Nanoscale

ARTICLE TYPE

Cite this: DOI: 00.0000/xxxxxxxxxx

Supplementary material for: Inverse deep learning methods and benchmarks for artificial electromagnetic material design†

Simiao Ren [a‡], Ashwin Mahendra [a‡], Omar Khatib[a], Yang Deng [a], Willie J. Padilla,[a] and Jordan M. Malof [a]

1 Invertible neural networks

Invertible neural networks (INNs) and conditional INNs (cINNs) have recently been found to achieve state-of-the-art results for solving general data-driven inverse problems [1–3], however they have yet to be explored for AEM problems. In this work we adapt INNs and cINNs to solve inverse AEM problems. We next summarize the technical details of these methods, however, readers can find a complete description for the INN in Ardizzone et al. [1] and the cINN in Kruse et al. [2].

INNs are based upon flow-based models [4–6], which assume that all data can be modeled as a sample from some non-elementary probability distribution, denoted as $S \sim p_S(s)$ where $p_S(s)$ is the probability distribution of S. Then it is assumed that $S = f_\theta(Z)$, where $Z \sim p_Z(z)$ is a multi-dimensional Gaussian distribution with a diagonal covariance matrix (i.e., each random variable is independent), and f is some function parameterized by θ. Although most real-world data (e.g., images or spectra) are not well-modeled by a multi-dimensional Gaussian, if f_θ is sufficiently complex then p_S will also be sufficiently complex to accurately model real-world data.

To achieve this level of complexity, the transformation is often implemented with a deep neural network, and the parameters θ are learned based upon real-world data. Other contemporary models make the aforementioned assumptions (notably, the variational autoencoder) however, flow-based models (FBMs) are unique because they impose structure on the deep neural network that ensures that it is bijective (a one-to-one mapping), and therefore invertible. Furthermore, the inverse function f^{-1} can be readily evaluated once the FBM is trained (i.e., once we infer f), so that it is trivial to evaluate by $s = f(z)$ or $z = f^{-1}(s)$

Invertible Neural Network (INN). Inspired by the success of FBMs in modeling complicated distributions, [5,6] Ardizzone et al. [1] adapted the FBM to solving inverse problems. One constraint of the FBMs is that the dimensionality of its input and output must be identical, however, in most inverse problems the output is lower-dimensional than the input (i.e., $|s| < |g|$ using our notation). To overcome this issue, they propose to formulate the forward model as $[s,z] = f(g)$, where z is a normally distributed random variable. The dimensionality of z is chosen so that $|s| + |z| = |g|$. Then we train a FBM (i.e., an invertible neural network), f_θ, that approximates the forward model, and where θ represents the network parameters. Because the network is invertible, given a particular value of s and a randomly-sampled value of z, we can also compute $\hat{g} = f(s,z)$. We can propose multiple solutions for a given value of s by sampling multiple values of z and passing them through f_θ^{-1} with the same value of s.

The INN is trained by making forward passes (i.e., computing $[s,z] = f_\theta(g)$) for samples of data in the training dataset. The parameters of the model are adapted to make accurate predictions of s as well as making predictions of z that are normally distributed. Mathematically, the objective function, or "loss", for training the INN is given by:

$$\mathscr{L} = \frac{1}{2} \cdot \left(\frac{1}{\sigma^2} \cdot (\hat{s} - s)^2 + z^2 \right) - \log |\det J_{g \mapsto [s,z]}|. \quad (1)$$

Here σ is a hyper-parameter controlling the trade-off between

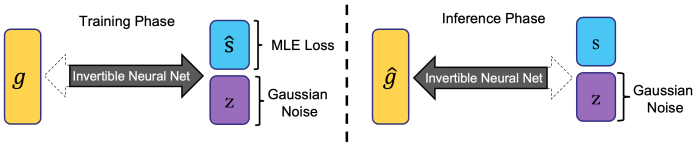

Fig. 1 Schematic architecture of an Invertible Neural Network.

accurate s prediction and the normality of z. The variable J represents the Jacobian matrix of the mapping learned by the neural network to account for the probability volume change due to change of variable, which is trivially computed using invertible network structure . INNs can also optionally be trained with an additional loss term comprising a maximum mean discrepancy (MMD) [7] measure, however, the INN was found to make more accurate predictions without it [1], and therefore we exclude it from our implementation.

In all of the AEM problems considered in our benchmark the dimensionality of the output is substantially larger than the input (i.e., $|s| >> |g|$), breaking an assumption of the INN that $|s| < |g|$. To address this problem we follow the approach in [1] and concate-

‡ *These authors contributed equally*

[a] *Department of Electrical and Computer Engineering, Duke University, Box 90291, Durham, NC 27708, USA. E-mail: jordan.malof@duke.edu*

nate a vector of zeros to g so that $|g|+|g_0| = |s|+|z|$, where g_0 is the vector of zeros. This ad-hoc procedure is remedied by the cINN, which is described next.

Conditional Invertible Neural Network (cINN). The cINN was introduced in Kruse et al.[2], and reformulates the inverse problem as $z = f(g|s)$, where z is a vector sampled from a multi-dimensional Normal distribution. Once again we use a neural network to model f based upon data, however the goal is now to learn a mapping between g and z that is *conditioned* upon s. The authors propose an architectural change to the original invertible neural network so that the function can be conditioned on some auxiliary value (e.g., s in our case) whether mapping in the forward or inverse direction. This formulation implies that the dimensionality of s and g no longer need to match, and we simply need to set the dimensionality of z to match g. The cINN model trains with the following loss function:

$$\mathscr{L} = \frac{1}{2}z^2 - \log|\det J_{g\mapsto z}|. \tag{2}$$

Similar to the INN, the loss enforces the normality of z, however, s no longer appears because the cINN is not tasked with predicting it. Once again J represents the Jacobian matrix of the mapping learned by the neural network, this time when mapping from g to z.

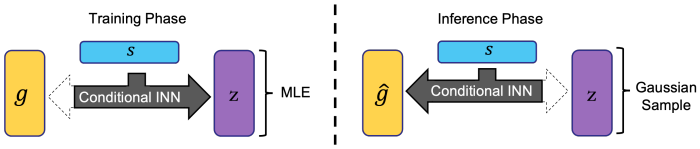

Fig. 2 Schematic architecture of a conditional Invertible Neural Network.

2 Benchmark deep inverse models

Here we describe two key properties of DIMs with respect to AEM applications. Table 2 is main text classifies each of our benchmark models according to several properties – discussed next. The first important property is whether a DIM produces multiple solutions. As we find in result section of main text, models that are permitted to propose several solutions often also find progressively better solutions (i.e., r_T reduces as T grows). This capability also makes it possible for the designer to consider several viable solutions that may have somewhat different scattering, and choose the one that is best-suited for the application. It is important to note that each additional model proposal that is considered requires an evaluation of the true forward model, f, which imposes a (usually modest) trade-off between design quality and computation time.

A second important property of some DIMs is that they rely on an iterative process for inferring each inverse solution. Most DIMs attempt to learn a direct mapping from s to g, and therefore inference of a single solution proposal is computationally efficient. In contrast, iterative methods usually make an initial set of guesses for the inverse solution, denoted Z_0. A search for superior solutions is then performed based upon Z_0, and the results are used to update Z_0. This process is then repeated for some fixed number of iterations, or until the quality of the solutions (e.g., estimated re-simulation error) no longer improves. As we find iterative methods can often achieve the superior accuracy, although the computation required to infer solutions can be substantially larger than other methods. It is worth noting however, that this additional computation time is usually only a small fraction of the time for other processes, such as training the DIMs or evaluating f using computational simulators.

2.1 Description of benchmark models

In this section we provide brief descriptions of each benchmark model. We focus on describing the specific implementation that we use for our experiments, and its justification. For a more thorough treatment of the models we refer readers to recent reviews[8].

Conventional Deep Neural Network (DNN). In this approach the inverse problem is treated as a standard regression problem and a conventional DNN is used[9–13]. Input is the spectra or response property of metamaterial and the output is the geometry properties. The backpropagation of a standard regression loss (e.g., mean-squared error) between the predicted and the true geometry updates the parameters of the neural network. No additional techniques are incorporated to compensate for one-to-many mappings between spectra and geometry. During training, the network is penalized for deviations between predicted geometries and the input spectrum's true geometry regardless of the re-simulation error between the target spectrum and spectrum of the predicted geometry. Residual connections, convolutional layers, batch normalization and dropouts can be added to boost performance when needed. In our specific implementation, we added batch normalization for all of three datasets for convergence acceleration.

Genetic Algorithm (GA). Genetic Algorithms (GAs), a set of iterative, optimization-based algorithms for inverse AEM design[14–16]. Our GA closely follows the model employed in Zhang et al[15] and Forestiere et al[16] to solve inverse AEM problems. The GA first produces a set, or population, of initial geometries by randomly sampling each parameter of the geometry from a uniform distribution. Each geometry is then passed through the forward model of the process, $s = f(g)$, so that its re-simulation error can be computed with respect to the target spectrum. This re-simulation error is used to determine the quality or "fitness" of each geometry in the population. The traditional implementation of the GA relies on evaluating the true forward model for each candidate geometry, which can be computationally intensive if this requires computational simulation, as is the case for our ADM problem. To overcome this obstacle, we train a deep neural network to approximate f, and use this to compute the re-simulation error, dramatically accelerating the inference time of the GA.

Once the fitness of each candidate geometry is comptued, the next step is the selection of "parent" geometries from this population. Crossover and mutation operations on this "parent" subset produce the next generation. We utilize a "roulette-wheel" selection pattern, as implemented by Zhang et al[15] and Forestiere et al[16], to select parent geometries. In roulette-wheel selection, the

probability of a particular geometry being selected is proportional to that geometry's fitness. Our GA also implements "elitism", whereby copies of a certain number (determined by the elitism hyper-parameter) of geometries of absolute greatest fitness automatically pass to the future child population without undergoing mutation or crossover.

Neural Adjoint (NA). Neural Adjoint (NA) method was recently proposed in Ren et al.[17] and subsequently found to be highly effective for solving AEM inverse problems Deng et al.[18]. The NA is a gradient-based optimization approach that was developed based upon similar earlier methods employed in the AEM community[17–21]. The NA relies on the pre-trained forward network that maps from g to s space accurately as a proxy forward model to provide analytical gradient feedback during inverse inference. After the training of the forward network, by freezing the network weights and setting the input geometries as trainable parameters, it iteratively trains a set of (randomly) initial geometries to approach the target spectrum output by backpropagation of a mean squared error and boundary loss over the output and target spectra. The key difference between the NA and prior approaches is an additional boundary loss term[17], which steeply penalizes geometries outside the training domain of the proxy forward neural network to ensure that the network only returns g values that are within the domain of the training data (where the proxy model makes accurate predictions). The boundary loss was found to greatly improve the performance compared to prior methods, and therefore we adopt the NA from Ren et al.[17] here.

Tandem Network (TD). The so-called data collision problem causes unstable gradient signals, and one method proposed to mitigate this issue is the tandem structure[22–35]. TDs first train a neural network, the forward network, capable of accurately solving the forward problem from g to s with standard regression loss (MSE). The forward network's weights are then fixed and its inputs are tied to the outputs of a second, separate network. The combined network is trained on the inverse problem. The fixed-weight forward network approximates a geometry-to-spectra simulator, training the pre-pended network with a standard regression loss between the predicted and input spectrum. The pre-pended network then learns a one-to-one relationship from input spectra and output geometries. As there is not much variation between TD architectures, we used the initial implementation of[22] with the modification to add a boundary loss term that was proposed in Ren et al.[17], since this was found to significantly improve its performance.

Mixture Density Network (MDN). MDNs[36] account for the one-to-many nature of the inverse problem by assuming the distribution of conditional probability $p(g|s)$ is a mixture of Gaussian random variables. Its effectiveness has been shown on recent AEM design[17,37,38]. The number of Gaussian mixtures approximates the number of "many" in the one-to-many setting (hardly known beforehand) and is a sensitive hyper-parameter that has to be tuned carefully using a validaton set. We used the same implementation of the MDN as that presented in the literature where a diagonal covariance matrix were used.[37]

Conditional Variational Autoencoder (VAE). VAEs[39] also model the inverse problem probabilisticly and have shown successful design results in AEM[40–47]. Similar to an auto-encoder, where the inputs are encoded into latent space and decoded back for the reconstruction of the original input signal, VAEs uses a variational approach and encourage the latent space to become a Gaussian distribution. In AEM design space, the encoder ($p(z|g,s)$) encodes g into latent space z conditioned on s, and the decoder ($p(g|z,s)$) decodes the sampled latent space variable z into g conditioned on s as well. During the inference phase, a random sample is drawn from the latent space and passed to the decoder to get the inverse solution conditioned on a given desired s. We used the original architecture from Ma et al.[41] without the convolution layers as none of our geometries are 2D and a parametric assumption of Gaussian latent space is effectively the same but architecturally simpler than the adversarial approach proposed by Kudyshev et al.[42].

3 Deep inverse models: further details

Here we supply additional details of the deep inverse models being benchmarked in this work.

3.1 Genetic Algorithm (GA).

For GA, we provide extra details for how the mutation steps are carried out here: Parents are arranged into pairs, with each pair producing two child geometries via single-point crossover and point mutation. The probability of a pair undergoing crossover is defined by a hyper-parameter termed the crossover rate. In single-point crossover, a splice point is randomly selected along the geometry parameter vector with uniform probability. Parameters following the splice point are swapped between the pair of parents. The resulting population undergoes point mutation. In point mutation, parameters of each geometry in the population are replaced at random by new parameters in the domain with a probability determined by a hyper-parameter called the mutation rate. The resulting child population replaces the original population in the next iteration. The overall process can be found below:

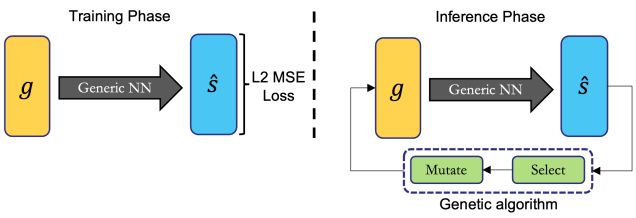

Fig. 3 Schematic architecture of Genetic Algorithm

3.2 Neural Adjoint with Boundary Loss (NA).

We supply the loss function and the process flow chart of the NA method. The loss function during inference time can be written

as:

$$\mathscr{L}_{train} = (\hat{f}(g) - s_{gt})^2 \tag{3}$$

$$\mathscr{L}_{infer} = (\hat{f}(\hat{g}) - g_{gt})^2 + \mathscr{L}_{bdy} \tag{4}$$

$$\mathscr{L}_{bdy} = ReLU(|\hat{g} - \mu_g| - \frac{1}{2}R_g) \tag{5}$$

where $\hat{f}$ is the forward proxy function, μ_g and R_g are the mean

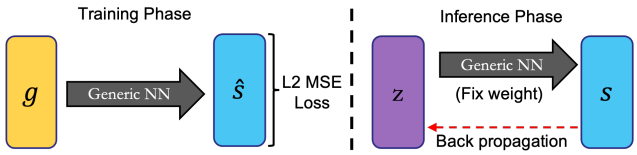

Fig. 4 Schematic architecture of Neural Adjoint

and range of the training set g. $\hat{g}$ is the trainable parameter of geometry during inference (g_0 randomly initialized and equivalent to z in probabilistic methods)

3.3 Naive Neural Network (NN).

We supply the loss function and the process flow chart of the NN method. The loss function is as follows where f^{-1} represents the naive neural network.

$$\mathscr{L} = (f^{-1}(s) - g_{gt})^2 \tag{6}$$

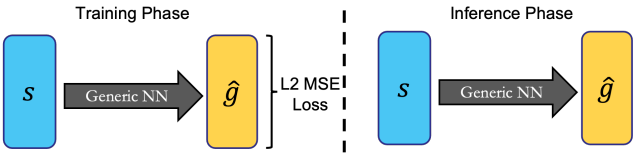

Fig. 5 Schematic architecture of Naive Neural Network

3.4 Tandem Network (TD).

We supply the loss function and the process flow chart of the TD method. The 2-stage training loss function for TD are as follows

$$\mathscr{L}_1 = (\hat{f}(g) - s_{gt})^2 \tag{7}$$

$$\mathscr{L}_2 = (\hat{f}(f^{-1}(s)) - g_{gt})^2 + \mathscr{L}_{bdy} \tag{8}$$

where the $\mathscr{L}_{bdy}$ is the same as defined in eq 5. In our benchmarking tasks, to make a fair comparison, the same forward network $\hat{f}$ is used for TD, the Genetic Algorithm, and the Neural Adjoint with boundary loss.

Fig. 6 Schematic architecture of Tandem model

3.5 Mixture Density Network (MDN).

We supply the loss function and the process flow chart of the MDN method. The loss function is as follows where p_i, μ_i, Σ_i represents the probability of geometry coming from Gaussian distribution i and the mean and variance of Gaussian distribution i.

$$\mathscr{L} = -\log(\sum_i p_i * |\Sigma_i^{-1}|^{\frac{1}{2}} * \exp(-\frac{1}{2}(\mu_i - g)^T \Sigma_i^{-1}(\mu_i - g))) \tag{9}$$

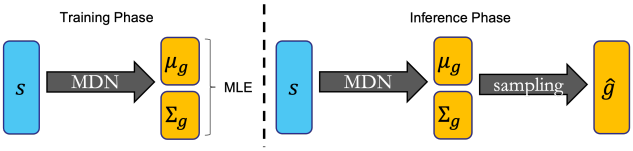

Fig. 7 Schematic architecture of Mixture Density Network

3.6 Conditional Variational Autoencoder (cVAE).

We supply the loss function and the process flow chart of the TD method. The following loss term captures the training process of a cVAE. z is forced to follow a normal distribution (μ_z, σ_z is the mean and variance of z) and trained by minimizing the KL divergence between the encoder $q(z|g,s)$ and a normal distribution. A mean squared reconstruction loss encourages the decoder $p(g|z,s)$ to accurately reconstruct the input geometry given a latent vector sampled from z and the condition s. α is a hyper-parameter trading off the reconstruction and normality of the latent space and needs to be tuned carefully on a validation set.

$$\mathscr{L} = (g - \hat{g})^2 - \frac{\alpha}{2} \cdot (1 + log\sigma_z + \mu_z^2 - \sigma_z) \tag{10}$$

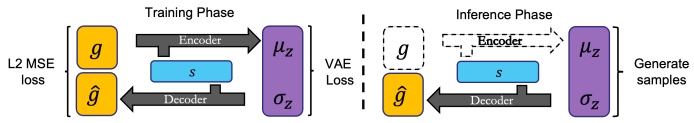

Fig. 8 Schematic architecture of Conditional Variational Autoencoder

4 Forward surrogate models

4.1 Forward model performance compared to original work

Our forward models are optimized for each benchmarking task. For the Shell problem we achieved a forward model MRE of 0.65%, compared to 1.5% in the original work over 5,000 test samples. For the Graphene-Si_3N_4 2D Multi-layer Stack, our NN achived MRE of 0.4% while Chen et al. reported with MRE of 3.2% given 1,000 samples[9]. On the Super-unit Cell benchmark solved by Deng et al[18], we get similar MSE values (1.2e-3) on $T = 1$ performance.

5 Model size

Here we present the table of model size in number of trainable parameters in Table 1.

Model	Multi-layer Stack (Chen)	Multi-layer Shell (Peurifoy)	Super-unit Cell (Deng)
Conventional Deep Neural Network (NN)	16M	25M	36M
Tandem (TD)	3M	93M	47M
Genetic Algorithm	4M	40M	17M
Neural Adjoint (NA)	4M	40M	17M
Variational Auto-encoder (cVAE)	12M	18M	14M
Invertible Neural Network (INN)	8M	35M	25M
Conditional Invertible Neural Network (cINN)	3M	35M	23M
Mixture Density Network (MDN)	0.4M	0.1M	1M

Table 1 Model size reflected by the total number of trainable parameters. Note that the tandem model is typically bigger due to it make use of a forward and a backward model and MDN models are small due to after hyper-parameter sweep all larger models exhibit much worse performance than the current selected best one.

6 $\hat{r}_T$ metric visualization

The $\hat{r}_T$ metrics captures the performance of our DIMs' improvement with respect to the number of simulations allowed (or trails). It is defined as

$$\hat{r}_T = \frac{1}{|D_{te}|} \sum_{s \in D_{te}} [\min_{i \in [1,T]} \mathscr{L}(\hat{s}(z_i), s)]$$

where D_{te} is the test set and $\hat{s_{z_i}}$ is the simulated spectra of the inverse solution when latent vector z_i is input into the DIM of interest.

To better help with the understanding of this value, we plot $\mathscr{L}(\hat{s}(z_i), s)$ along with $\min_{i \in [1,T]} \mathscr{L}(\hat{s}(z_i), s)$ to illustrate this process of taking the minimum in Fig 9. As the plot shows, the minimal error is taken across all values of [1, T] for each target spectra and hence it is monotonic decreasing. The final $\hat{r_T}$ is averaged across 500 different test target spectra.

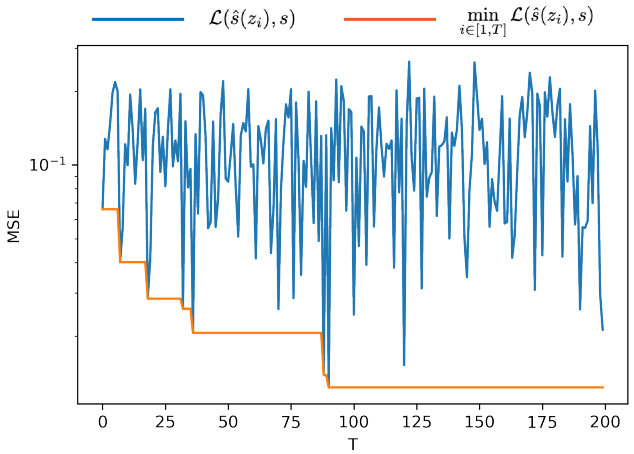

Fig. 9 Comparison between individual solution error and minimal error for error of the 200 solution proposals made by cINN on Stack dataset (randomly selected) on one single randomly chosen target spectra. X axis is T, the number of proposed solutions and y axis is $\mathscr{L}(\hat{s}, s)$, which is measured in MSE. As the plot shows, as minimal error is taken till current T value, the minimal value is monotonic decreasing (non-increasing).

7 Stack dataset DIM performance improvement over T

In main text, we discussed that the performance improvement over T for DIMs on Stack dataset is mainly due to "jittering" around the solution space instead of actually finding new solutions, here we show empirical evidence Fig 10 supporting the claim.

As Fig 10 shows, all top 50 points cluster nicely in local regions, supporting the statement that these DIMs only explore locally for the Stack dataset (with the lowest non-uniqueness) when T gets larger.

8 Another metric for one-to-many based on neighbourhood

We also provide some quantified measurement of the distance that is not distorted by the dimensionality reduction, we provide D_r (defined below) to reflect the "localness" of the clustered points, normalized by the pairwise distance of the whole dataset. $D_r \approx 1$ would mean that the cluster is closer to randomly drawn from the whole dataset, therefore a localized draw would have a $D_r < 1$.

$$D_r = \frac{\text{Avg pairwise } G \text{ distance within cluster}}{\text{Avg pairwise } G \text{ distance between all points}}$$

$$= \frac{\frac{1}{C}\Sigma_c^C \frac{1}{K(K-1)} \Sigma_i^K \Sigma_{j \neq i}^K |G_{c,i} - G_{c,j}|^2}{\frac{1}{N(N-1)} \Sigma_i^N \Sigma_{j \neq i}^N |G_i - G_j|^2}$$

where C is the number of cluster (5 in our case), K is the number of data points in each cluster (5 in our case), N is the size of the whole dataset, $G_{c,i}$ represents the i-th geometry of cluster c.

The D_r value is labelled on the right top corner of Fig **??**. As this "closeness" measure increase from order: Stack < Shell < ADM, we see the same trend quantitatively with the perceived one-to-manyness on the plot as well.

9 Hardware for training used

Here we list the hardwares we used in the model training and inference to provide context for our running time. We are using NVIDIA GTX3090 cards for GPU, AMD 3990X (64 cores, 2.90GHz,

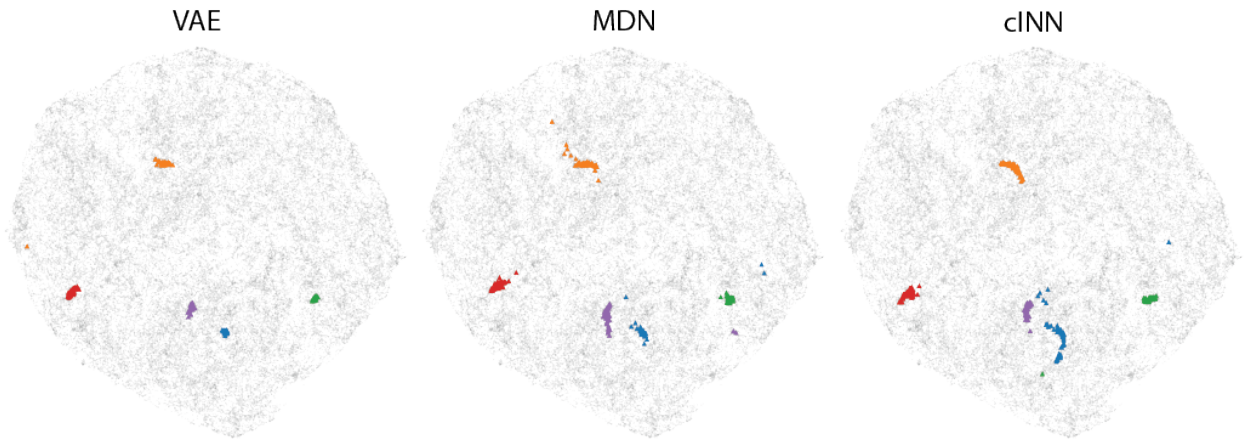

Fig. 10 Cluster effect of retrieved $\hat{g}$ of Stack dataset top-3 performing algorithms (at T=200) top 50 proposed solution. Same as the main text umap plot, the background grey points are the scatter point of all point of Stack dataset. On top of that, for each of the 5 random spectra (labelled by different color), 200 random $\hat{g}$ was retrieved by individual models and top 50 chosen by the real simulator.

Dataset	Stack	Shell	ADM
γ	0.52	1.43	15.0
D_r	0.24	0.58	0.84

Table 2 Ratio of forward problem MSE and inverse problem MSE and perceived one-to-manyness

256 MB cache), RAM: 256GB.

Conflicts of interest

Acknowledgements

Notes and references

1 L. Ardizzone, J. Kruse, C. Rother and U. Köthe, International Conference on Learning Representations, 2019.

2 J. Kruse, L. Ardizzone, C. Rother and U. Köthe, Workshop on Invertible Neural Networks and Normalizing Flows, International Conference on Machine Learning, 2019.

3 *Invertible Neural Nets and Normalizing Flows*, `https://invertibleworkshop.github.io/`.

4 L. Dinh, D. Krueger and Y. Bengio, *arXiv preprint arXiv:1410.8516*, 2014.

5 L. Dinh, J. Sohl-Dickstein and S. Bengio, *arXiv preprint arXiv:1605.08803*, 2016.

6 D. P. Kingma and P. Dhariwal, *arXiv preprint arXiv:1807.03039*, 2018.

7 A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf and A. Smola, *Journal of Machine Learning Research*, 2012, **13**, 723–773.

8 O. Khatib, S. Ren, J. Malof and W. J. Padilla, *Advanced Functional Materials*, 2021, 2101748.

9 Y. Chen, J. Zhu, Y. Xie, N. Feng and Q. H. Liu, *Nanoscale*, 2019, **11**, 9749–9755.

10 M. H. Tahersima, K. Kojima, T. Koike-Akino, D. Jha, B. Wang, C. Lin and K. Parsons, *Scientific Reports*, 2019, **9**, 1–9.

11 T. Zhang, J. Wang, Q. Liu, J. Zhou, J. Dai, X. Han, Y. Zhou and K. Xu, *Photonics Research*, 2019, **7**, 368.

12 N. Akashi, M. Toma and K. Kajikawa, *Applied Physics Express*, 2020, **13**,.

13 R. Lin, Y. Zhai, C. Xiong and X. Li, *Optics Letters*, 2020, **45**, 1362.

14 J. Johnson and V. Rahmat-Samii, *IEEE Antennas and Propagation Magazine*, 1997, **39**, 7–21.

15 T. Zhang, Q. Liu, Y. Dan, S. Yu, X. Han, J. Dai and K. Xu, *Optics Express*, 2020, **28**, 18899.

16 C. Forestiere, A. J. Pasquale, A. Capretti, G. Miano, A. Tamburrino, S. Y. Lee, B. M. Reinhard and L. D. Negro, *Nano Letters*, 2012, **12**, 2037–2044.

17 S. Ren, W. J. Padilla and J. M. Malof, Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020.

18 Y. Deng, S. Ren, K. Fan, J. M. Malof and W. J. Padilla, *Optics Express*, 2021, **29**, 7526.

19 J. Peurifoy, Y. Shen, L. Jing, Y. Yang, F. Cano-Renteria, B. G. DeLacy, J. D. Joannopoulos, M. Tegmark and M. Soljačić, *Science Advances*, 2018, **4**, 1–8.

20 T. Asano and S. Noda, *Optics Express*, 2018, **26**, 32704.

21 Y. Miyatake, N. Sekine, K. Toprasertpong, S. Takagi and M. Takenaka, *Japanese Journal of Applied Physics*, 2020, **59**, SGGE09.

22 D. Liu, Y. Tan, E. Khoram and Z. Yu, *ACS Photonics*, 2018, **5**, 1365–1369.

23 W. Ma, F. Cheng and Y. Liu, *ACS Nano*, 2018, **12**, 6326–6334.

24 E. Ashalley, K. Acheampong, L. Vázquez, P. Yu, A. Neogi, A. O. Govorov and Z. Wang, *Photonics Research*, 2020.

25 S. So, J. Mun and J. Rho, *ACS Applied Materials and Interfaces*, 2019, **11**, 24264–24268.

26 I. Malkiel, M. Mrejen, A. Nagler, U. Arieli, L. Wolf and H. Suchowski, *Light: Science and Applications*, 2018, **7**,.

27 L. Pilozzi, F. A. Farrelly, G. Marcucci and C. Conti, *Communications Physics*, 2018, **1**, 1–7.

28 Y. Long, J. Ren, Y. Li and H. Chen, *Applied Physics Letters*, 2019, **114**,.

29 L. Xu, M. Rahmani, Y. Ma, D. A. Smirnova, K. Z. Kamali, F. Deng, Y. K. Chiang, L. Huang, H. Zhang, S. Gould, D. N. Neshev and A. E. Miroshnichenko, *Advanced Photonics*, 2020, **2**, 1.

30 Z. Hou, T. Tang, J. Shen, C. Li and F. Li, *Nanoscale Research Letters*, 2020, **15**,.

31 J. He, C. He, C. Zheng, Q. Wang and J. Ye, *Nanoscale*, 2019, **11**, 17444–17459.

32 L. Gao, X. Li, D. Liu, L. Wang and Z. Yu, *Advanced Materials*, 2019, **31**, 1905467.

33 A. Mall, A. Patil, D. Tamboli, A. Sethi and A. Kumar, *Journal of Physics D: Applied Physics*, 2020, **53**, 49LT01.

34 A. D. Phan, C. V. Nguyen, P. T. Linh, T. V. Huynh, V. D. Lam, A.-T. Le and K. Wakabayashi, *Crystals*, 2020, **10**, 125.

35 R. Singh, A. Agarwal and B. W. Anthony, *Optics Express*, 2020, **28**, 27893.

36 C. M. Bishop, 1994.

37 R. Unni, K. Yao and Y. Zheng, *ACS Photonics*, 2020, **7**, 2703–2712.

38 R. Unni, K. Yao, X. Han, M. Zhou and Y. Zheng, *Nanophotonics*, 2021.

39 D. P. Kingma and M. Welling, 2014.

40 W. Ma, F. Cheng, Y. Xu, Q. Wen and Y. Liu, *Advanced Materials*, 2019, **31**, 1901111.

41 W. Ma and Y. Liu, *Science China: Physics, Mechanics and Astronomy*, 2020, **63**,.

42 Z. A. Kudyshev, A. V. Kildishev, V. M. Shalaev and A. Boltasseva, *Applied Physics Reviews*, 2020, **7**, 021407.

43 Z. A. Kudyshev, A. V. Kildishev, V. M. Shalaev and A. Boltasseva, *Nanophotonics*, 2020, **1**,.

44 T. Qiu, X. Shi, J. Wang, Y. Li, S. Qu, Q. Cheng, T. Cui and S. Sui, *Advanced Science*, 2019, **6**,.

45 X. Shi, T. Qiu, J. Wang, X. Zhao and S. Qu, *Journal of Physics D: Applied Physics*, 2020, **53**, 275105.

46 Z. Liu, L. Raju, D. Zhu and W. Cai, *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2020, **10**, 126–135.

47 Y. Kiarashinejad, S. Abdollahramezani and A. Adibi, *npj Computational Materials*, 2020, **6**, 1–12.