the observed data, d_{obs}, and the synthetic data, $F(v)$, generated by a forward model. The forward model, F, numerically solves a wave-propagation partial differential equation (PDE), such as the acoustic wave equation, for a given velocity model, v, [55, 33]. The optimization problem is generally expressed as:

$$\min_{v} J(v) = \|d_{obs} - F(v)\|_2^2 + \lambda R(v) \tag{2}$$

Here, the first term represents the data misfit, often measured by the L_2-norm, while the second term, $R(v)$, is a regularization functional weighted by a parameter λ. Regularization is essential to constrain the solution space and ensure the recovery of geologically plausible models, given the inherent ill-posedness of the problem [55].

1.3 The Limitations of Conventional FWI

Despite its theoretical power to produce images at half the propagated wavelength, the practical application of conventional FWI is hindered by several fundamental and deeply interconnected challenges that have limited its routine use, particularly in geologically complex environments [Zerafa, Galea, and Sebu [67]; Pratt [35]; Li, Li, and Fomel [21]. These challenges can be categorized into three primary areas.

1. **Prohibitive Computational Cost**: First, FWI is characterized by a prohibitive computational cost. The inversion process is iterative, and each iteration requires solving the full wave equation for every seismic source in the survey to compute the synthetic data and the gradient of the misfit function [64].
2. **Ill-posed Problem**: Second, the FWI problem is inherently ill-posed [55, 33, 2]. This mathematical property means that the solution is highly sensitive to noise and perturbations in the input data, and a unique solution is not guaranteed—multiple distinct velocity models may explain the observed data equally well [14].
3. **Cycle Skipping**: Conventional gradient-based FWI is plagued by the cycle-skipping problem. This issue is a direct manifestation of the highly non-convex nature of the FWI objective function [54, 53]. Cycle-skipping occurs

when the phase difference between an event in the observed seismogram and the corresponding event in the synthetic data exceeds half a wavelength.

1.4 Deep Learning as a Data-Driven Inversion Paradigm

Deep Learning (DL) has emerged as a promising new paradigm that fundamentally reframes the FWI problem, offering a potential path to overcome the long-standing limitations of conventional methods [67, 65]. The core concept is to train a deep neural network to learn the inverse operator, F^{-1}, that maps seismic data directly to the corresponding subsurface velocity model [55, 63]. This is often formulated as an image-to-image translation task, where input seismic shot gathers are translated into output velocity maps [63, 69]. There have been several distinct architectural approaches in DL:

1.4.1 Convolutional Encoder-Decoders are typically based on the U-Net architecture [43], represent the foundational approach. The InversionNet model, proposed by Wu and Lin [63], became the canonical example. They demonstrated that a U-Net could be trained to map raw shot gathers to velocity models in near-real-time, effectively learning the inverse operator and serving as a key baseline for subsequent research, including the OpenFWI benchmarks [63, 66, 23].

1.4.2 Generative Adversarial Networks (GANs) were introduced to address a common shortcoming of simple encoder-decoder models, which can produce overly smooth or blurry velocity maps that lack sharp geological features. To combat this, Zhang and Lin [69] introduced VelocityGAN. By adding a discriminator network trained to distinguish between the U-Net's output and real velocity models, they used an adversarial loss to force the generator to produce models with sharper, more geologically plausible features. This effectively learns a data-driven regularization that promotes realism in the output [69, 47].

1.4.3 Physics-Informed Neural Networks (PINNs) represent a hybrid approach that seeks to bridge the gap between purely data-driven methods and first-principles physics [55, 65]. Instead of relying solely on vast quantities of labeled data, PINNs incorporate physical knowledge by embedding the governing PDE—in this case, the wave equation—as a soft constraint directly into the network's loss function. For instance, Goren and Treister [10] have explored frameworks where the residual of the acoustic wave equation is added as a penalty term to the loss function. This forces the network's output to not only match the data but also to be physically consistent, which can improve generalization, especially when training data is limited [55, 65, 30].

1.4.4 Transformers have begun to be explored for seismic inversion tasks [12, 8]. The latest frontier involves adapting Transformer architectures, originally from natural language processing, to seismic problems. Harsuko and Alkhalifah

[12], for instance, proposed StorSeismic, a framework using Transformers to process seismic data sequentially. They argue that the self-attention mechanism is better suited to capture the long-range dependencies inherent in wave propagation than the local receptive fields of CNNs, a hypothesis central to our work.

The following table provides a concise comparison of these dominant deep learning paradigms for FWI.

Table 1: Comparison of Inversion Approaches

Approach	Core Architecture	Guiding Principle	Key Advantage(s)	Primary Limitation(s)	Canonical Example(s) & Citation(s)
Direct Inversion	CNN Encoder-Decoder (U-Net)	Supervised Learning (Image-to-Image Translation)	Extremely fast inference; conceptually simple.	Data-hungry; may produce blurry results; struggles with out-of-distribution generalization.	InversionNet [**16**]
Generative Inversion	Generative Adversarial Network (GAN)	Unsupervised/ Adversarial Learning	Learns data distribution as implicit regularization; produces sharper, more realistic models.	Unstable training; mode collapse; still data-hungry.	VelocityGAN [**43; 44**] FWIGAN [**25**]
Physics-Informed	Feed-Forward Network or CNN	PDE-Constrained Optimization	Enforces physical consistency; reduces reliance on labeled data; better generalization.	Higher training cost per iteration; complex loss function formulation.	PINN-FWI [**32**], Physics-Guided DL [**14**]
Sequence Modeling	Transformer	Self-Attention Mechanism	Captures long-range dependencies in wavefields; theoretically robust to input size variations.	Very high computational cost; data-hungry; less mature in FWI applications.	StorSeismic [**52**], CLWTNet [**51**]

1.5 OpenFWI

The OpenFWI initiative was launched to address the lack of large-scale, public and standardized benchmark datasets [6, 17, 23, 22]. It is large-scale and multi-scale, with a total size of over 2 TB and individual datasets containing between 2,000 and 67,000 data-model pairs, enabling novel studies on the impact of "big data" on FWI model performance and generalization [6, 17, 15, 16]. It is also multi-structural and diverse, encompassing a wide range of geological priors and complexities. These are organized into families, such as the Vel family with flat and curved layers, the Fault family containing various fault structures, and the Style family with complex textures derived from natural images [6, 18, 19, 31]

The release of this resource was accompanied by community-focused events, most notably the "Yale/UNC-CH - Geophysical Waveform Inversion" Kaggle competition [18, 31, 58]. This competition served as a catalyst, spurring widespread research and innovation using the OpenFWI datasets. It provided a clear and well-defined challenge: to develop novel, physics-guided machine learning models capable of solving the FWI problem as framed by this new, comprehensive benchmark [29]. This paper is situated directly within this ongoing community-wide effort as we also participated in this competition.

1.6 Our Work

This paper presents our final model we developed as part of the project to address a data-driven approach to FWI. Our proposed model is designed to more effectively capture long-range wavefield interactions via self-attention while maintaining the computational efficiency and strong inductive bias of convolutional layers, addressing a key limitation of prior U-Net based approaches.

2 Metadata

This research is built upon the OpenFWI benchmark dataset, a large-scale, open-access collection of datasets designed to facilitate diversified, rigorous, and reproducible research in data-driven FWI [23, 6, 17, 22, 6]. OpenFWI provides a comprehensive platform with multiple datasets encompassing a wide range of geological structures, complexities, and scales, making it an ideal foundation for developing and testing robust deep learning models [6, 17].

2.1 Data Families and Characteristics

For this project, we utilize three distinct 2D data families from the OpenFWI collection: the "Vel" family, the "Fault" family, and the "Style" family [58]. These families were chosen to represent a diverse set of geological priors and structural complexities [6, 17, 7].

1. Vel Family: This group of datasets is characterized by velocity models with simple layered structures, including both flat and curved interfaces [6, 17, 7]. It provides a fundamental test case for an inversion model's ability to resolve basic subsurface stratigraphy. This family contains 24,000 training samples and 6,000 testing samples [31].

2. Fault Family: This family introduces significant structural complexity in the form of various fault systems [6, 17, 7]. It is designed to test a model's performance on geologically realistic discontinuities, which are a common challenge in seismic imaging. The Fault family is larger, with 48,000 training samples and 6,000 testing samples [31].

3. Style Family: This collection contains the most complex velocity models, with textures derived from natural images to simulate highly heterogeneous and geologically intricate subsurface environments [6, 17, 7]. It serves as a stress test for a model's generalization capabilities. This is the largest family, providing 60,000 training samples and 7,000 testing samples [31].

The Kimberlina dataset family, also part of OpenFWI, was explicitly excluded from this study, as it was designed for a different geophysical task (time-lapse CO_2 monitoring) and falls outside the scope of this competition [18, 42].

2.2 Datasets and Preprocessing

For initial experimentation, we used a reduced subset of the OpenFWI data provided by the competition hosts, which only totaled about 10,000 samples. The data in this original subset had the following dimensions [31, 41]:

1. Seismic Data (Inputs): 4D arrays of shape `(500, 5, 1000, 70)`, representing `(batch_size, num_sources, time_steps, num_receivers)`.
2. Velocity Maps (Outputs): 4D arrays of shape `(500, 1, 70, 70)`, representing `(batch_size, channels, height, width)`.

2.3 Final Preprocessed Dataset

Following the competition's recommendation to train on the full dataset for better performance, we transitioned to the openfwi-preprocessed 72x72 dataset for our main experiments [1]. This dataset maintains the same family structure (Vel, Fault, Style) but contains a much larger volume of samples. The key preprocessing steps applied to this dataset were [1]:

1. The input seismic waveform data was resized from its original dimensions of `1000x70 (time_steps x num_receivers)` to a square shape of `72x72`.
2. The data was converted to `float16` precision for improved computational and memory efficiency.

This preprocessing standardized the input dimensions, making the data more amenable to common deep learning architectures while leveraging the full scale of the OpenFWI benchmark, with about 470,000 samples in total.

3. Methodology and Implementation

3.1 Model Architecture

The heart of any deep learning solution lies in its architectural design. For the Full Waveform Inversion (FWI) problem, which is inherently an image-to-image translation task (mapping seismic data to subsurface velocity models), a U-Net-like architecture with a powerful Swin Transformer backbone is employed. This section delves into the fundamental components and enhancements of this model.

3.1.1 U-Net Architecture The U-Net, originally developed by Ronneberger, Fischer and Brox in 2015 for biomedical image segmentation, is a convolutional neural network architecture characterized by its distinctive 'U' shape [43]. It consists of two main paths:

1. **Encoder (Contracting Path)**: This path is responsible for capturing context and progressively reducing the spatial dimensions of the input while increasing the feature complexity. It is typically comprised of a sequence of

convolutional layers (e.g., nn.Conv2d), followed by activation functions (e.g., ReLU, GELU), and downsampling operations such as max pooling or strided convolutions [43].

2. **Bottleneck**: At the bottom of the U-Net is a botteneck layer that connects the contracting path (encoder) to the expansive path (decoder). The bottleneck layer captures the most abstract and high-level features of the input image.
3. **Decoder (Expansive Path)**: This path aims to precisely reconstruct the output (the velocity model in this case) from the compressed, high-level features learned by the encoder. It is generally comprised of upsampling layers (e.g., transpose convolutions, or interpolation followed by convolutions) and additional convolutional layers.
4. **Skip Connections**: These connections directly concatenate feature maps from corresponding encoder layers to the decoder layers at the same spatial resolution. This allows the decoder to leverage fine-grained, local information lost during downsampling in the encoder, which is vital for precise reconstruction, especially for tasks requiring high-resolution outputs like FWI [43].

3.1.2 Vision Transformers (ViTs) The foundational work on Transformers, "Attention Is All You Need" by Vaswani et al. (2017), revolutionized sequence modeling in natural language processing (NLP) [56]. Dosovitskiy et al. (2020) demonstrated that a pure Transformer architecture, applied directly to sequences of image patches, could achieve state-of-the-art results on image recognition tasks [9], which allowed them to model long-range dependencies across the entire image.

3.1.2.1 Swin Transformer V1

The original Swin (Shifted Window) Transformer, introduced by Liu et al. (2021), addresses some limitations of earlier ViTs, particularly their high computational cost with global self-attention and their lack of inductive biases for local image structures [26].

1. **Shifted Windows**: Instead of computing attention globally, which would lead to quadtratic complexity for images, Swin Transformers compute attention within local, non-overlapping windows. To enable cross-window connections and hierarchical feature learning, a "shifted window" partitioning approach is used in successive layers, which allows information to flow between different windows, capturing both local and broader context [26].
2. **Hierarchical Feature Maps**: Swin builds a hierarchical representation by merging image patches in deeper layers, producing feature maps at various resolutions. This is analogous to the multi-scale feature maps generated by convolutional networks and makes it a suitable backbone for U-Net-like architectures, where skip connections require features at different scales [26].

3.1.2.2 Swin Transformer V2

Swin Transformer V2, presented by Liu et al. (2022b), was developed to address key challenges in scaling up vision models, including training instability, resolution gaps between pre-training and fine-tuning, and the hunger for labeled data. It introduces three main techniques:

1. **Residual Post Normalization**: Swin V1 used pre-normalization (LayerNorm before attention/MLP). However, when scaling up model capacity, activation values in deeper layers could increase dramatically, leading to training instability. Swin V2 adopts a residual post normalization strategy, moving the LayerNorm to the end of the residual branch [25].
2. **Scaled Cosine Attention** Additionally, it replaces the dot-product attention with scaled cosine attention

$$\text{Sim}(q, k) = \frac{\cos(q, k)}{\tau} + B_{ij} \tag{3}$$

   where τ is a learnable scalar and B_{ij} is the relative position bias between pixel i and j. This helps to stabilize training by moderating the magnitude of attention scores, especially for pairs with large impact [25].
3. **Log-Spaced Continuous Relative Position Bias**: Swin V2 proposes using log-spaced coordinates for relative position bias, generated by a small meta-network (MLP). This significantly reduces the extrapolation ratio, allowing for more effective transfer learning across different input resolutions and window sizes [25].
4. **Self-Supervised Pre-training (SimMIM)**: To alleviate the data hungry problem of transformers, Swin V2 leverages SimMIM (Simple Masked Image Modeling) for self-supervised pre-training. This approach helps the model learn robust representations from unlabeled data, enabling the successful training of billion-parameter models with significantly less labeled data [25].

3.1.2.3 Benefits of Swin Transfromer (V1 & V2)

For FWI, the Swin Transformer offers several advantages:

1. **Capturing Long-Range Dependencies**: Seismic waves propagate through complex subsurface structures, and their behavior at one point can be influenced by distant features. Swin's scaled cosine attention performes better at modeling these long-range dependencies.
2. **Handling Variable Input Scales**: The hierarchical nature of Swin Transformer features can implicitly handle variations in the scale of geological features.
3. **Pretrained Weights**: The ability to use pre-trained Swin Transformer weights (e.g., from ImageNet) provides a strong initialization, accelerating convergence and improving performance.

3.1.3 Enhanced Decoder Design To follow the transformr-based encoder, we developed an advanced U-Net-style decoder with the following enhancements:

3.1.3.1 Learned Upsampling

Instead of basic interpolation methods (like bilinear or nearest-neighbor), the we implemented `LearnedUpsample` blocks, which internally employ `nn.PixelShuffle`. This technique, also known as sub-pixel convolution and introduced by Shi et al. (2016), is an upsampling technique that rearranges pixels from a lower-resolution feature map into a higher-resolution output [48].

Given an input tensor $X \in \mathbb{R}^{C\times H\times W}$, a convolution transforms it to $X' \in \mathbb{R}^{(C\cdot r^2)\times H\times W}$. The `PixelShuffle` operation then reshapes X' into $Y \in \mathbb{R}^{C\times(H\cdot r)\times(W\cdot r)}$, where r is the upsampling factor. This allows the model to learn optimal upsampling filters, leading to smoother, better reconstructions compared to fixed interpolation methods [48].

3.1.3.2 Residual Blocks in the Decoder

The decoder also includes `ResidualBlock` components, which is typically comprised of two convolutional layers (`nn.Conv2d`), followed by normalization layers (e.g., `nn.GroupNorm`), activation functions (e.g., `nn.GELU`), and often a dropout layer (`nn.Dropout2d`). The key characteristic is the addition of the input tensor to the output of these layers, creating a shortcut (residual) connection that helps mitigate vanishing gradients and allows the network to learn identity mappings more easily, which improves the model's ability to refine features during the upsampling process.

3.1.3.3 Spatial and Channel Squeeze-and-Excitation (SCSE) Blocks

The `SCSEBlock` is integrated into the `ResidualBlock` and thus into the decoder. SCSE blocks, proposed by Hu et al. (2018), enhance feature representation by performing adaptive recalibration of feature channels and spatial locations [13]. An `SCSEBlock` is comprised of two parallel branches:

1. **Channel Squeeze-and-Excitation (`cSE`)**: This branch uses global average pooling (`nn.AdaptiveAvgPool2d`) to squeeze spatial information into a channel descriptor, followed by two fully connected layers (implemented as `nn.Conv2d` with 1x1 kernels) and a `nn.GELU` activation, ending with a sigmoid activation to produce channel-wise weights [13].
2. **Spatial Squeeze-and-Excitation (`sSE`)**: This branch uses a single convolutional layer (`nn.Conv2d` with 1x1 kernel) followed by a sigmoid activation to produce spatial attention weights [13]. These weights are also multiplied with the input feature map.

3.1.3.4 Integration of Encoder and Decoder Features

The decoder block explicitly concatenates the upsampled features from the previous decoder stage with the corresponding skip connection features from the

Swin V2 backbone. A `channel_reduce` convolution then processes this combined feature map. Furthermore, a block-level skip connection (`self.skip_conv`) is added, allowing the entire decoder block to learn a residual mapping from its input, further enhancing information flow and stability.

3.1.4 Multi-Source Input Handling A key characteristic of seismic data acquisition is the use of multiple sources (e.g., shot points) to illuminate the subsurface. The model is designed to leverage this multi-source information (in our case `NUM_SOURCES=5`):

3.1.5 Independent Source Processing The input seismic data, which has a shape of (`Batch, Num_Sources, Height, Width`), is reshaped to (`Batch x Num_Sources, 1, Height, Width`). This allows each seismic shot record from different sources within a batch to be processed independently by the shared encoder backbone.

3.1.6 Feature Aggregation After encoding, the feature maps from all sources for a given batch are averaged (`.mean(dim=1)`). This aggregation step combines the information gleaned from different source perspectives into a single, comprehensive feature representation. This averaged representation then feeds into the decoder, allowing the model to make a unified prediction of the subsurface velocity model that implicitly accounts for information from all available sources.

The architectural design of this enhanced U-Net model for FWI is a fusion of cutting-edge deep learning concepts. By leveraging powerful backbones like ConvNeXt or Swin Transformer as the encoder, the model benefits from their strong feature extraction capabilities and ability to capture both local and global context, which is vital for high-fidelity image reconstruction.

3.2 Data Management

Effective data handling is paramount for successful deep learning model training. This section explores the strategies we employed for managing, preparing, and augmenting the seismic data, ensuring it is in an optimal format for the neural network.

3.2.1 Dataset Structure and Loading

3.2.1.1 Custom Dataset

Inheriting from PyTorch's `torch.utils.data.Dataset`, the custom `FWIDataset` provides a standardized interface for accessing data samples. It encapsulates the logic for locating, loading, and performing initial processing on each sample.

3.2.1.2 Memory

For large `.npy` files, the dataset employs memory mapping (`mmap_mode='r'`), which allows the program to treat the file on disk as if it were loaded into RAM, but data is only actually read into memory when it's accessed. This significantly reduces memory footprint.

3.2.1.3 Flat Indexing for Efficient Sample Retrieval

Seismic datasets contain multiple samples within a single `.npy` file. `FWIDataset` constructs a `flat_index` that maps a global sample index to a specific file and the sample's position within that file. This allows the `DataLoader` to request any sample efficiently, regardless of which physical file it resides in, optimizing data access patterns.

3.2.1.4 Robust Data Loading

The `__getitem__` method within `FWIDataset` includes a retry mechanism with a `MAX_RETRIES` limit. If a sample fails to load due to corruption or other transient issues, the dataset attempts to load it again, or tries a different sample, preventing training crashes due to isolated data problems. This enhances the overall robustness of the training pipeline.

3.2.2 Data Normalization The input seismic data X is normalized to have a zero mean and unit standard deviation (i.e., $X \sim \mathcal{N}(0, 1)$)using the z-score method. A small ϵ (`MIN_STD_CLAMP`)is added to the standard deviation to prevent division by zero. The target velocity models V are normalized to a $[0, 1]$ range using min-max scaling, based on predefined `VELOCITY_MIN` and `VELOCITY_MAX` values, which we noted to be 1500 m/s and 4500 m/s based on our exploratory data analysis. This ensures that the model's output predictions are within a consistent and bounded range.

3.2.3 Data Splitting Due to the nature of the data, we employ a stratified splitting strategy. This means that instead of a simple random split, the dataset is divided while ensuring that the proportions of different data "groups" (e.g., "Vel", "Style", "Fault") are maintained in both the training and validation sets.

3.2.4 Data Augmentation We employ a `GPUBatchProcessor` which dynamically applies the following comprehensive augmentations on the GPU:

1. **Elastic Deformation**: This technique, popularized by Simard et al. (2003) for digit recognition, involves applying a smooth, random displacement field to the image pixels, effectively "stretching" or "compressing" regions, which helps the model become more robust to variations in subsurface structures.
2. **Fault Simulation**: Geological fault lines introduce sharp discontinuities in velocity models. Simulating these faults by applying random displacements

along generated fault lines helps the model learn to predict and reconstruct such features accurately.

3. **Amplitude Jitter**: This involves randomly scaling the amplitude of the seismic signals, which helps the model become invariant to minor changes in signal strength, focusing on the underlying structural information.
4. **Receiver Dropout**: Simulating the loss of data from certain seismic receivers by setting their corresponding traces to zero.
5. **Gaussian Noise**: Adding random Gaussian noise to the seismic data helps the model become more robust to ambient noise present in real-world seismic recordings.
6. **Velocity Scaling and Smoothing**:
   - *Velocity Scaling*: Randomly scales the entire velocity model, teaching the model to be robust to overall shifts in velocity values.
   - *Velocity Smoothing*: Applies a smoothing filter to the velocity models. This can help the model learn to predict smoother transitions in velocity, which is often a characteristic of geological formations, and can also act as a form of regularization.

3.3 Data Optimizations PyTorch's `DataLoader` is a powerful utility for efficient batching and loading of data. We leverage several optimizations to maximize data throughput and GPU utilization.

1. `num_workers`: This parameter specifies how many subprocesses to use for data loading. A higher number can speed up data loading by parallelizing the process.
2. `pin_memory`: When set to True, the DataLoader copies Tensors into CUDA pinned memory before returning them, which allows for faster and more asynchronous data transfer from CPU to GPU, reducing potential bottlenecks.
3. `persistent_workers`: Setting this to True keeps worker processes alive across epochs, which avoids the overhead of respawning workers at the beginning of each epoch.
4. `prefetch_factor`: This controls the number of batches that each worker loads ahead of time, which helps keep the GPU busy by ensuring that new batches are ready as soon as the current batch finishes improving throughput.
5. `worker_init_fn`: This function is called to ensure data augmentations are applied consistently across workers for a given epoch but vary appropriately between epochs.
6. `multiprocessing_context='fork'`: On Unix-like systems, using 'fork' for the multiprocessing context can be more efficient than 'spawn' as it avoids reloading the entire Python interpreter and modules in each worker.

3.4 Advanced Training Techniques for Optimization and Stability

Beyond the core architecture and data preparation, the effectiveness of deep learning models heavily relies on sophisticated training techniques.

3.4.1 Distributed Data Parallel (DDP) To handle large datasets and complex models efficiently as well as to effectively use the GPU cluster, we implemented PyTorch's `DistributedDataParallel` (DDP).

3.4.1.1 Motivation for DDP

DDP significantly reduces training time by allowing each GPU to process a portion of the batch independently and in parallel [51].In addition, gradients are synchronized and averaged across all GPUs, allowing for larger effective batch sizes than what a single GPU could accommodate, which can improve generalization [51]. And, DDP is highly scalable, supporting efficient training from a few GPUs on a single machine to hundreds across a cluster, making it indispensable for state-of-the-art research [51].

3.4.1.2 How DDP Works

The DDP training process involves several synchronized steps:

1. **Data Scattering**: In each training iteration, we divide the input batch and distributed (scattered) it across all participating GPUs. Each GPU received a unique subset of the batch [51].
2. **Local Forward Pass**: Each GPU performs a forward pass using its local replica of the model and its assigned data subset. It calculated the loss based on its local predictions [51].
3. **Local Backward Pass**: Subsequently, each GPU computed gradients for its local model replica based on its local loss.
4. **Gradient Synchronization (All-Reduce)**: After local gradients are computed, we synchronize them across all GPUs. This ensures that every model replica had identical, averaged gradients [51].
5. **Optimizer Step**: Finally, each GPU performs an optimizer step using the synchronized (averaged) gradients to update its local model weights.

3.4.1.3 `DDPManager` Implementation

We centralized the setup and teardown of the distributed environment using our `DDPManager` class.

1. **Initialization**: We initialized `dist.init_process_group` using the `nccl` backend (optimized for NVIDIA GPUs) and set a timeout to prevent indefinite hangs [39].

2. **Rank and World Size**: Each process was assigned a unique rank and knew the total number of processes (`world_size`), which were crucial for coordinated operations.
3. **Device Assignment**: We explicitly assigned each DDP process to a specific GPU (`torch.cuda.set_device`), ensuring that each GPU was utilized by a single process.
4. **Graceful Fallback**: We included a robust mechanism to fall back to single-GPU training if DDP initialization failed, preventing the entire script from crashing.
5. **Cleanup**: We ensured that the distributed process group was properly destroyed (`dist.destroy_process_group`) at the end of training, releasing resources.

3.4.2 Gradient Accumulation Gradient accumulation allows simulating larger batch sizes than what can fit into GPU memory by accumulating gradients over several mini-batches before performing an optimizer step. For a global batch size B_{global} and per-GPU batch size B_{local}, if K is the accumulation steps ($K = \frac{B_{global}}{B_{local}}$), then gradients ∇L_k from each mini-batch k are accumulated [20]:

$$G = \sum_{k=1}^{K} \nabla L_k \tag{4}$$

Only after K steps is the optimizer step performed using the aggregated gradient G.

3.4.3 Synchronized Operations (`dist.barrier`): In distributed training, it's critical to ensure that all processes are synchronized at certain points (e.g., after loading a checkpoint, before starting an epoch, or before cleanup). In addition, we implemented a `safe_barrier` utility that wraps dist.barrier with error handling and a timeout. This prevents individual processes from hanging indefinitely if another process encounters an issue, contributing to the robustness of the distributed pipeline. Operations like logging, checkpoint saving, and final visualizations are restricted to the `is_main_process()` (rank 0), preventing redundant operations and ensuring consistent output.

3.4.4 Automatic Mixed Precision (AMP) We implemented Automatic Mixed Precision (AMP), a technique that combines `float32` and `float16` (or `bfloat16`) numerical precisions during training to accelerate training and reduce memory usage. We used `torch.cuda.amp.autocast` to automatically cast operations to the appropriate data type for maximum performance and `torch.cuda.amp.GradScaler` to scale the loss before the backward pass [4], which prevents vanishing gradients that can occur with `float16` due to underflow.

3.5 Loss Function

We implement a `CombinedLoss` class, an `nn.Module`, that incorporates multiple loss terms, tailored for this image reconstruction task. In addition, we utilize MAE and SSIM (discussed later) also as evaluation metrics.

3.5.1 Mean Absolute Error (MAE) Mean Absolute Error (MAE), or L1 Loss, is a fundamental regression loss function that calculates the average of the absolute differences between predicted and actual values. For predictions $\hat{y}_i$ and targets y_i over N samples, the MAE is calculated as:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{5}$$

In FWI, where seismic data can sometimes contain anomalous noise or extreme values, MAE can provide a more stable gradient for optimization. It encourages the model to produce predictions that are close to the median of the target values, which can be desirable for velocity models.

3.5.2 Huber Loss Huber loss, introduced by Huber (1964), is a robust loss function that combines the best properties of Mean Squared Error (MSE) and Mean Absolute Error (MAE). For an error $e = (y - \hat{y})$, where y is the target and $\hat{y}$ is the prediction, and a hyperparameter δ, the Huber loss ($\mathcal{L}_{\text{Huber}}(e)$) is defined as [14]:

$$\mathcal{L}_{\text{Huber}}(e) = \begin{cases} \frac{1}{2}e^2, & \text{if } |e| \leq \delta \\ \delta(|e| - \frac{1}{2}\delta), & \text{if } |e| > \delta \end{cases} \tag{6}$$

This makes it less sensitive to outliers or noisy data compared to pure MSE, which is important in real-world seismic data that can contain significant noise or anomalies.

3.5.3 Gradient Loss Gradient Loss penalizes differences in the spatial gradients between the predicted and target velocity models. It often uses filters like Sobel operators to compute these gradients, which are small, separable convolution kernels used for edge detection. They approximate the image gradient magnitude and direction by computing intensity difference across a small neighborhood of pixels [49]. The horizontal Sobel filter S_x and vertical Sobel filter S_y are typically defined as:

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \quad S_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \tag{7}$$

These filters are convolved with the image to approximate its gradients in the x and y directions. Let $\nabla_x \hat{Y}$ and $\nabla_y \hat{Y}$ be the horizontal and vertical gradients of the prediction $\hat{Y}$, and similarly, $\nabla_x Y$ and $\nabla_y Y$ for the target Y [49]. The gradient loss $\mathcal{L}_{\text{Grad}}$ is typically computed as the sum of L1 differences:

$$\mathcal{L}_{\text{Grad}} = \left\| \nabla_x \hat{Y} - \nabla_x Y \right\|_1 + \left\| \nabla_y \hat{Y} - \nabla_y Y \right\|_1 \tag{8}$$

By matching gradients, the model is encouraged to preserve sharp edges and interfaces in the velocity model, which are crucial for accurately representing geological structures like faults or layer boundaries.

3.5.4 Total Variation Loss Total variation (TV) loss, a concept popularized by Rudin, Osher, & Fatemi (1992) for noise removal, is a regularization term that penalizes large variations between neighboring pixels. For a 2D image $\hat{Y}$, the discrete TV loss, $\mathcal{L}_{\text{TV}}$, is defined as [44]:

$$\mathcal{L}_{\text{TV}} = \sum_{i,j} \left(\left| \hat{Y}_{i+1,j} - \hat{Y}_{i,j} \right| + \left| \hat{Y}_{i,j+1} - \hat{Y}_{i,j} \right| \right) \tag{9}$$

TV loss promotes spatial smoothness in the predicted output, helping to reduce noise and artifacts.

3.5.5 Per Pixel Loss Calculation and Class Weighting The overall combined loss $\mathcal{L}_{\text{Combined}}$ is a weighted sum of the aforementioned components, defined as:

$$\mathcal{L}_{\text{Combined}} = w_{\text{Huber}} \cdot \mathcal{L}_{\text{Huber}}(\hat{Y}, Y) + w_{\text{Grad}} \cdot \mathcal{L}_{\text{Grad}}(\hat{Y}, Y) + w_{\text{TV}} \cdot \mathcal{L}_{\text{TV}}(\hat{Y}, Y) \tag{10}$$

where, w_{Huber}, w_{Grad}, and w_{TV} are the respective weights from the configuration (`HUBER_LOSS_WEIGHT`, `GRADIENT_LOSS_WEIGHT`, `TV_LOSS_WEIGHT`).

3.5.5.1 Inverse Frequency Weighting

This technique addresses class imbalance by assigning higher weights to samples from less frequent classes and lower weights to samples from more frequent classes. This ensures each class contributes more equally to the total loss, preventing bias towards majority classes [62]. The weight for class c is :

$$W_{\text{c}} = \frac{N_{\text{total}}}{N_{\text{classes}} \cdot N_{\text{c}}} \tag{11}$$

where N_{c} is the count of samples in class c, N_{total} is the total number of samples across all unique classes, and N_{classes} is the total number of unique classes.

3.6 Learning Rate Scheduling

The learning rate (LR) schedule dictates how the learning rate changes over the course of training. A well-designed schedule can significantly impact convergence speed and final model performance.

3.6.1 Warmup (`LinearLR`) Warmup gradually increases the learning rate from a very small value to the initial base learning rate over a few epochs, which stabilizes the training process and prevents early divergence, We use PyTorch's `LinearLR` for this linear increase [38].

3.6.2 Cosine Annealing with Warm Restarts After the warmup phase, the scheduler employs `CosineAnnealingWarmRestarts`, a method proposed by Loshchilov & Hutter (2017). This scheduler causes the learning rate η_t to decay following a cosine curve, periodically "restarting" to the initial learning rate (or a slightly lower maximum) [28]. The learning rate at iteration t within cycle i is given by:

$$\eta_t = \eta_{\min}^{(i)} + \frac{1}{2}\left(\eta_{\max}^{(i)} - \eta_{\min}^{(i)}\right)\left(1 + \cos\left(\frac{T_{\text{cur}}}{T_i}\pi\right)\right) \tag{12}$$

where,

- T_{cur} is the number of epochs since the last restart
- T_i is the total number of epochs in the current cycle
- $\eta_{\min}^{(i)}$ and $\eta_{\max}^{(i)}$ are the minimum and maximum learning rates for cycle i [28, 37].

A dedicated `WarmRestartMonitor` class is implemented to track the learning rate and detect when a warm restart occurs.

3.7 Model Regularization and Generalization

Regularization techniques are crucial for preventing overfitting and improving the model's ability to generalize to unseen data. We implement the following tools and techniques in our project:

3.7.1 Weight Decay (L2 Regularization) Applied through the `AdamW` optimizer (discussed later), weight decay adds a penalty to the loss function proportional to the square of the model's weights. For a given set of model weights W, the L2 regularization term added to the loss $\mathcal{L}$ is:

$$\mathcal{L}_{\text{total}} = \mathcal{L} + \lambda \sum_{w \in W} w^2 \tag{13}$$

where λ is the weight decay coefficient. This discourages large weights, effectively simplifying the model and reducing its tendency to overfit [27].

3.7.2 Gradient Clipping Gradient clipping, a technique highlighted by Pascanu et al. (2013) for training recurrent neural networks, limits the magnitude of gradients during the backward passv[32]. If the L2 norm of the gradients $\|\mathbf{g}\|_2$ for a parameter group $\mathbf{g}$ exceeds a certain threshold C (`GRAD_CLIP_NORM`), the gradients are scaled down by a factor of $\frac{C}{\|\mathbf{g}\|_2}$ [32]:

$$\mathbf{g} \leftarrow \mathbf{g} \cdot \frac{C}{\|\mathbf{g}\|_2}, \text{ if } \|\mathbf{g}\|_2 > C \tag{14}$$

This technique is particularly important when dealing with recurrent neural networks or very deep networks, or when using mixed precision, as it helps prevent exploding gradients, which can lead to unstable training and NaN values.

3.7.3 Exponential Moving Average (EMA) We implement a class `ModelEMA` which maintains a separate set of model weights that are an EMA of the training model's weights. This concept, often associated with Polyak-Ruppert averaging (Polyak & Juditsky, 1992), involves updating the EMA weights θ_{EMA} smoothly at each step based on the current training weights θ [34]:

$$\theta_{\text{EMA}} \leftarrow \text{decay} \cdot \theta_{\text{EMA}} + (1 - \text{decay}) \cdot \theta \tag{15}$$

where decay is the EMA decay rate. EMA models often exhibit better generalization performance and provide more stable predictions than the final weights of the training model, especially in the presence of noisy gradients or fluctuating loss landscapes.

3.7.4 Dropouts in Decoder Dropout, introduced by Srivastava et al. (2014), randomly sets a fraction p of input features to zero during training (and scales the remaining activations by $1/(1-p)$ at training time or uses p at inference) [50]. This prevents co-adaptation of neurons and forces the network to learn more robust features, therefore reducing overfitting. A lower dropout layer is used in the final refinement layer, as it's closer to the output and less prone to severe overfitting.

3.7.5 Optimization and Compiler Techniques Beyond the standard optimizer, we incorporate several low-level optimizations to squeeze out maximum performance.

3.7.5.1 Primary Optimizer

`AdamW` proposed by Loshchilov & Hutter (2019), is a variant of the popular `Adam` optimizer that decouples weight decay from the adaptive learning rate mechanism. In standard `Adam`, L2 regularization (weight decay) is applied directly to

the gradients, which can interact unfavorably with adaptive learning rates, especially for parameters with small adaptive learning rates [27]. `AdamW` addresses this by applying weight decay directly to the weights, independently of the adaptive learning rates. The weight update rule for `AdamW` is:

$$w_t \leftarrow w_{t-1} - \eta_t \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda w_{t-1} \right) \tag{16}$$

where,

- w_t are weights at time t
- η_t is the learning rate
- $\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected first and second moment estimates of the gradients (as in `Adam`)
- ϵ is a small constant for numerical stability
- λ is the weight decay coefficient [27].

We utilize `AdamW` as our primary optimizer implementing it via the PyTorch module `torch.optim.AdamW` and leveraging its improved regularization properties and efficiency.

3.7.5.2 Fused Optimizers

Fused optimizers combine multiple operations (e.g., gradient computation, weight update, and L2 regularization) into a single, highly optimized kerne, which reduces kernel launch overheads and improves cache utilization, leading to faster training steps. We checked for the availability of `fused=True` in `torch.optim.AdamW` and used it if supported.

3.7.5.3 Channels-Last Memory Format

PyTorch tensors typically use a channels-first memory format (e.g., `[B, C, H, W]`). `torch.channels_last` (e.g., `[B, H, W, C]`) can sometimes offer performance benefits on certain hardware (particularly NVIDIA GPUs with Tensor Cores) by aligning memory access patterns more efficiently for convolutional neural networks [36]. Hence, we explicitly convert input tensors to `channels_last` format at the beginning of the training and validation loops.

3.7.5.4 Flash Attention

FlashAttention, proposed by Dao et al. (2022), is an optimized attention algorithm that speeds up and reduces memory usage for attention mechanisms, especially on modern GPUs [5]. In our script, we enabled `torch.backends.cuda.enable_flash_sdp(True)` if a CUDA-capable GPU is available. This allows the Swin Transformer backbone to potentially leverage FlashAttention for its self-attention computations, leading to further performance gains.

3.8 High Performance Computing (HPC) Infrastructure and Management

Leveraging High-Performance Computing (HPC) environments is crucial for training complex deep learning models. This section details how we utilized an HPC infrastructure and job management system to conduct our experiment.

3.8.1 Grid5000 HPC Environment We conducted the training within the Grid5000 distributed computing infrastructure, a large-scale experimental platform for research in parallel and distributed computing [11].

The experiments specifically utilized the `musa` nodes in the Sophia cluster in Grid5000. These nodes are equipped with powerful 2x NVIDIA H100 GPUs per node. The H100 GPUs, based on the NVIDIA Hopper architecture, offer substantial advancements in AI performance, including fourth-generation Tensor Cores for accelerated mixed-precision calculations and Transformer Engine for efficient processing of large language models (LLMs).

3.8.2 OAR Job Scheduler OAR (Open Advanced Resource Reservation) is an open-source batch scheduler and resource manager widely used in research and academic HPC clusters, including Grid5000.

3.8.2.1 How OAR Works

Users submit job scripts (e.g., Shell scripts, Python scripts) to OAR, specifying their resource requirements (e.g., number of nodes, number of GPUs per node, CPU cores, memory, wall-time limit, etc.). OAR then places the job in a queue and, based on priority, resource availability, and scheduling policies, allocates the requested resources. Once resources are allocated, OAR launches the job on the designated nodes. For DDP training, OAR ensures that the specified number of processes are launched across the allocated GPUs, and it manages the environment variables necessary for PyTorch DDP to discover and communicate between processes.

3.9 Training Workflow and Experiment Management

A well-structured training workflow and robust experiment management practices are crucial for conducting reproducible research and developing reliable deep learning models.

3.9.1 Reproducibility Ensuring that experiments can be replicated is a cornerstone of scientific research. We implemented several measures to achieve high levels of reproducibility:

1. **Setting Random Seeds**: We explicitly seeded all sources of randomness: `random.seed()` (for Python's build in `random` module), `np.random.seed()`

(for NumPy operations), `torch.manual_seed()` (for CPU-based PyTorch operations), and `torch.cuda.manual_seed_all()` (for GPU-based PyTorch operations) [40]. By setting a fixed RANDOM_SEED in the configuration, the initial conditions for all random processes are controlled.

2. **Saving and Loading RNG States in Checkpoints**: To enable seamless resumption of training from a checkpoint while maintaining exact reproducibility, we saved the random number generators (RNG) states of Python, NumPy, and PyTorch (CPU and GPU) as part of the checkpoint. When a checkpoint is loaded, these states are restored, ensuring that the subsequent random operations continue from precisely where they left off.
3. **Seeding DataLoader Workers**: The `worker_init_fn` function passed to DataLoader sets the random seeds for NumPy and Python's random module within each worker, ensuring that data augmentations are applied consistently across workers for a given epoch but vary appropriately between epochs.

3.9.2 Checkpointing and Resumption We employed the following checkpointing and resumping capabilities:

1. **Atomic Checkpoint Saves**: Checkpoints are saved to a temporary file (`.tmp`) first and then atomically renamed to their final destination. This practice prevents the creation of corrupted checkpoint files if the saving process is interrupted (e.g., by a power outage or system crash).
2. **Loading Checkpoints**: The `load_checkpoint` function is responsible for restoring the entire training states (model weights, optimizer state, scheduler state, EMA model state, and RNG states).
3. **Resuming Training from a Specific Epoch**: By loading the epoch number from the checkpoint, the training loop can correctly resume from the epoch immediately following the last saved epoch, rather than starting from scratch.

3.9.3 Early Stopping Early stopping is a regularization technique that prevents overfitting by halting the training process when the model's performance on a validation set stops improving. The `EarlyStopping` class continuously monitors a chosen validation metric (in this case, the average validation Mean Absolute Error, MAE). The `PATIENCE` parameter defines how many epochs the model can go without improvement before training is halted. This prevents premature stopping due to minor fluctuations in the validation metric.

3.10 Logging and Monitoring

Comprehensive logging and real-time monitoring are essential for tracking training progress, debugging issues, and understanding model behavior.We

used Python's standard logging module to output messages to both the console (`sys.stdout`) and a dedicated log file.

3.11 Model Evaluation and Deployment Readiness

After a deep learning model has been trained, its performance must be evaluated to ensure it meets the desired criteria before being deployed into the real world.

3.11.1 Evaluation Metrics

3.11.1.1 MAE

As discussed in Section 3.5, MAE served as a key evaluation metric, consistent with the Kaggle competition's leaderboard criteria, and was used in conjunction with our `CombinedLoss` function.

3.11.1.1 Structural Similarity Index Measure (SSIM)

SSIM, introduced by Wang et al. (2004), is a perceptual metric that quantifies the similarity between two images. SSIM considers image composition, including luminance, contrast, and structural information [59]. For two non-negative images x and y, SSIM is defined as:

$$\mathrm{SSIM}(x, y) = [l(x, y)]^{\alpha} \cdot [c(x, y)]^{\beta} \cdot [s(x, y)]^{\gamma} \tag{17}$$

where $l(x, y)$, $c(x, y)$ and $s(x, y)$ are the luminance, contrast and structure comparison functions, respectively, and α, β, and γ are the weights (often set to 1) [59]. The most common form is when $\alpha = \beta = \gamma = 1$:

$$\mathrm{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{18}$$

where μ_x, μ_y are the means, σ_x, σ_y are the standard deviations, and σ_{xy} is the cross-covariance of the images x and y. C_1 and C_2 are small constants to avoid division by zero [59]. It typically operates on images scaled to a `[0,1]` range.

3.11.2 Model Export for Inference Once a model is trained and validated, it often needs to be deployed for inference in various environments. Exporting the model to a standardized format facilitates this process.

3.11.2.1 ONNX Export

ONNX (Open Neural Network Exchange) is an open standard that defines a common set of operators and a file format for representing deep learning models. It acts as an intermediate representation, allowing models trained in one

framework (like PyTorch) to be easily converted and run in another (e.g., TensorFlow, ONNX Runtime). The ONNX standard was initiated by companies like Microsoft and Facebook (2017). Hence, we export the final trained EMA model to the ONNX format, making it ready for deployment in production or integration into other systems.

4 Results

4.1 Experiment Configuration

The experiment began with a successful validation of DDP communication, confirming a robust multi-GPU setup utilizing the 2x H100 GPU architecture in the `musa` cluster of Grid5000 HPC environment.

4.2 Dataset Distribution

We meticulously split the dataset into training and validation sets, maintaining the proportions of each data group:

1. Training Set: A total of 376,500 samples, comprising 173,000 "Fault" samples, 107,500 "Style" samples, and 96,000 "Vel" samples.
2. Validation Set: A total of 93,500 samples, including 43,000 "Fault" samples, 26,500 "Style" samples, and 24,000 "Vel" samples.
3. The script calculated specific class weights to balance the loss contributions from different prediction tasks: "Fault" received a weight of 0.5549, "Vel" 1.0, and "Style" 0.8930.

We configured the DataLoaders to use `NUM_WORKERS = 48` per process, resulting in 735 training batches per GPU. This high number of workers was determined based on the `os.cpu_count()` of the compute node and ensures efficient data loading, preventing bottlenecks during training.

4.3 Model Architecture

The model combines a powerful feature extractor with an advanced decoder for precise predictions.

1. **Swin Transformer V2 Encoder**: The SwinV2 backbone was initialized with pretrained weights from `timm/swinv2_tiny_window8_256.ms_in1k` imported from PyTorch's `timm` library.
2. **Enhanced U-Net Decoder**: The decoder alone accounted for 47,520,064 parameters, representing 63.3% of the total model parameters. Its enhancements included learned upsampling, residual blocks and SCSE attention blocks—as discussed in the previous section of this report.

4.4 Loss Function and Class Weighting

As discussed, we utilized a CombinedLoss function, designed to optimize multiple aspects of the prediction: Huber Loss with `HUBER_LOSS_WEIGHT = 0.4`, Gradient Loss with `GRADIENT_LOSS_WEIGHT = 0.55`, Total Variation (TV) Loss with `TV_LOSS_WEIGHT = 0.05`.

The class weights applied to the loss (Fault: `0.5549`, Vel: `1.0`, Style: `0.8930`) address dataset imbalances, ensuring that the model learns effectively from all categories of samples.

4.5 Learning Rate Schedule

We used `CosineAnnealingWarmRestarts`, a sophisticated learning rate scheduler, which included the following schedule:

1. A warmup phase of 3,675 steps, where the learning rate gradually increased from a starting factor of 0.01 to the initial learning rate of 1.5×10^{-4}
2. An initial restart period of 5880 steps (equivalent to 8 epochs), with $T_{mult} = 2$. This means the learning rate periodically drops to a minimum ($\eta_{min} = 5 \times 10^{-7}$) and then gradually increases, allowing the model to escape local minima and explore the solution space more thoroughly.
3. The total training comprised 36,750 steps, and we observed two expected restarts at steps 9,555 and 21,315 each.

4.6 Training Strategy

The training strategy incorporated several advanced techniques to enhance performance and stability:

- **Automatic Mixed Precision (AMP)**: We utilized AMP with `torch.cuda.amp.autocast` and `GradScaler`, which leverages lower precision (e.g., `bfloat16` or `float16`) for certain operations.
- **`AdamW` Optimizer**: `AdamW` was used to update model parameters. We used `WEIGHT_DECAY = 1e-4`.
- **Exponential Moving Average (EMA)**: We maintained an EMA models with `EMA_DECAY = 0.99`.
- **Gradient Clipping**: We applied gradient clipping with a maximum `GRAD_CLIP_NORM = 1.0` to prevent exploding gradients, which can destabilize training.
- **Adaptive Data Augmentation**: We utilized adaptive augmentation, which decayed progressively based on validation performance, allowing the model to learn from diverse data early on and then refine its understanding on less augmented data later.

– **Early Stopping**: We employed the EarlyStopping mechanism with a patience of 5 epochs prevented overfitting.

4.7 Training Performance

The model trained for 34 out of 50 planned epochs before early stopping triggered. The total training duration was approximately 18.23 hours using efficient training technologues like DDP and AMP in 2x H100 GPUs. The best validation MAE was achieved at Epoch 29 with an overall (average) validation MAE of 63.27 m/s:

Table 2: Best EMA model

Metric	Loss	MAE	Grad Norm	SSIM
Train (Avg)	0.0375	66.69	0.08	-
Val "Vel" Group	0.0141	43.07	-	0.8725
Val "Style" Group	0.0232	69.82	-	0.9188
Val "Fault" Group	0.0253	76.94	-	0.8844

Table 1 shows that our average training combined loss was approximately 0.038, with an average validation combined loss of 0.021, indicating effective training. A low gradient norm of 0.08 further confirms training stability. The high SSIM metrics suggest excellent visual and perceptual fidelity of the predicted velocity models. Notably, the 'Fault' group exhibited the highest combined loss and MAE among the validation sets.

4.7.1 Key Performance Metrics Over Epochs As noted, the model demonstrated consistent improvement across all metrics, with the overall validation Mean Absolute Error (MAE) serving as the primary performance indicator.

Table 3: Model training metrics across epochs

Metric	Epoch 1	Epoch 5	Epoch 10	Epoch 13	Epoch 20	Epoch 29	Epoch 34
Train Loss	0.1154	0.0659	0.0508	0.0462	0.0446	0.0375	0.0418
Train MAE	439.84	157.19	102.60	88.68	84.61	66.69	77.88
Overall Val MAE	308.34	126.90	85.95	79.90	70.42	**63.27**	64.97
Val Vel MAE	398.78	134.28	70.91	61.36	51.97	43.07	47.37
Val Style MAE	191.19	105.33	84.34	81.46	73.99	69.82	69.50
Val Fault MAE	335.04	141.08	102.61	96.88	85.29	76.94	78.05
Val Vel SSIM	0.4973	0.7426	0.8196	0.8311	0.8583	0.8725	0.8714
Val Style SSIM	0.7389	0.8542	0.8919	0.8966	0.9116	0.9188	0.9214
Val Fault SSIM	0.6343	0.8011	0.8500	0.8556	0.8756	0.8844	0.8864

As seen in the above table, the model achieved its best overall validation MAE of 63.27 at Epoch 29. This significant reduction from an initial MAE of 308.34 in Epoch 1 demonstrates the effectiveness of the combined loss function, robust architecture, and comprehensive augmentation strategy. Furthermore, we observed a consistent reduction in MAE across all three prediction tasks (Vel, Style, and Fault) and a corresponding increase in SSIM, indicating improved structural similarity between predictions and ground truth.

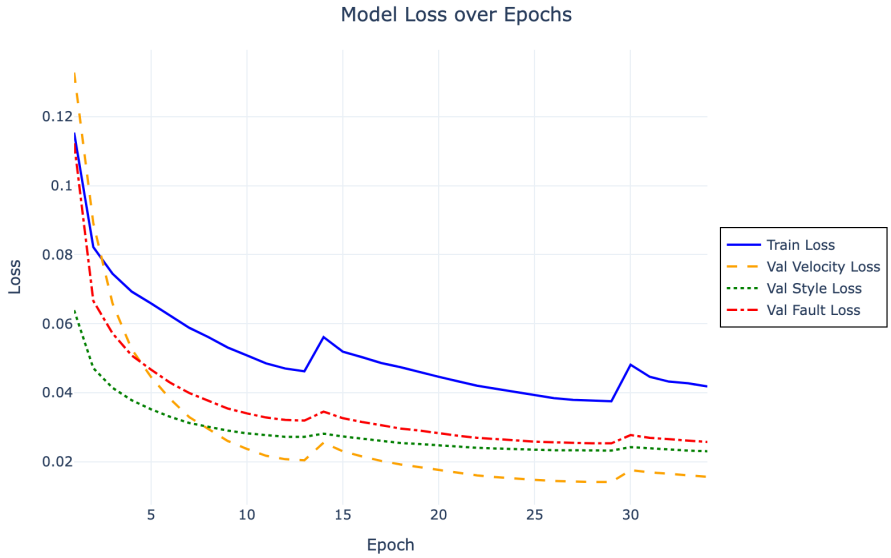

Figure 2: Combined Loss (0.4 Huber + 0.55 Gradient + 0.05 Total Variation)

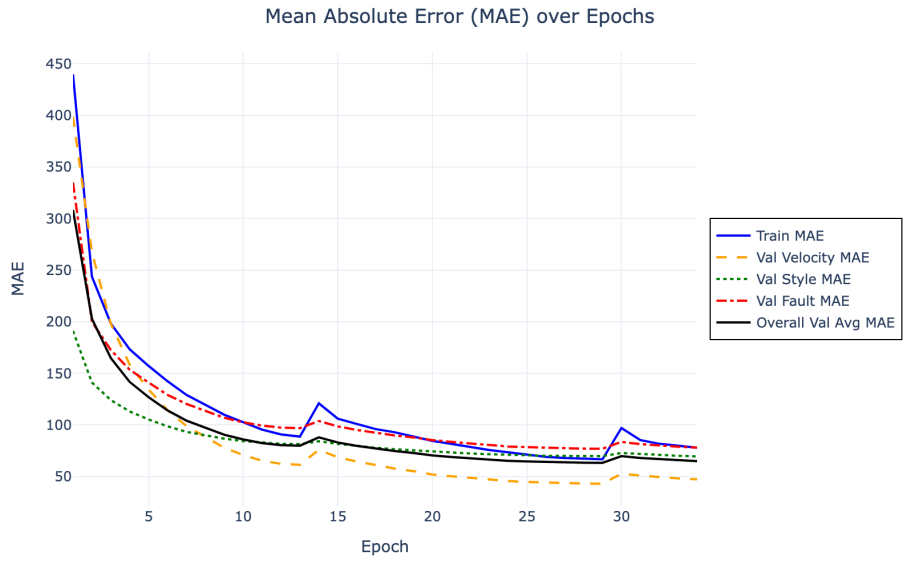

Figure 3: Mean Absolute Error

The learning rate warm restarts were detected at step 9,554 (Epoch 13) and step 21,314 (Epoch 29), which seems to have played a crucial role in escaping a local minimum and improving the optimization process in general. At these points, the learning rat reset from 5×10^{-7} to 1.5×10^{-4}, allowing the model to explore new regions of the loss landscape, which likely contributed to the continued improvement in MAE observed after these epochs. For instance, the MAE continued to drop from 79.90 at Epoch 13 to 63.27 (the best MAE yet) at Epoch 29.

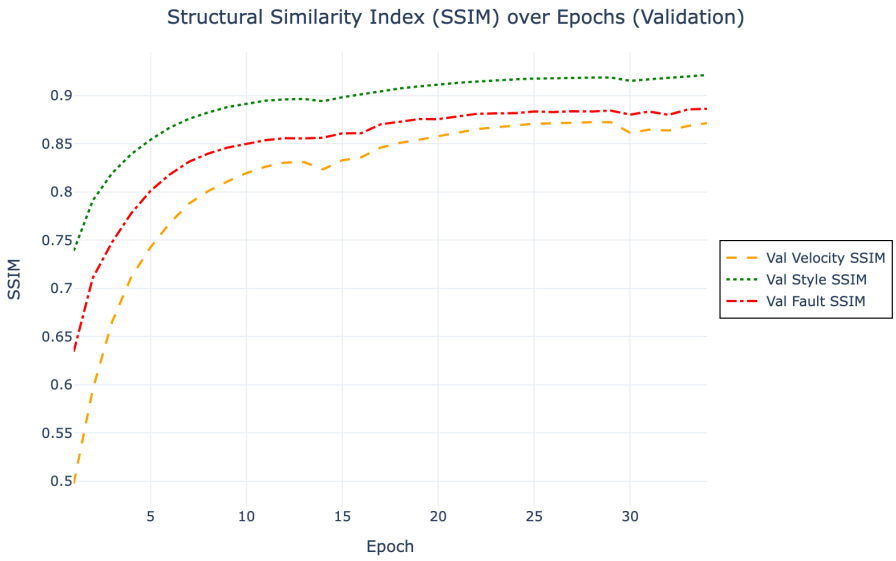

Figure 4: Structural Similarity

Early stopping halted the training at Epoch 34 after five consecutive epochs without improvement in the overall validation MAE. This indicates that the model had converged to its optimal performance given the current configuration and prevented further overfitting.

4.7.8 Predictions Analysis The `visualize_predictions` prepares a qualitative assessment of the model's performance on various validation samples across the three data groups ("Vel", "Style", and "Fault"). The visualization is a 9x3 array of images. Each row displays the "Ground Truth" (actual velocity model), "Prediction" generated by the model, and the "Absolute Difference" between the two. The color bar for the "Absolute Difference" indicates the magnitude of error, with lighter colors (yellow/orange) representing higher errors/differences and darker colors (black/moroon) representing lower errors/differences.

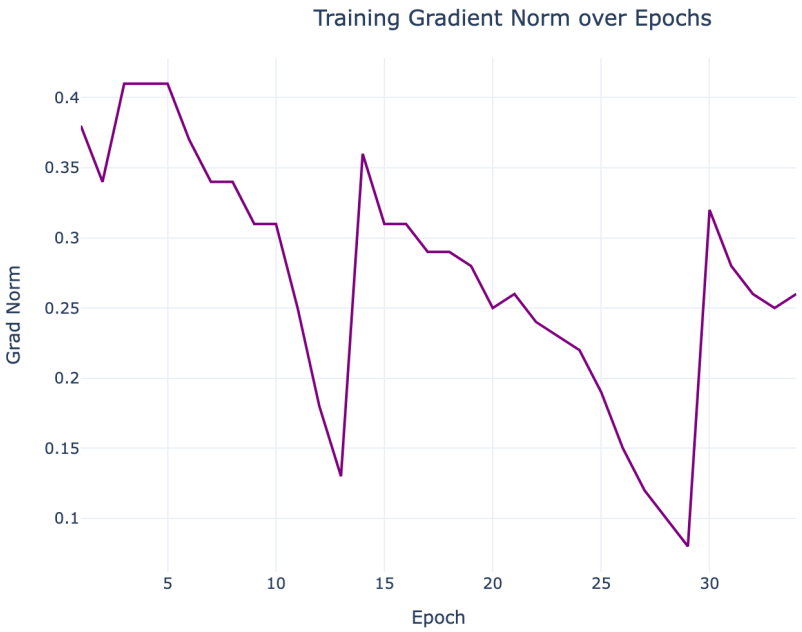

Figure 5: Gradient Normalization

4.7.8.1 General Observations

Upon review of the visualization, noted the following:

1. **High Fidelity Predictions**: Across all displayed samples and groups, the predicted images closely resemble their corresponding actual counterparts. This qualitative visual similarity is evidenced by the strong quantitative MAE and SSIM metrics, indicating that the model effectively captures the overall structure and value distributions of the target properties.

2. **Sharpness and Detail Preservation**: The model appears to maintain a good level of sharpness and detail in its predictions, especially for complex geological structures. This is likely a direct benefit of using the Swin Transformer backbone as well as the stronger-weighted gradient loss component in the combined loss function that encourages the preservation of edges and fine features.

3. **Effective Generalization**: The diversity of the samples shown (ranging from layered structures to more complex, chaotic patterns) suggests that the model has done a good job in generalizing the various geological scenarios within the dataset.

4.7.8.2 Group-Specific Performance

1. **“Vel” Predictions**: The “Vel” samples show excellent agreement between ground truth and prediction. The layered structures are accurately reproduced, and the transitions between different velocity zones are smooth and well-defined. The “Absolute Difference” maps for “Vel” samples show generally low error values, with small, localized areas of higher difference. These higher differences often appear at the interfaces between distinct velocity layers, which are inherently challenging areas for any model to perfectly reconstruct. This is a common characteristic even for high-performing models.

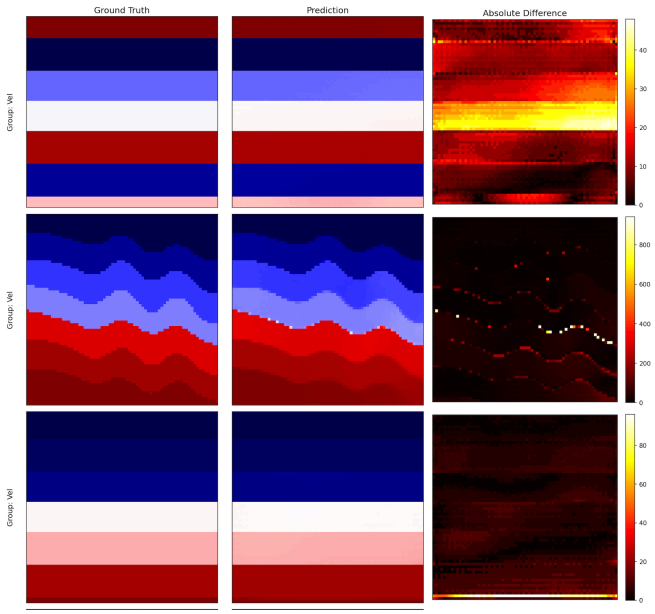

Figure 6: ”Vel” group sample predictions based on the best ema model.

2. **“Style” Predictions**: The “Style” samples, which appear to represent more abstract or textural properties, are also well-predicted. The model captures the overall “flow” and patterns, even in intricate designs. The “Absolute Difference” for “Style” samples indicates low errors across most of the image. Any notable differences tend to be in areas with rapid changes in texture or complex, fine-scale variations, where precise pixel-to-pixel matching can be difficult.

3. **“Fault” Predictions**: The “Fault” samples demonstrate the model’s ability to delineate complex fault structures. The predictions successfully capture the presence and general orientation of the faults, which are critical for geological interpretation. The “Absolute Difference” maps for “Fault” samples show slightly higher, more distributed errors compared to “Vel” and “Style”

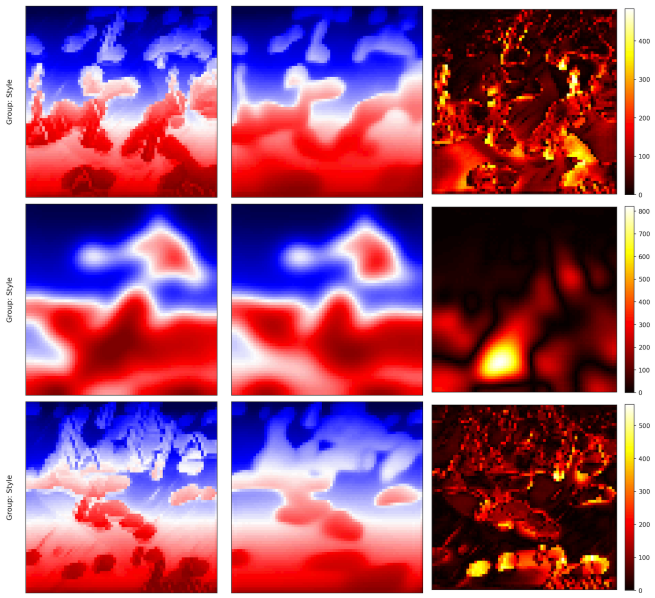

Figure 7: "Style" group sample predictions based on the best ema model.

samples, particularly along the fault lines themselves. This is expected, as faults represent sharp discontinuities, and even minor misalignments or blurring in the prediction can result in noticeable differences. However, the overall structure of the faults is still clearly visible in the predictions, indicating strong performance in this challenging task. The combined loss function, with its emphasis on gradients, is likely key to this performance.

4.7.8.3 Interpretation of Absolute Difference Maps

The "Absolute Difference" column provides valuable insights into where the model's predictions deviate most from the ground truth.

1. Low Overall Error: The prevalence of darker colors (black/maroon) in the difference maps suggests that the model's predictions are, on average, very close to the true values.
2. Errors at Boundaries/Complexities: Hotter spots (yellow/orange) in the difference maps are predominantly located at:
   - Sharp interfaces between different regions (e.g., velocity layers, fault lines)
   - Areas with highly complex or fine-grained features that might be difficult to perfectly reproduce This is a typical behavior for image reconstruction models; achieving perfect reconstruction at every single pixel, especially at boundaries, is extremely challenging

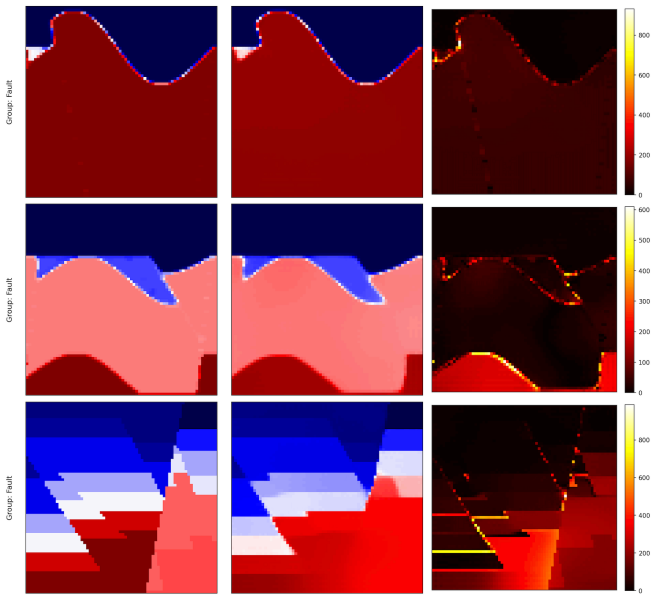

Figure 8: "Fault" group sample predictions based on the best ema model.

3. Validation of Loss Components: The visual quality, particularly the preservation of structural details and the relatively low errors at boundaries, implicitly validates the effectiveness of the Gradient Loss and Total Variation Loss components in the CombinedLoss function.

In conclusion, the prediction visualizations support the quantitative metrics we observed during training, evidencing that the model produces highly accurate and visually consistent results across different types of geophysical data. The observed differences are largely concentrated in areas of high complexity or sharp transitions, which are inherent challenges in this domain.

5 Conclusion

In this paper, we presented a robust and effective deep learning framework for FWI. Our hybrid CNN-Transformer architecture, which we would like to call Swin-U-Net, leverages the Swin V2 transformer backbone and an advanced U-Net-style decoder. The model exhibited performance in reconstructing subsurface velocity models from seismic data.

We conducted a comprehensive evaluation on the OpenFWI benchmark dataset, which included the "Vel," "Style," and "Fault" data families. We achieved a low overall MAE of 63.27 m/s and high SSIM values (80+%) across diverse geological structures, indicating both high accuracy and high fidelity. The combined loss function, which included Huber, Gradient, and Total Variation terms, along with

inverse frequency class weighting, proved instrumental in optimizing for both overall accuracy and the preservation of crucial geological features like sharp boundaries and faults.

The successful implementation of advanced training techniques such as Distributed Data Parallel (DDP), Automatic Mixed Precision (AMP), gradient accumulation, and a sophisticated learning rate schedule with warm restarts enhanced the training efficiency and model stability. Our approach to data augmentation further improved the model's generalization capabilities, making it more robust to unseen data.

Overall, our work demonstrated the significant potential of deep learning, particularly with Transformer-based architectures, to address the long-standing challenges of FWI, such as computational cost, ill-posedness, and cycle skipping. The high-fidelity predictions and robust generalization capabilities underscore the model's ability to learn complex wave physics and effectively translate seismic measurements into meaningful subsurface images.

6 Future Work

While our current model showed promising results, several avenues for future research and improvement exist. There are still opportunities to explore with more advanced architectures such as other Transformer variants, multiscale attention or incorporating 3D convolutions for processing volumetric seismic data. Additionally, integrating concepts from PINNs more deeply into the architecture, beyond just the loss function, could potentially lead to models that are not only data-driven but also inherently more physically consistent.

In addition, A significant limitation of many data-driven inversion methods, including ours, is the lack of explicit uncertainty quantification. Future work could focus on developing Bayesian deep learning approaches or ensemble methods to provide estimates of uncertainty alongside the predicted velocity models. This would be crucial for practical applications where confidence in the inversion results is paramount.

While our model is significantly faster at inference than conventional FWI, further optimization can still be made to reduce the training time of 30 minutes per epoch. This could involve exploring more aggressive model pruning, quantization, or specialized hardware acceleration (e.g., custom ASICs for inference). Also, although our model demonstrated strong generalization within the OpenFWI dataset families, its performance on entirely out-of-distribution or real-world field data remains an open question. Future work should focus on transfer learning strategies, fine-tuning techniques with limited real data, or meta-learning approaches to enable the model to adapt more effectively to new geological environments without extensive re-training.

References

[1] B. Artley. *OpenFWI Preprocessed 72x72*. Kaggle. 2025. URL: https://www.kaggle.com/datasets/brendanartley/openfwi-preprocessed-72x72.

[2] R. C. Aster, B. Borchers, and C. H. Thurber. *Parameter Estimation and Inverse Problems*. 3rd ed. Elsevier, 2018. URL: https://www.sciencedirect.com/book/9780128134238/parameter-estimation-and-inverse-problems.

[3] G. C. Biondi. *3-D Seismic Imaging*. Stanford, CA, USA: Society of Exploration Geophysicists, 2006. DOI: 10.1190/1.9781560801399.

[4] Michael Carilli. *Automatic Mixed Precision*. 2022. URL: https://docs.pytorch.org/tutorials/recipes/recipes/amp_recipe.html (visited on 07/30/2025).

[5] T. Dao et al. "FlashAttention: Fast and Memory-Efficient Attention". In: *Advances in Neural Information Processing Systems (NeurIPS)* (2022). URL: https://arxiv.org/abs/2205.14135.

[6] C. Deng et al. "OpenFWI: Large-Scale Multi-Structural Benchmark Datasets for Seismic Full Waveform Inversion". In: *arXiv preprint arXiv:2111.02306* (2021). URL: https://arxiv.org/abs/2111.02306.

[7] C. Deng et al. *OpenFWI: Large-Scale Multi-Structural Benchmark Datasets for Seismic Full Waveform Inversion*. ResearchGate. 2021. URL: https://www.researchgate.net/publication/355924871_OpenFWI_Large-Scale_Multi-Structural_Benchmark_Datasets_for_Seismic_Full_Waveform_Inversion.

[8] S. Dong et al. "Transformer for seismic image super-resolution". In: *arXiv preprint arXiv:2408.01695* (2024). URL: https://arxiv.org/abs/2408.01695.

[9] A. Dosovitskiy et al. "An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations (ICLR)*. 2020. URL: https://arxiv.org/abs/2010.11929.

[10] M. Goren and E. Treister. "Physics-guided Full Waveform Inversion using Encoder-Solver Convolutional Neural Networks". In: *arXiv preprint arXiv:2405.17696* (2024). URL: https://arxiv.org/abs/2405.17696.

[11] Grid5000. *Official Website*. https://www.grid5000.fr/. Accessed: July 2025.

[12] R. Harsuko and T. Alkhalifah. "Optimizing a Transformer-based network for a deep learning seismic processing workflow". In: *arXiv preprint arXiv:2308.04739* (2023). URL: https://arxiv.org/abs/2308.04739.

[13] J. Hu, L. Shen, and G. Sun. "Squeeze-and-Excitation Networks". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018. URL: https://arxiv.org/abs/1709.01507.

[14] P. J. Huber. "Robust Estimation of a Location Parameter". In: *Annals of Statistics* 35.1 (1964), pp. 73–101. URL: https://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-35/issue-1/Robust-Estimation-of-a-Location-Parameter/10.1214/aoms/1177703732.full.

[15] P. Jin et al. "OpenFWI: Large-Scale Multi-Structural Benchmark Datasets for Seismic Full Waveform Inversion". In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round*

1). 2021. URL: https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/182be0c5cdcd5072bb1864cdee4d3d6e-Abstract-round1.html.

[16] P. Jin et al. *OpenFWI: Large-Scale Multi-Structural Benchmark Datasets for Seismic Full Waveform Inversion*. Papers with Code. 2021. URL: https://paperswithcode.com/dataset/openfwi.

[17] P. Jin et al. "An empirical study of large-scale data-driven full waveform inversion". In: *Scientific Reports* 14.1 (2024), p. 19758. DOI: 10.1038/s41598-024-70678-7.

[18] Kaggle. *Yale/UNC-CH - Geophysical Waveform Inversion*. https://www.kaggle.com/competitions/waveform-inversion. Accessed: July 26, 2025. 2025.

[19] Kaggle. *Yale/UNC-CH - Geophysical Waveform Inversion Data*. https://www.kaggle.com/competitions/waveform-inversion/data. Accessed: July 26, 2025. 2025.

[20] J. Lamy-Poirier. "Layered gradient accumulation and modular pipeline parallelism: fast and efficient training of large language models". In: *arXiv preprint arXiv:2106.02679* (2021). URL: https://arxiv.org/abs/2106.02679.

[21] X. Li, L. Li, and S. Fomel. "FWIGAN: Full-waveform inversion via a physics-informed generative adversarial network". In: *Geophysics* 88.3 (2023), R319–R332. DOI: 10.1190/geo2022-0313.1.

[22] Y. Lin. *OpenFWI*. https://sites.google.com/site/youzuolin044/openfwi. Accessed: July 26, 2025.

[23] Y. Lin, USDOE Laboratory Directed Research, and Development. *InversionNet*. Tech. rep. Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Aug. 2022. DOI: 10.11578/dc.20220915.7. URL: https://doi.org/10.11578/dc.20220915.7.

[24] Yong Liu et al. "iTransformer: Inverted Transformers Are Effective for Time Series Forecasting". In: *arXiv preprint arXiv:2310.06625* (2023). URL: https://arxiv.org/abs/2310.06625.

[25] Z. Liu et al. "Swin Transformer V2: Scaling Up Capacity and Resolution". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022b. URL: https://arxiv.org/abs/2111.09883.

[26] Z. Liu et al. "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows". In: *International Conference on Computer Vision (ICCV)*. 2021. URL: https://arxiv.org/abs/2103.14030.

[27] I. Loshchilov and F. Hutter. "Decoupled Weight Decay Regularization". In: *International Conference on Learning Representations (ICLR)*. 2019. URL: https://arxiv.org/abs/1711.05101.

[28] I. Loshchilov and F. Hutter. "SGDR: Stochastic Gradient Descent with Warm Restarts". In: *International Conference on Learning Representations (ICLR)*. 2017. URL: https://arxiv.org/abs/1608.03983.

[29] O. Lundstrom. *Geophysical Waveform Inversion Solution*. Kaggle. 2025. URL: https://www.kaggle.com/code/olaflundstrom/geophysical-waveform-inversion-solution.

[30] A. Mardan. *pinn_fwi*. GitHub repository. 2022. URL: https://github.com/AmirMardan/pinn_fwi.

[31] OpenFWI Team. *OpenFWI Datasets*. https://openfwi-lanl.github.io/docs/data.html. Accessed: July 26, 2025.

[32] R. Pascanu, T. Mikolov, and Y. Bengio. "On the Difficulty of Training Recurrent Neural Networks". In: *International Conference on Machine Learning (ICML)*. 2013. URL: https://arxiv.org/abs/1211.5063.

[33] R. E. Plessix. "A review of the adjoint-state method for computing the gradient of a functional with applications to seismology". In: *Geophysical Journal International* 167.2 (2006), pp. 495–503. DOI: 10.1111/j.1365-246X.2006.02978.x.

[34] B. T. Polyak and A. B. Juditsky. "Acceleration of Stochastic Approximation by Averaging". In: *SIAM Journal on Control and Optimization* 30.4 (1992), pp. 838–855. URL: https://epubs.siam.org/doi/abs/10.1137/0330046.

[35] R. G. Pratt. "Seismic waveform inversion in the frequency domain, Part 1: Theory and verification in a physical scale model". In: *Geophysics* 64.3 (1999), pp. 888–901. DOI: 10.1190/1.1444597.

[36] PyTorch. *Accelerating PyTorch Vision Models with Channels Last on CPU*. 2025. URL: https://pytorch.org/blog/accelerating-pytorch-vision-models-with-channels-last-on-cpu/ (visited on 07/30/2025).

[37] PyTorch. *CosineAnnealingWarmRestarts*. 2025. URL: https://docs.pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.CosineAnnealingWarmRestarts.html (visited on 07/30/2025).

[38] PyTorch. *LinearLR*. 2025. URL: https://docs.pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.LinearLR.html (visited on 07/30/2025).

[39] PyTorch. *torch.nn.parallel.DistributedDataParallel*. Accessed: July 2025. 2023. URL: https://docs.pytorch.org/docs/stable/generated/torch.nn.parallel.DistributedDataParallel.html (visited on 10/05/2023).

[40] PyTorch Developers. *Randomness in PyTorch*. Accessed: 2025-07-30. PyTorch. 2025. URL: https://docs.pytorch.org/docs/stable/notes/randomness.html.

[41] L. Qin. *Geophysical Waveform Inversion with KFold*. Kaggle. 2025. URL: https://www.kaggle.com/code/lonnieqin/geophysical-waveform-inversion-with-kfold/input.

[42] S. Rajup. *OpenFWI InversionNet [Train] with 670G Datasets*. Kaggle. 2025. URL: https://www.kaggle.com/code/seshurajup/openfwi-inversionnet-train-with-670g-datasets/input.

[43] O. Ronneberger, P. Fischer, and T. Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. 2015, pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28.

[44] L. I. Rudin, S. Osher, and E. Fatemi. "Nonlinear Total Variation Based Noise Removal Algorithms". In: *Physica D: Nonlinear Phenomena* 60.1-4

(1992), pp. 259–268. URL: https://www.sciencedirect.com/science/article/pii/016727899290242F.

[45] Bahaa E. A. Saleh, ed. *Introduction to Subsurface Imaging*. Cambridge, UK: Cambridge University Press, 2011. URL: https://www.cambridge.org/core/books/introduction-to-subsurface-imaging/939A3E26E5A448A4E25E034D519B35C7.

[46] Bahaa E. A. Saleh and Malvin Carl Teich. *Fundamentals of Photonics*. 3rd ed. Hoboken, NJ, USA: John Wiley & Sons, 2019. URL: https://www.wiley.com/en-us/Fundamentals+of+Photonics,+3rd+Edition-p-9781119506874.

[47] M. Saraiva et al. "Data-driven Full-waveform Inversion Surrogate using Conditional Generative Adversarial Networks". In: *arXiv preprint arXiv:2105.00100* (2021). URL: https://arxiv.org/abs/2105.00100.

[48] W. Shi et al. "Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016. URL: https://arxiv.org/abs/1609.05158.

[49] I. Sobel. "An Isotropic 3x3 Image Gradient Operator". Presented at the Stanford Artificial Intelligence Laboratory (SAIL). 1970.

[50] N. Srivastava et al. "Dropout: A Simple Way to Prevent Overfitting". In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958. URL: https://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf.

[51] Suraj Subramanian. *What is Distributed Data Parallel (DDP)*. 2024. URL: https://docs.pytorch.org/tutorials/beginner/ddp_series_theory.html (visited on 07/30/2025).

[52] W. W. Symes. "Migration velocity analysis and waveform inversion". In: *Geophysical Prospecting* 56.6 (2008), pp. 765–790. DOI: 10.1111/j.1365-2478.2008.00724.x.

[53] W. W. Symes. "The seismic reflection inverse problem". In: *Inverse Problems* 25.12 (2009), p. 123008. DOI: 10.1088/0266-5611/25/12/123008.

[54] A. Tarantola. "A strategy for nonlinear elastic inversion of seismic reflection data". In: *Geophysics* 51.10 (1986), pp. 1893–1903. DOI: 10.1190/1.1442249.

[55] A. Tarantola. "Inversion of seismic reflection data in the acoustic approximation". In: *Geophysics* 49.8 (1984), pp. 1259–1266. DOI: 10.1190/1.1441754.

[56] A. Vaswani et al. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017. URL: https://arxiv.org/abs/1706.03762.

[57] J. Virieux and S. Operto. "An overview of full-waveform inversion in exploration geophysics". In: *Geophysics* 74.6 (2009), WCC1–WCC26. DOI: 10.1190/1.3215770.

[58] H. Wang. *Waveform Inversion Kaggle Competition Tutorial*. Kaggle. 2025. URL: https://www.kaggle.com/code/hanchenwang114/waveform-inversion-kaggle-competition-tutorial.

[59] Z. Wang et al. “Image Quality Assessment: From Error Visibility to Structural Similarity”. In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. URL: https://ieeexplore.ieee.org/document/1284395.

[60] M. Warner et al. “Anisotropic 3D full-waveform inversion”. In: *Geophysics* 78.2 (2013), R59–R73. DOI: 10.1190/geo2012-0293.1.

[61] N. D. Whitmore and W. S. Crawley. “Seismic modeling with vector reflectivity”. In: *SEG Technical Program Expanded Abstracts 2020*. 2020, pp. 2708–2712. DOI: 10.1190/segam2020-3426299.1.

[62] Wikipedia contributors. *Inverse Probability Weighting*. Accessed: 2025-07-30. Wikipedia. 2025. URL: https://en.wikipedia.org/wiki/Inverse_probability_weighting.

[63] Y. Wu and Y. Lin. “InversionNet: A Real-Time and Accurate Full Waveform Inversion with CNNs and continuous CRFs”. In: *arXiv preprint arXiv:1811.07875* (2018). URL: https://arxiv.org/abs/1811.07875.

[64] P. Yang, J. Ma, and G. B. Biondi. “Time-domain full-waveform inversion on a GPU”. In: *SEG Technical Program Expanded Abstracts 2015*. 2015, pp. 1362–1366. DOI: 10.1190/segam2015-5858005.1.

[65] S. Yu and J. Ma. “Deep Learning for Geophysics: Current and Future Trends”. In: *Reviews of Geophysics* 59.3, e2021RG000742 (2021). DOI: 10.1029/2021RG000742.

[66] Q. Zeng et al. “InversionNet3D: Efficient and Scalable Learning for 3D Full Waveform Inversion”. In: *arXiv preprint arXiv:2103.14158* (2021). URL: https://arxiv.org/abs/2103.14158.

[67] Christopher Zerafa, Paul Galea, and Cristina Sebu. “Data-Driven and Theory-Guided Pseudo-Spectral Seismic Imaging using Deep Neural Network Architectures”. PhD thesis. University of Malta, 2023.

[68] Christopher Zerafa, Paul Galea, and Cristina Sebu. “Synergizing Deep Learning and Full-Waveform Inversion: Bridging Data-Driven and Theory-Guided Approaches for Enhanced Seismic Imaging”. In: *arXiv preprint arXiv:2502.17585* (2025). URL: https://arxiv.org/abs/2502.17585.

[69] Z. Zhang and Y. Lin. “VelocityGAN: Data-Driven Seismic Waveform Inversion Using Conditional Adversarial Networks”. In: *arXiv preprint arXiv:1809.10262* (2018). URL: https://arxiv.org/abs/1809.10262.