Computational Geosciences (2019) 23:925–952
https://doi.org/10.1007/s10596-019-09850-7

ORIGINAL PAPER

Parametric generation of conditional geological realizations using generative neural networks

Shing Chan[1] · **Ahmed H. Elsheikh**[1]

Received: 17 July 2018 / Accepted: 18 June 2019 / Published online: 13 July 2019

Abstract

Deep learning techniques are increasingly being considered for geological applications where—much like in computer vision—the challenges are characterized by high-dimensional spatial data dominated by multipoint statistics. In particular, a novel technique called *generative adversarial networks* has been recently studied for geological parametrization and synthesis, obtaining very impressive results that are at least qualitatively competitive with previous methods. The method obtains a neural network parametrization of the geology—so-called a *generator*—that is capable of reproducing very complex geological patterns with dimensionality reduction of several orders of magnitude. Subsequent works have addressed the conditioning task, i.e., using the generator to generate realizations honoring spatial observations (hard data). The current approaches, however, do not provide a parametrization of the conditional generation process. In this work, we propose a method to obtain a parametrization for direct generation of conditional realizations. The main idea is to simply extend the existing generator network by stacking a second *inference network* that learns to perform the conditioning. This inference network is a neural network trained to sample a posterior distribution derived using a Bayesian formulation of the conditioning task. The resulting extended neural network thus provides the conditional parametrization. Our method is assessed on a benchmark image of binary channelized subsurface, obtaining very promising results for a wide variety of conditioning configurations.

Keywords Parametrization · Deep learning · Geological models · Generative models · Multipoint geostatistics

1 Introduction

The large scale nature of geological models makes reservoir simulation an expensive task, prompting numerous works on parametrization methods that can preserve complex geological characteristics required for accurate flow modeling. A wide variety of methods exist including zonation [1, 2], PCA-based methods [3–5], SVD methods [6, 7], discrete cosine transform [8, 9], level set methods [10–12], and dictionary learning [13, 14]. Very recently, a new method from the machine learning community called *generative adversarial networks* [15] has been investigated [16–22] for the purpose of parametrization, reconstruction, and synthesis of geological properties, obtaining very competitive results in the visual quality of the generated images compared with previous methods. This adds to the recent trend in applying machine learning techniques [23–29] to leverage rapid advances in the field as well as the increasing availability of data and computational resources that enable these techniques to be effective.

Generative adversarial networks (GAN) is a novel technique for training a neural network to sample from a distribution that is unknown and intractable, by only using a dataset of realizations from this distribution. The result is a neural network parametrization called a *generator*, which is capable of generating new realizations from the target distribution—in our case, geological images—using a very efficient representation. Recent works show that using the

✉ Shing Chan
sc41@hw.ac.uk

Ahmed H. Elsheikh
a.elsheikh@hw.ac.uk

[1] Heriot-Watt University, Edinburgh, UK

generator to parametrize the geology is very effective in preserving high-order flow statistics [18, 22], two-point spatial statistics [16, 19], and morphology [16], all while achieving dimensionality reduction of several orders of magnitude.

Subsequent works on GAN focused on the problem of conditioning the generator: given a generator trained on unconditional realizations, the task is to generate realizations conditioned to spatial observations (hard data). In [20, 21], an image inpainting technique was used which adopts a sampling by optimization approach, i.e., it requires solving an optimization problem for each conditional realization that is generated. The method obtained very good results—in particular, [20] reported superior performance in many aspects compared to standard geomodeling tools. However, sampling by optimization can be expensive if realizations need to be continuously generated during deployment, e.g., for history matching or uncertainty quantification. An alternative approach was presented in [19], where the authors addressed conditioning using a Bayesian framework and performed Markov chain Monte Carlo to sample conditional realizations. Neither of these approaches, however, provides a parametrization for the conditional sampling process. As the authors in [19, 20] express, it is of interest to obtain such parametrization to directly sample conditional realizations without running optimizations or Monte Carlo methods.

In this work, we propose a method to obtain a parametrization to directly sample conditional realizations. The main idea is to simply extend the existing generator network by stacking a second *inference network* that performs the conditioning. This inference network is a neural network trained to sample a posterior distribution, derived using a Bayesian formulation of the conditioning task. The resulting extended neural network thus provides the conditional parametrization; hence, direct conditional sampling can be done very efficiently. We assess our method on the benchmark image of [30], finding positive results for a wide variety of conditioning configurations.

Note that although previous works [16, 19, 20] study applications of GAN mainly in the context of geomodeling and multipoint geostatistical simulations, here we emphasize on the effectiveness of GAN—and neural networks in general—for parametrization and dimensionality reduction, highlighting their ability to learn efficient representations for complex and high-dimensional data. The rest of this work is organized as follows: In Section 2, we describe parametrization using generative adversarial networks, and the Bayesian formulation of the conditioning problem. We introduce our method in Section 3 where we describe how the inference network is obtained. In Section 4, we show results for unconditional and conditional parametrization of binary channelized subsurface images. We discuss related work in Section 5 including other alternatives to train the inference network, and conclude our work in Section 6.

2 Background

In this section, we discuss the importance of parametrization for subsurface simulations (Section 2.1), we describe generative adversarial networks (Section 2.2), and we describe the Bayesian formulation of the conditioning problem (Section 2.3).

2.1 Parametrization

Parametrization is useful in subsurface simulations where the large number of uncertain variables are highly correlated and redundantly represented as a consequence of the grid

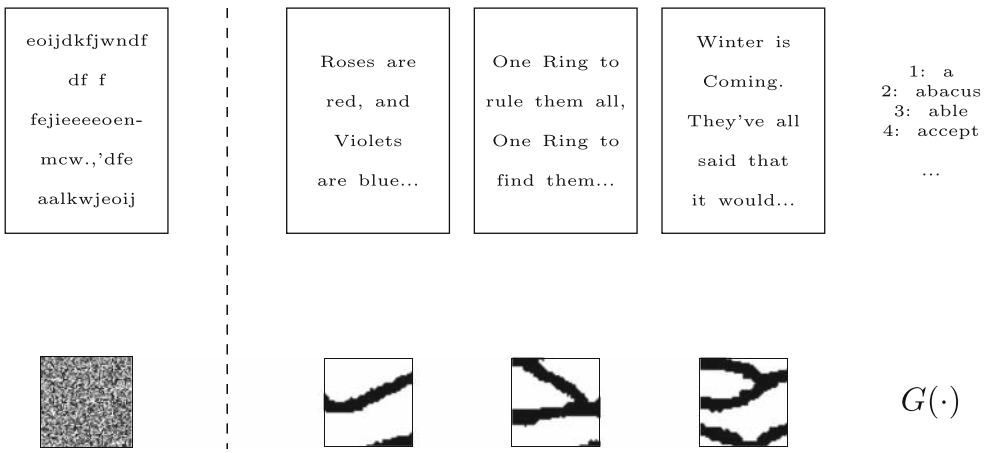


Fig. 1 An index of words provides all *plausible* arrangement of letters (top row). Similarly, a geological parametrization provides all *plausible* realizations of the subsurface (bottom row)

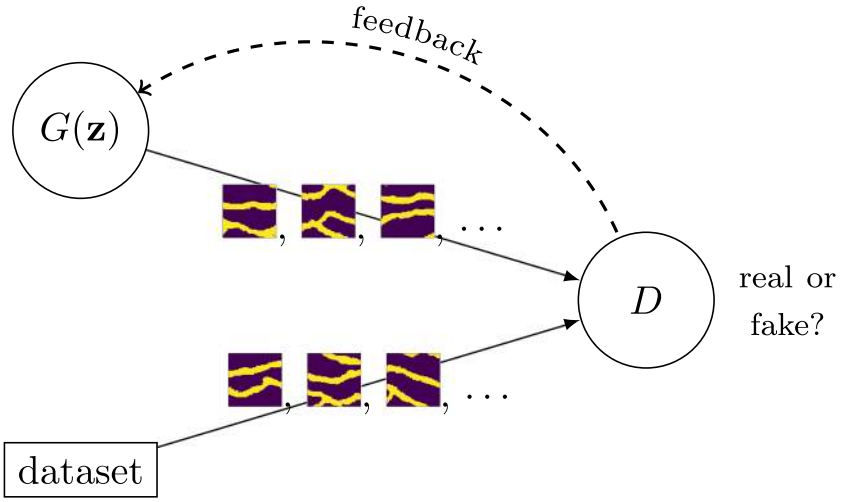


Fig. 2 Generative adversarial networks

discretization. One useful analogy to parametrization is an index of words or a dictionary: Consider the task of inferring the content of a book using only indirect information such as the frequency of letters. A priori, this task would need to consider any possible arrangement of letters however implausible (top row, left of Fig. 1). On the other hand, since most books consist of words, we know that most arrangements are unlikely and can be quickly discarded. The task, although still difficult, is suddenly much easier with the inclusion of this prior information via an index of words (top row, right of Fig. 1). Likewise, consider the task of inferring the subsurface from indirect information such as the oil production history. Without any other information, attempting to deliberately model the subsurface to match the production history would almost certainly result in unrealistic images (bottom row, left of Fig. 1). On the other hand, we know that real subsurface images are not completely random but instead tend to exhibit clear spatial correlations. By using a suitable parametrization of the subsurface, we can embed this information and narrow our search to only the plausible realizations (bottom row, right of Fig. 1), thus reducing the number of simulations required in uncertainty quantification and inversion problems.

Let the random vector $\mathbf{y} \in \mathbb{R}^{n_y}$ represent plausible subsurface images. Parametrization aims to construct a well-behaved function $G\colon \mathbb{R}^{n_z} \to \mathbb{R}^{n_y}$ such that $\mathbf{y} = G(\mathbf{z})$ where $\mathbf{z} \in \mathbb{R}^{n_z}$ (normally $n_z \ll n_y$) is a *latent* random vector with known pre-defined distribution (for example, $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$). Generally, strictly achieving $\mathbf{y} = G(\mathbf{z})$ for complex and high-dimensional $\mathbf{y}$ is hard; hence, many methods settle for replicating simple statistics of $\mathbf{y}$ such as the mean and covariance. For example, in a parametrization based on principal component analysis, G is an affine transformation

$$G(\mathbf{z}) = A\mathbf{z} + b$$

where A, b are fitted so that $G(\mathbf{z})$, $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ preserves the sample mean and covariance estimated from an available dataset $\{y_1, \cdots, y_n\}$ of realizations of $\mathbf{y}$. Note that for nature this parametrization is often too simplistic, resulting in unrealistic realizations that are overly smooth in practice.

In this work, we use a parametrization based on deep neural networks:

$$G(\mathbf{z}) = f_l \circ f_{l-1} \circ \cdots \circ f_1(\mathbf{z}), \quad f_i(x) = \sigma_i(A_i x + b_i) \quad (1)$$

where $\circ$ denotes composition ($f_2 \circ f_1(x) = f_2(f_1(x))$), and σ_i denotes a component-wise non-linearity[1]. This is motivated by the high expressive power of deep neural networks as it is now evident from the state-of-the-art results in computer vision (see e.g. [31, 32] for recent examples). In addition to the more flexible parametrization, instead of training the weights A_i, b_i to preserve only mean and covariance as in principal component analysis, we leverage once more the expressive power of neural networks and let a second neural network learn and decide the relevant statistics directly from the dataset. This is possible due to a recent technique called generative adversarial networks, described below in Section 2.2.

[1]The non-linearity adds expressivity; otherwise, the composition reduces to an affine transformation. Typical choices include $\tanh(x)$, $\max(0, x)$, and $\text{sigmoid}(x)$.

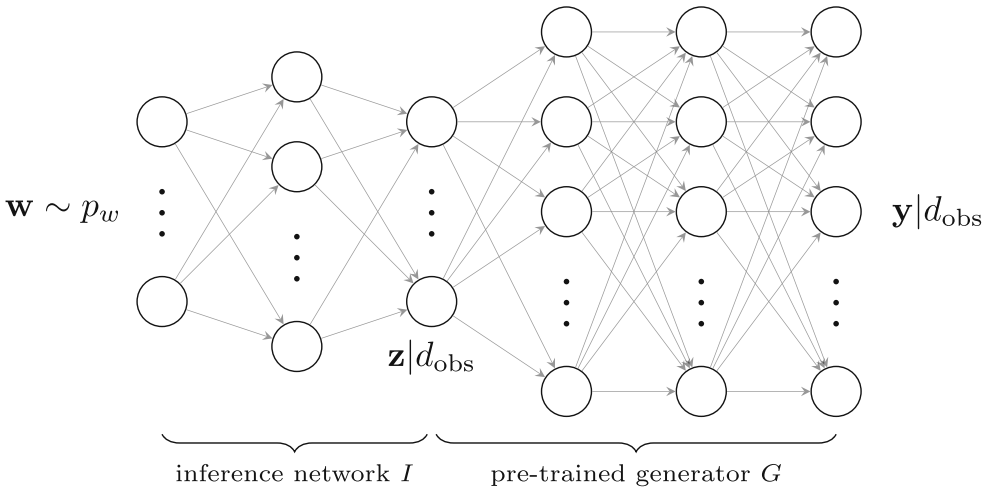


Fig. 3 Illustration of methodology, $G \circ I$

2.2 Generative adversarial networks

We use generative adversarial networks to obtain the (unconditional) parametrization of the geology. This method can be used to obtain a parametrization of a general random vector given a dataset of its realizations. Let the random vector $\mathbf{y} \in \mathbb{R}^{n_y}$ represent the uncertain subsurface property of interest, where n_y is very large (e.g., permeability discretized by the simulation grid). This random vector follows a distribution $\mathbf{y} \sim \mathbb{P}_y$ that is unknown and possibly intractable (e.g., distribution of plausible channelized permeability images). Instead, we are only given a dataset of realizations $\{y_1, \cdots, y_N\}$ of the random vector (e.g., a set of permeability realizations deemed representative of the area under study). Using this training dataset, we aim to find a parametrization for $\mathbf{y}$: Consider now a *latent* random vector $\mathbf{z} \in \mathbb{R}^{n_z}$ with $n_z \ll n_y$ and $\mathbf{z} \sim p_z$ where p_z is manually chosen to be easy to sample from (e.g., a multivariate normal or uniform distribution); and a neural network $G_\theta : \mathbb{R}^{n_z} \to \mathbb{R}^{n_y}$, that we call a *generator*, where θ denotes the weights of the neural network. We aim to determine θ so that $\mathbf{y} = G_\theta(\mathbf{z})$. In other words, let $\mathbb{P}_\theta$ denote the distribution induced by G_θ (i.e., $G_\theta(\mathbf{z}) \sim \mathbb{P}_\theta$), which depends on θ; the goal is to determine θ so that $\mathbb{P}_\theta = \mathbb{P}_y$.

A difficulty with the problem statement above is that $\mathbb{P}_y$ is completely unknown (we only have realizations of $\mathbf{y}$) and $\mathbb{P}_\theta$ is unknown and intractable (even if p_z is simple, G_θ is a neural network with several non-linearities). On the other hand, sampling from these distributions is easy: For $\mathbb{P}_y$, we "sample" by drawing realizations from the training set, assuming the set is large enough to be representative. For $\mathbb{P}_\theta$, we simply sample $\mathbf{z} \sim p_z$ and evaluate $G_\theta(\mathbf{z})$. We therefore have two distributions that we can sample from but we cannot model analytically, and yet we need to optimize $\mathbb{P}_\theta$ to approximate $\mathbb{P}_y$. Informally, we need to teach the generator G_θ to generate *plausible* realizations.

Following this observation, the seminal work in [15] (see also [33]) introduces the idea of using a classifier neural network $D_\psi : \mathbb{R}^{n_y} \to [0, 1]$ called a *discriminator*, with weights ψ, to assess the *plausibility* of generated realizations. The discriminator D_ψ is trained to distinguish between "fake" (from generator) and "real" (from training dataset) realizations, and it essentially outputs a probability estimate. The aim of the generator is then to fool the discriminator (see Fig. 2); hence, the discriminator and the generator are adversaries. The discriminator is trained to solve a binary classification problem by maximizing the following loss:

$$\begin{aligned} \mathcal{L}(\psi, \theta) &:= \mathbb{E}_{\mathbf{y} \sim \mathbb{P}_y} \log D_\psi(\mathbf{y}) + \mathbb{E}_{\tilde{\mathbf{y}} \sim \mathbb{P}_\theta} \log(1 - D_\psi(\tilde{\mathbf{y}})) \\ &\approx \frac{1}{M} \sum_{i=1}^{M} \log D_\psi(y_i) + \frac{1}{M} \sum_{i=1}^{M} \log(1 - D_\psi(G_\theta(z_i)) \end{aligned} \tag{2}$$

which is in essence a binary classification score. The expectations in the expression above are approximated by taking a batch of $M \leq N$ realizations from the training set for the first term, and sampling M realizations $z_1, \cdots, z_M$ from p_z for the second term.

The generator on the other hand is trained to minimize the *same* loss. Thus, an adversarial game is created where G and D optimize the loss in opposite directions,

$$\min_\theta \max_\psi \mathcal{L}(\psi, \theta) \tag{3}$$

In practice, this optimization is performed alternately using stochastic gradient descent, where the gradients with respect to θ and ψ are obtained using automatic differentiation algorithms. The equilibrium is reached when G effectively learns to approximate $\mathbb{P}_y$ and D is $\frac{1}{2}$ in the support of $\mathbb{P}_y$ (coin toss scenario). It is shown in [15] that in the infinite capacity setting, the process minimizes the Jensen-Shannon divergence between $\mathbb{P}_\theta$ and $\mathbb{P}_y$. Once trained, we can discard the discriminator and keep the generator as our parametrization.

Note that the method is very general and directly applicable in practice to all types of geological models including multi-facies and multimodal geology, since minimal assumptions are imposed on $\mathbb{P}_y$ and $\mathbb{P}_\theta$ as we do not need to model them explicitly. We only require realizations $\{y_1, \cdots, y_N\}$ from the unknown target distribution $\mathbb{P}_y$ and the discriminator is in charge of inferring it from the realizations.

Variations of GAN Stability issues with the original formulation of GAN led to numerous works to improve and generalize the method (see e.g. [34–37] and references therein). One line of research generalizes GAN in the framework of integral probability metrics [38]: Given two distributions $\mathbb{P}$ and $\mathbb{Q}$, and a class of real valued functions $\mathcal{D}$, an integral probability metric measures the discrepancy between $\mathbb{P}$ and $\mathbb{Q}$ as follows:

$$\mathrm{d}_{\mathcal{D}}(\mathbb{P}, \mathbb{Q}) = \sup_{D \in \mathcal{D}} \{\mathbb{E}_{\mathbf{y} \sim \mathbb{P}} D(\mathbf{y}) - \mathbb{E}_{\tilde{\mathbf{y}} \sim \mathbb{Q}} D(\tilde{\mathbf{y}})\}$$

Note the slight similarity with Eq. 2. In comparison, this new formulation[2] drops the logarithms and performs the optimization within a class $\mathcal{D} \ni D$ that may be more general, i.e., not necessarily limited to classifier functions. The choice of $\mathcal{D}$ is important and leads to different flavors of GAN. For example, when $\mathcal{D}$ is a ball in a Reproducing Kernel Hilbert Space, $\mathrm{d}_{\mathcal{D}}$ is the Maximum Mean Discrepancy (MMD GAN) [39, 40]. When $\mathcal{D}$ is a set of 1-Lipschitz functions, $\mathrm{d}_{\mathcal{D}}$ is the Wasserstein distance (WGAN) [41, 42]. When $\mathcal{D}$ is a Lebesgue ball, we obtain Fisher GAN [43], and when $\mathcal{D}$ is a Sobolev ball, we obtain Sobolev GAN [44] (see [44, 45] for further discussion). Our unconditional generator is trained using the Wasserstein formulation (see also our recent work [18, 22]).

[2] Actually, this formulation precedes GAN by almost two decades [38], although it is introduced in a different context within probability theory. The connection was drawn recently and led to the numerous works mentioned.

2.3 Conditioning to observations

Given a pre-trained generator G, we aim to generate realizations conditioned to spatial observations (hard data), i.e., find z such that $G(z)$ honors the observations. Let d_{obs} denote the observations and $d(z) = G(z)_{\mathrm{obs}}$ the values at the observed locations given $G(z)$. Under the probabilistic framework, we can formulate the problem as finding z^* that maximizes its posterior probability given observations,

$$z^* = \arg\max_z p(z|d_{\mathrm{obs}}) \tag{4}$$

From Bayes' rule and applying logarithms,

$$p(z|d_{\mathrm{obs}}) \propto p(d_{\mathrm{obs}}|z)p(z)$$
$$-\log p(z|d_{\mathrm{obs}}) = -\log p(d_{\mathrm{obs}}|z) - \log p(z) + \text{const.}$$

For the prior $p(z)$, a natural choice is p_z for which the generator has been trained. In most applications (and in ours), this is the multivariate standard normal distribution, then $p(z) \propto \exp(-\frac{1}{2}\|z\|^2)$. For the likelihood $p(d_{\mathrm{obs}}|z)$, we take the general assumption of i.i.d. Gaussian measurement noise, $p(d_{\mathrm{obs}}|z) \propto \exp(-\frac{1}{2\sigma^2}\|d(z) - d_{\mathrm{obs}}\|^2)$ where σ is the measurement standard deviation. Then, the optimization in Eq. 4 can be written as

$$z^* = \arg\min_z \mathcal{L}(z) \tag{5}$$

where

$$\begin{aligned}\mathcal{L}(z) &:= -\log p(z|d_{\mathrm{obs}}) \\ &\overset{(\times 2\lambda)}{=} \|d(z) - d_{\mathrm{obs}}\|^2 + \lambda\|z\|^2 \\ &= \|G(z)_{\mathrm{obs}} - d_{\mathrm{obs}}\|^2 + \lambda\|z\|^2\end{aligned} \tag{6}$$

where we multiplied everything by $\lambda = \sigma^2$ and discarded the irrelevant constant. One way to draw different conditional realizations is to optimize Eq. 5 repeatedly using a local optimizer and different initial guesses for z, as performed in [20, 21]. Another approach is to sample the full posterior using Markov chain Monte Carlo methods as performed in [19].

3 Conditional generator for geological realizations

As mentioned in Section 2.3, one way to sample multiple realizations conditioned to observations is to solve Eq. 5 repeatedly using local optimization with different initial guesses [20, 21]. However, this approach can be expensive if a large number of realizations need to be continuously generated in deployment, e.g., for uncertainty quantification and inversion problems, and it also may not cover

the full solution space. Another approach is to use Markov chain Monte Carlo methods—assuming the latent vector is of moderate size—to sample the full posterior distribution [19]. Neither approach, however, provides a parametrization of the sampling process. That is, we no longer have a functional relationship $\mathbf{y}_{\text{cond}} = G_{\text{cond}}(\mathbf{w})$, $\mathbf{w} \sim p_w$, where $\mathbf{y}_{\text{cond}}$ denotes the conditional geology and p_w is some fixed distribution.

Here we propose a method to obtain a conditional parametrization for direct and parametric sampling of conditional realizations. The idea is to extend the existing generator $G \circ I =: G_{\text{cond}}$ where I is another neural network—called the *inference network*—that performs the conditioning, as illustrated in Fig. 3. The inference network is trained to sample the Bayesian posterior $p(z|d_{\text{obs}})$ derived in Section 2.3. Let $I_\phi \colon \mathbb{R}^{n_w} \to \mathbb{R}^{n_z}$ where ϕ denotes the weights of the neural network to be determined. I_ϕ maps from yet another random vector $\mathbf{w} \in \mathbb{R}^{n_w}$, $\mathbf{w} \sim p_w$ with manually chosen p_w (we can naturally choose $p_w = p_z$ and $n_z = n_w$), to the conditional latent vector $\mathbf{z}|d_{\text{obs}} \sim p(z|d_{\text{obs}})$. Let q_ϕ denote the distribution density induced by I_ϕ, which depends on ϕ. This density function is unknown and intractable (I_ϕ is a neural network with several non-linearities), but is easy to sample from since it only requires sampling $\mathbf{w} \sim p_w$ and evaluating $I_\phi(\mathbf{w})$. The Kullback-Leibler divergence from $p(\cdot|d_{\text{obs}})$ to q_ϕ gives us

$$\begin{aligned} \mathrm{D}_{\mathrm{KL}}(q_\phi \parallel p(\cdot|d_{\text{obs}})) &= \mathbb{E}_{\mathbf{z}\sim q_\phi} \log \frac{q_\phi(\mathbf{z})}{p(\mathbf{z}|d_{\text{obs}})} \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi} - \log p(\mathbf{z}|d_{\text{obs}}) + \mathbb{E}_{\mathbf{z}\sim q_\phi} \log q_\phi(\mathbf{z}) \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi} \mathcal{L}(\mathbf{z}) + \mathbb{E}_{\mathbf{z}\sim q_\phi} \log q_\phi(\mathbf{z}) + \text{const.} \end{aligned} \tag{7}$$

The first term is the expected loss under the induced distribution q_ϕ, with the loss defined in Eq. 6. It can be approximated as

$$\mathop{\mathbb{E}}_{\mathbf{z}\sim q_\phi} \mathcal{L}(\mathbf{z}) \approx \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}(I_\phi(w_i)) \tag{8}$$

by sampling M realizations $w_1, \cdots, w_M$ from p_w. The second term, however, is more difficult to evaluate since we lack the unknown and intractable q_ϕ. The second term is also called the (negative) entropy of q_ϕ, usually denoted $H(q_\phi) := -\mathbb{E}_{\mathbf{z}\sim q_\phi} \log q_\phi(\mathbf{z})$. Fortunately, there are sample estimators $\hat{H}$ for H, so we can estimate it from a sample $\{z_1, \cdots, z_M\}$, $z_i = I_\phi(w_i)$. We use the Kozachenko-Leonenko estimator [46, 47],

$$\hat{H}(\{z_i, \cdots, z_M\}) = \frac{n_z}{M} \sum_{i=1}^{M} \log \rho(z_i) + \text{const.} \tag{9}$$

where $\rho(z_i)$ is the distance between z_i and its kth nearest neighbor in the sample. A good rule of thumb is $k \approx \sqrt{M}$ [47]. Intuitively, the entropy estimator measures how spread the elements of the sample are. If the entropy term were not present, minimizing Eq. 7 would reduce to finding the maximum a posteriori estimate, instead of sampling the full posterior.

To train the inference network I_ϕ, we minimize $\mathrm{D}_{\mathrm{KL}}(q_\phi \parallel p(\cdot|d_{\text{obs}}))$ (Eq. 7) using automatic differentiation algorithms. Once trained, we obtain our conditional parametrization $G_{\text{cond}} = G \circ I \colon \mathbb{R}^{n_w} \to \mathbb{R}^{n_y}$. Note that we now map from a new source distribution $\mathbf{w} \sim p_w$, although we can simply pick $p_w = p_z$. Sampling conditional realizations is done very efficiently by directly sampling $\mathbf{w} \sim p_w$ and forward-passing through $G \circ I$.

We summarize the training steps of the inference network in Algorithm 1. Note that we show a simple gradient descent update (line 7); however, it is more common to use dedicated schemes for neural networks such as Adam [48] or RMSProp [49].

Note that the inference network I is relatively easy to train compared with the generator G which is based on GAN. The network I is also usually small and the relative increase in evaluation cost of the composition $G \circ I$ is not significant. We find this to be the case in our experiments.

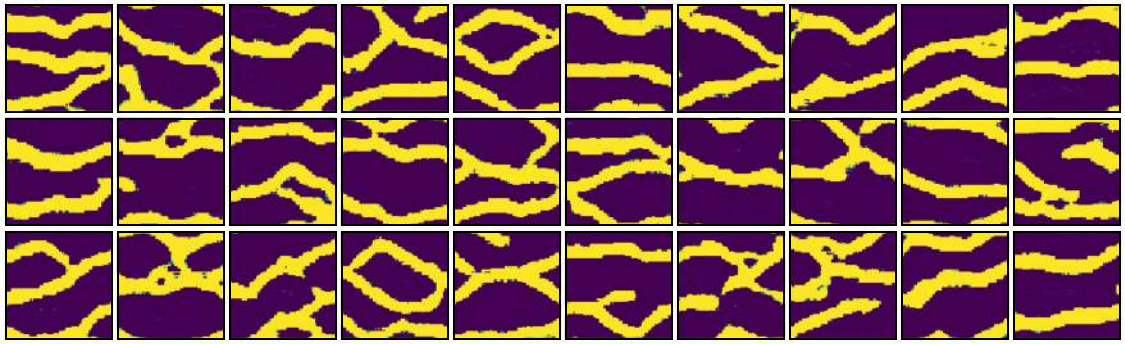
Algorithm 1 Inference network I_ϕ training.

Require: Negative log-posterior $\mathcal{L}(z) = -\log p(z|d_{\text{obs}})$. In our case (Equation 6), $\mathcal{L}(z) = \|G(z)_{\text{obs}} - d_{\text{obs}}\|^2 + \lambda\|z\|^2$, batch size M, learning rate η, source distribution p_w (usually equal to p_z).

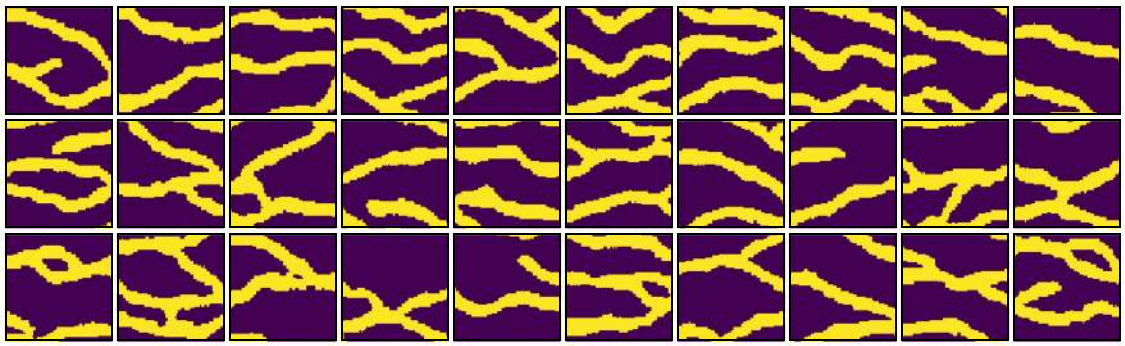
1: **while** ϕ has not converged **do**
2: Sample $\{w_1, \cdots, w_M\} \sim p_w$
3: Get $\{z_1, \cdots, z_M\}$, $z_i = I_\phi(w_i)$
4: Get $\{\rho_1, \cdots, \rho_M\}$, ρ_i = distance from z_i to its k^{th} nearest neighbor
5: $\nabla_\phi \mathbb{E}\mathcal{L} \leftarrow \frac{1}{M} \sum_{i=1}^{M} \nabla_\phi \mathcal{L}(z_i)$
6: $\nabla_\phi \hat{H} \leftarrow \frac{n_z}{M} \sum_{i=1}^{M} \nabla_\phi \log \rho_i$
7: $\phi \leftarrow \phi - \eta(\nabla_{\phi\mathbb{E}\mathcal{L}} - \nabla_{\phi\hat{H}})$
8: **end while**

4 Numerical experiments

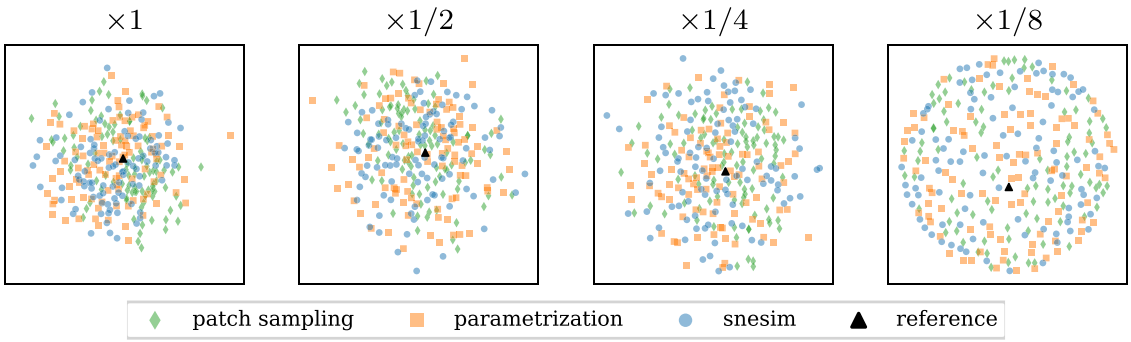
We train generative adversarial networks to obtain a parametrization of binary channelized subsurface images based on the benchmark image of [30]. We then condition the parametrization for a variety of configurations using



(a) $G(\mathbf{z})$



(b) `snesim`



(c) Multidimensional scaling visualization.

Fig. 4 Unconditional realizations. **a** $G(z)$. **b** `snesim`. **c** Multidimensional scaling visualization

our method described in Section 3. Finally, we also include in Appendix C a side experiment as a sanity check of the proposed method where we train neural samplers for simple mixture of Gaussians. All our numerical experiments are implemented using PyTorch [50], an open-source Python package for automatic differentiation that provides tools to facilitate the construction and training of neural network models. Our implementation code is available in our repository[3].

[3] https://github.com/chanshing/geocondition

4.1 Unconditional parametrization

We train a generator $G\colon \mathbb{R}^{30} \to \mathbb{R}^{64\times 64}$ using a dataset of 1000 realizations of size 64×64 of binary channelized subsurface images. The realizations were obtained using the `snesim` algorithm [30, 51] provided within the Stanford geostatistical modeling software [52], using the benchmark image from [30] as the reference image[4]. A few `snesim` realizations are shown in Fig. 4b.

[4] Also referred to as a *training image* in the geostatistics literature, although we avoid the term so that it is not confused with the images of the training set used to train the neural network.

Table 1 Unconditional realizations. ANODI scores (inconsistency/diversity)

	×1	×1/2	×1/4	×1/8
Patch sampling	0.0220/0.0360	0.0594/0.0900	0.2329/0.3098	0.6055/0.6527
G	0.0286/*0.0385*	0.0671/*0.1002*	*0.2500*/0.3239	*0.6133*/0.6596
`snesim`	*0.0279*/0.0353	*0.0648*/0.0934	0.2551/*0.3389*	0.6165/*0.6608*

4.1.1 Architecture design

The latent vector size $n_z = 30$ was chosen using principal component analysis as a heuristic, where the number of eigencomponents required to retain 75% of the variance is used as a reference. This results in a dimensionality reduction of two orders of magnitude—from 64×64 to 30. The latent vector is sampled from the standard normal distribution, $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Since the data is binary, it is reasonable to embed this knowledge into the neural network design using a suitable non-linearity in the output layer of the neural network. We use $\sigma = \tanh$, where we adopt 1 to denote channel material, and -1 to denote background material. Note that attempting to use a hard threshold here would render G discontinuous and introduce issues in the training. It can also be an issue in inversion problems during deployment. The rest of the neural network architecture (shapes of A_i, b_i, non-linearity σ_i, number of layers l, etc.—see Eq. 1) is designed according to the template provided in [34]. This template is the result of experimentation, heuristics, and experience. In particular, an important design choice is the use of convolutional layers. These are sparse matrices A_i that follow a certain structure that makes them effective for spatial data. A brief description of convolutional layers is provided in Appendix E. Further details of the generator architecture and training is given in Appendix A.1.

4.1.2 Quality assessment

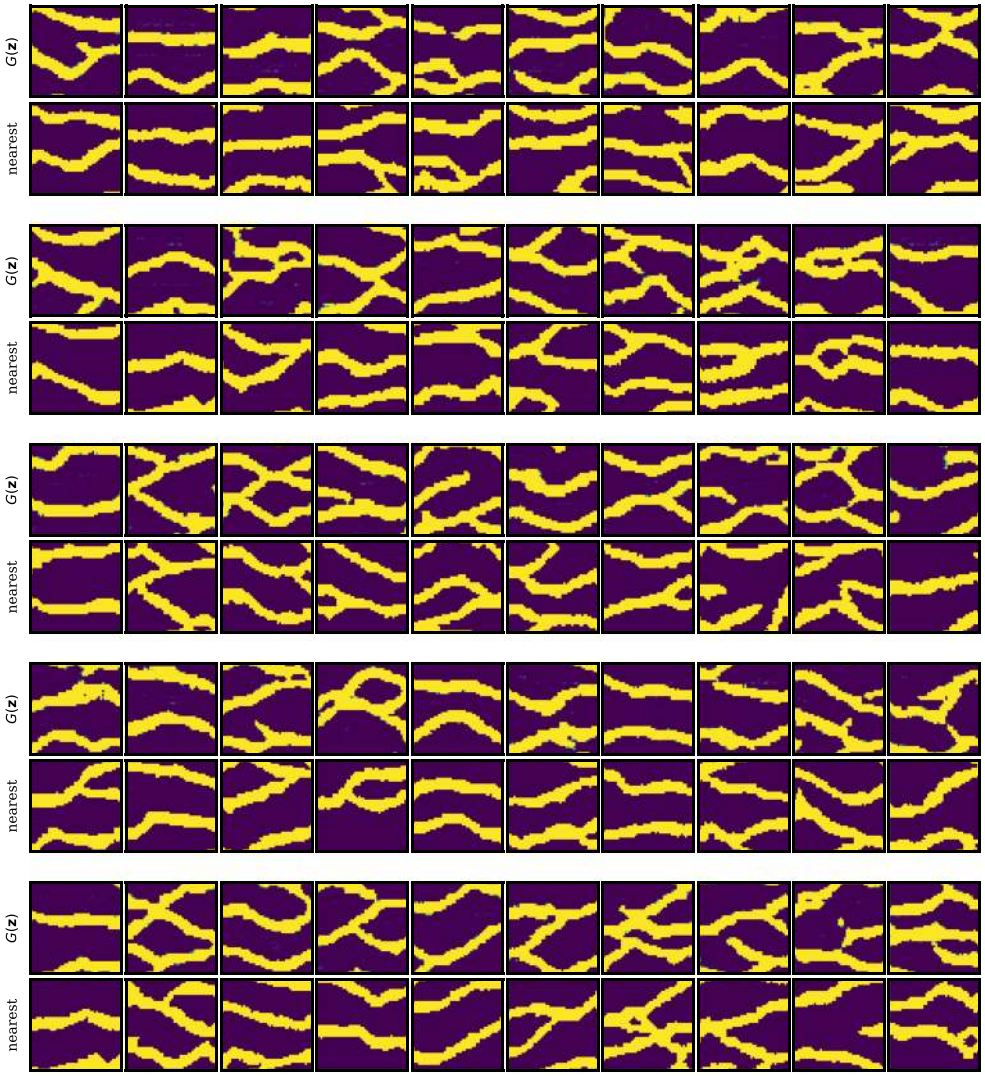
Realizations generated by the parametrization G are shown in Fig. 4a. We also show `snesim` realizations in Fig. 4b. We can already see from the figure that the parametrization is at least visually competitive with previous parametrization methods. The realizations of the parametrization are virtually indistinguishable from `snesim`, recreating crisp and clear channels from the reference (note that no thresholding has been performed). We next assess the results quantitatively.

Previous works have assessed the effectiveness of GAN-based parametrization using a variety of tools. Two-point probability functions, morphological measures, and effective porosity were assessed in [16]. Two-point probability and cluster functions, and fractions of facies were assessed in [19]. In our previous work [18], we assessed the effectiveness of the parametrization for preserving high-order flow statistics in uncertainty quantification. Here we add to the assessment using the method of analysis of distances (ANODI) [53] which captures multipoint statistics, providing a more reliable measure of quality for complex data where two-point statistics are insufficient. We also apply multidimensional scaling for visualization.

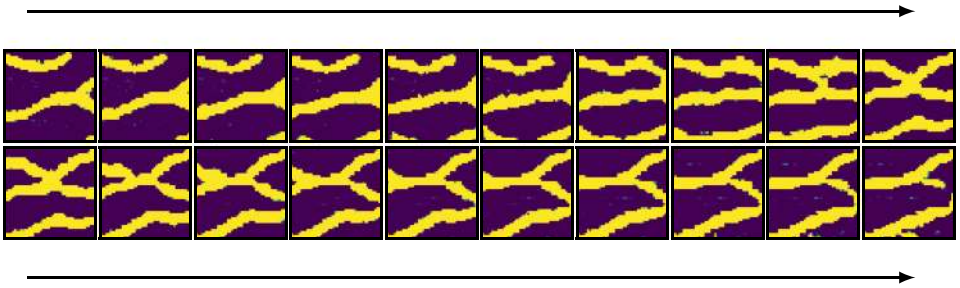
The ANODI method aims to capture multipoint statistics by comparing multipoint histograms at different resolutions. It computes an inconsistency score (how well it matches the statistics of the reference image) and a diversity score (variability between realizations)—therefore, we want low inconsistency and high diversity. Multidimensional scaling is a method that aims to project a set of high-dimensional objects to low dimensions in a way that preserves the distances between the objects. Although some information may be lost in the projection, the method provides a useful way of visualizing high-dimensional objects (e.g., images) using a scatter-plot. The notion of distance between images, as adopted in [53], is the Jensen-Shannon divergence between multipoint histograms of patterns extracted from the images within a window size.

We perform the analysis at four resolutions: ×1 (original), ×1/2, ×1/4, and ×1/8 resolution (i.e., at 64×64, 32×32, 16×16, and 8×8). We use a window size of 4×4 and sets of 100 realizations for the analysis. Importantly, note that the `snesim` realizations are fresh realizations, i.e not from the training dataset: Ultimately, we aim to plug the parametrization into a reservoir simulator, for which we are assuming that the parametrization replicates the data generating process. Therefore, the comparison is made against out-of-sample realizations to see if the parametrization generalizes. For multidimensional dimensional scaling, we use the SMACOF [54] algorithm with 300 iterations and tolerance of 10^{-3}.

For the analysis, we need to binarize the realizations generated by the parametrization which are continuous by design. For this, we use Otsu's thresholding method [55]. We also apply small object removal processing on the images to remove possible isolated pixels. Note that we



(a) Nearest neighbors for 50 generated realizations.

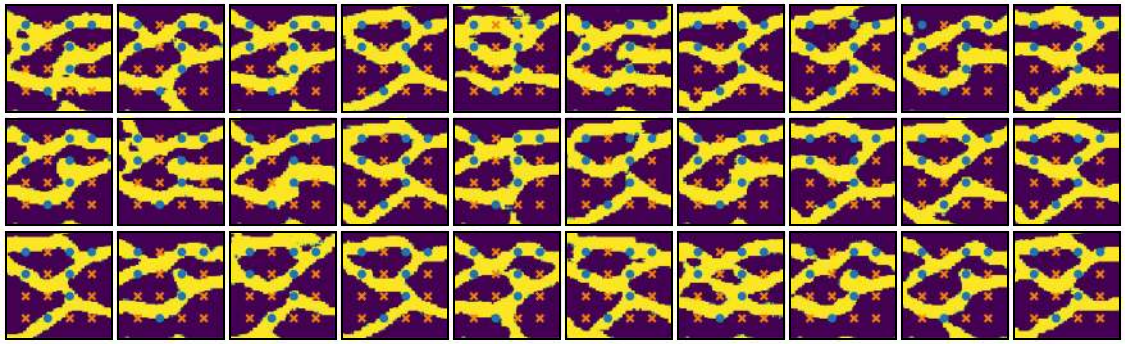


(b) Interpolation in the latent space.

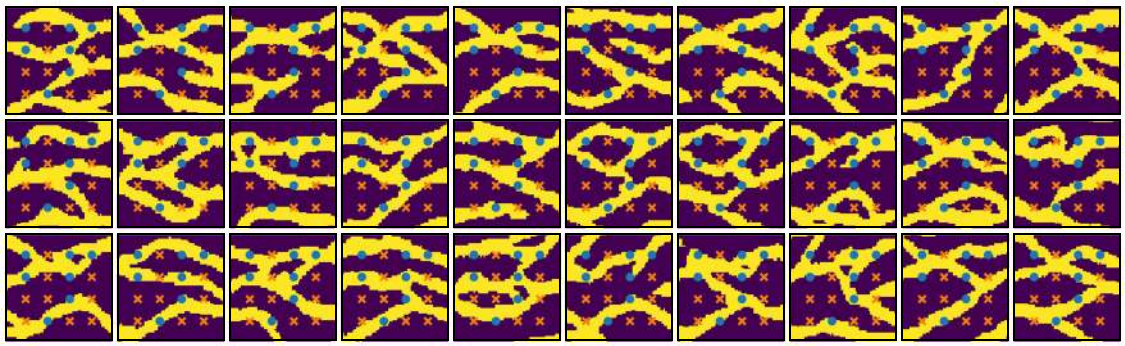
Fig. 5 Assessment of memorization of G. **a** Nearest neighbors for 50 generated realizations. **b** Interpolation in the latent space

do *not* apply thresholding nor any other post-processing on the displayed images in this work. Finally, since it can be difficult to gauge the differences in the ANODI scores, we include "patch sampling" method (i.e., drawing patches of 64×64 from the reference image) into the analysis to serve as a third point of comparison.

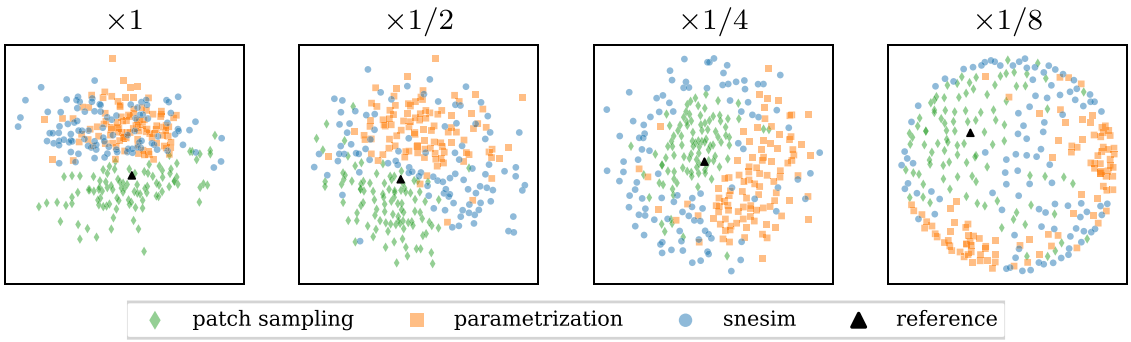
We show the ANODI scores and multidimensional scaling visualizations in Table 1 and Fig. 4c, respectively.



(a) $G \circ I(\mathbf{w})$



(b) `snesim`



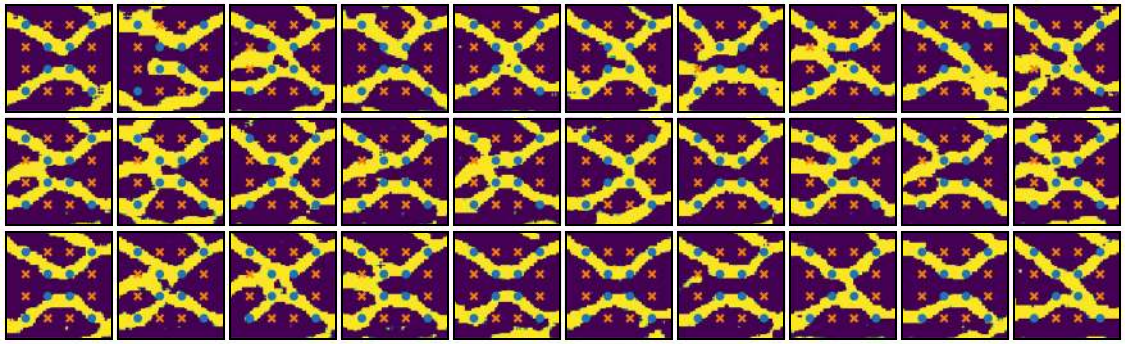
(c) Multidimensional scaling visualization.

Fig. 6 Example A. Conditional realizations. **a** $G \circ I(\mathbf{w})$. **b** `snesim`. **c** Multidimensional scaling visualization

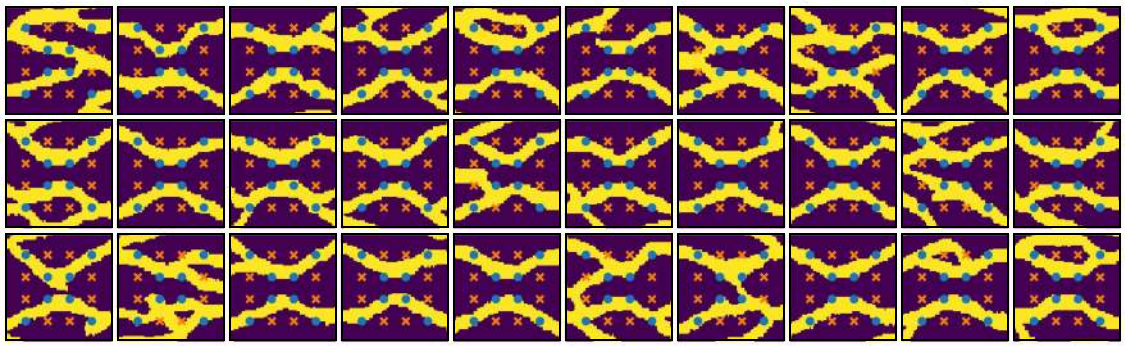
The patch sampling procedure understandably produces the highest consistency (lowest inconsistency), although it is also slightly less diverse. Regarding the parametrization, we find that the scores for `snesim` and G are relatively very close across all resolutions, suggesting that G effectively learned to replicate the data generating process. The multidimensional scaling visualization in Fig. 4c further supports this result, showing a very good overlap in the scatter-plots of `snesim` and G. The scatter plots are also well spread and centered around the reference image, verifying the good performance of both methods.

4.1.3 Memorization

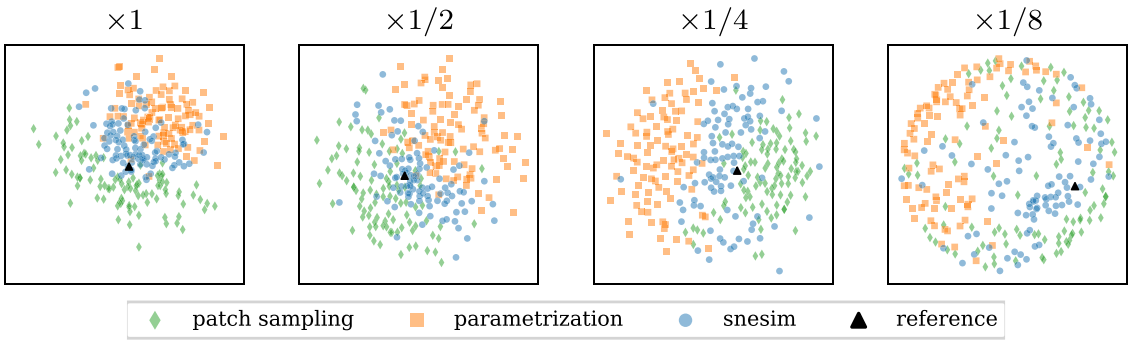
To verify that the parametrization is not simply memorizing the training dataset, we find the nearest neighbor in the



(a) $G \circ I(\mathbf{w})$



(b) `snesim`

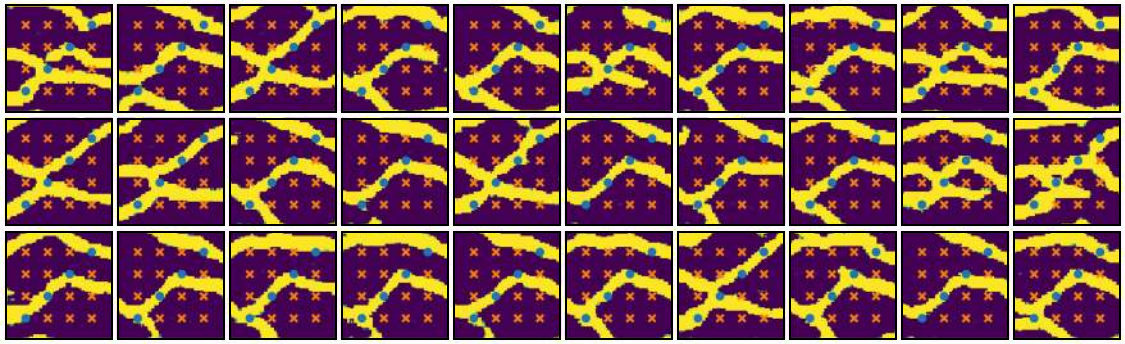


(c) Multidimensional scaling visualization.

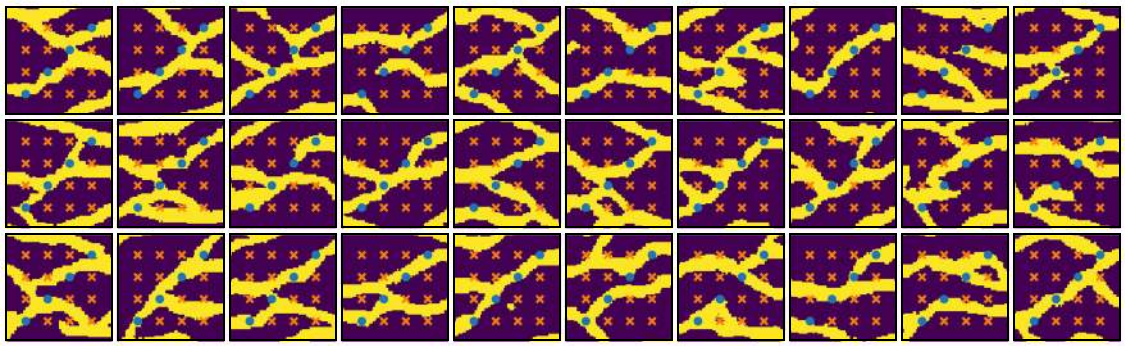
Fig. 7 Example B. Conditional realizations. **a** $G \circ I(\mathbf{w})$. **b** `snesim`. **c** Multidimensional scaling visualization

dataset for each of 50 generated realizations. To further verify that the generator is not simply learning trivial transformations, we data-augment our training dataset with horizontal and vertical flips, as well as 10 and −10 degrees rotation and shearing (with reflection filling at the boundaries). This results in 35 additional variations for each image in the dataset. Finally, to capture small translations, we apply a Gaussian blur to the images before computing the Euclidean distance.

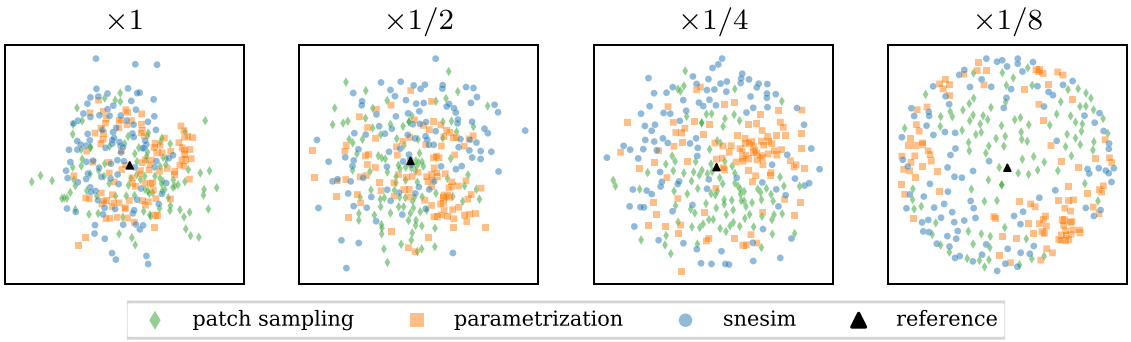
The 50 realizations along with the nearest neighbors are shown in Fig. 5a. We see that there is no perfect match despite the heavy data augmentation, verifying that the parametrization is capable of generating novel realizations that are not mere rotations, translations, shearing and flips of images from the dataset. The lack of memorization can be justified by the fact that the generator never has direct access to the training dataset (see Eq. 2). Instead, the generator only obtains indirect information about the dataset via the



(a) $G \circ I(\mathbf{w})$



(b) `snesim`



(c) Multidimensional scaling visualization.

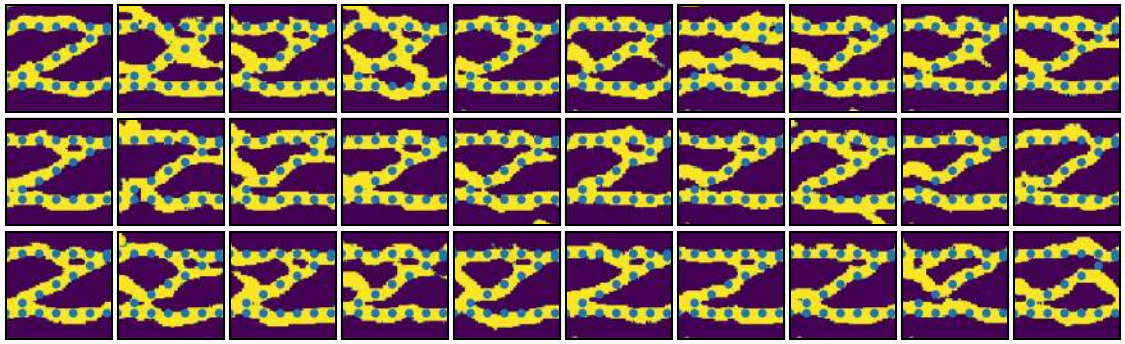
Fig. 8 Example C. Conditional realizations. **a** $G \circ I(\mathbf{w})$. **b** `snesim`. **c** Multidimensional scaling visualization

discriminator (i.e., through its gradients). This is similar to principal component analysis where the parametrization is only informed about the dataset covariance. In the case of GAN, the relevant dataset statistics are automatically discovered and informed by the discriminator.

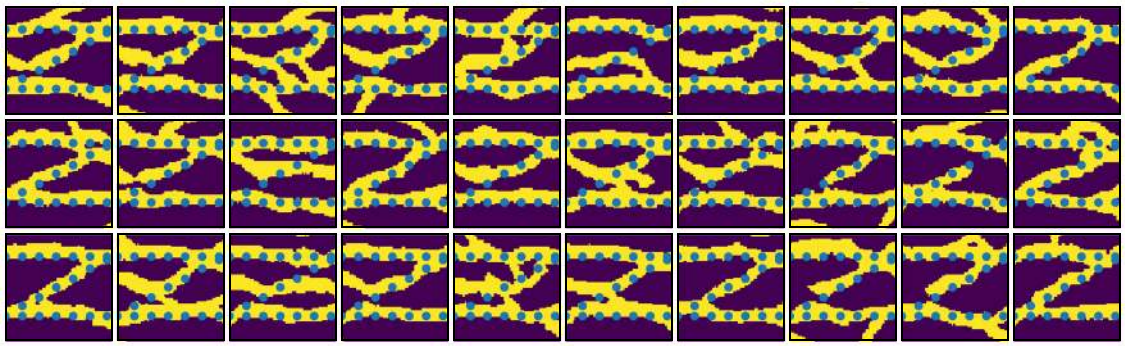
Finally, we provide a further verification by performing an interpolation in the latent space in Fig. 5b. If G is simply memorizing the dataset, we would expect to see sudden jumps from one image of the dataset to another, with implausible images in between as we interpolate in the latent space. We instead effectively find smooth transitions between plausible outputs. The smoothness is also justified by the fact that G is continuous and piecewise differentiable by design (see Eq. 1). Note also that the smoothness is critical in practice for efficient exploration of the solution space during deployment, e.g., for inversion and uncertainty quantification tasks.

4.2 Conditional parametrization

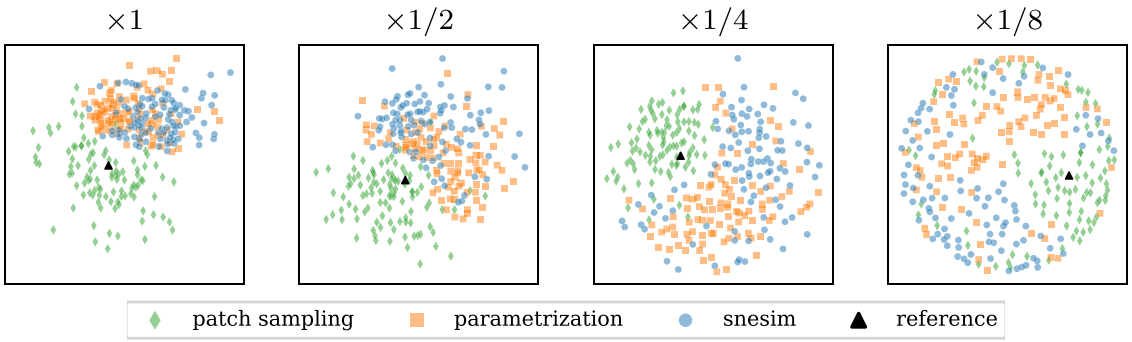
We now obtain conditional generators for 9 conditioning configurations, ranging from 16 to 49 spatial observations



(a) $G \circ I(\mathbf{w})$



(b) `snesim`



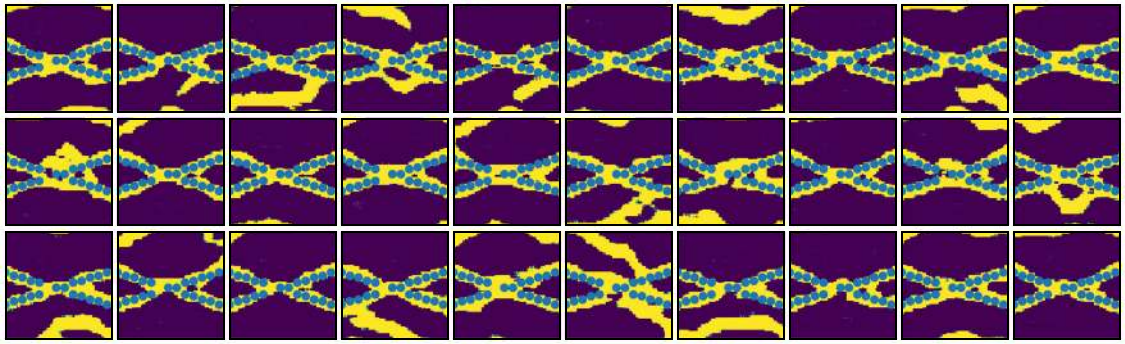
(c) Multidimensional scaling visualization.

Fig. 9 Example D. Conditional realizations. **a** $G \circ I(\mathbf{w})$. **b** `snesim`. **c** Multidimensional scaling visualization

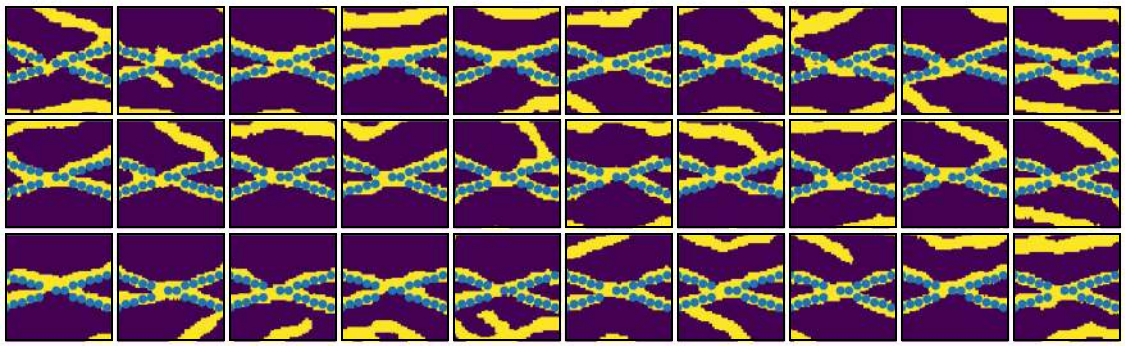
(hard data). The configurations are detailed in Table 4. These indicate the presence or absence of channels at different locations of the domain. For each configuration d_{obs}, we derive the Bayesian posterior $p(z|d_{\text{obs}})$ as described in Section 2.3, and train an inference network to sample from it as described in Section 3. We assume $\lambda = 0.1$ in Eq. 6 in all our test cases. Note that we use a Gaussian likelihood function in the Bayesian formulation, although one could also consider the binomial distribution for binary data. Also note that due to the Bayesian framework adopted here—i.e., that the observations are noisy—we cannot fully guarantee that conditioning is strictly honored, but only that it is honored with high probability.

4.2.1 Architecture design—inference network

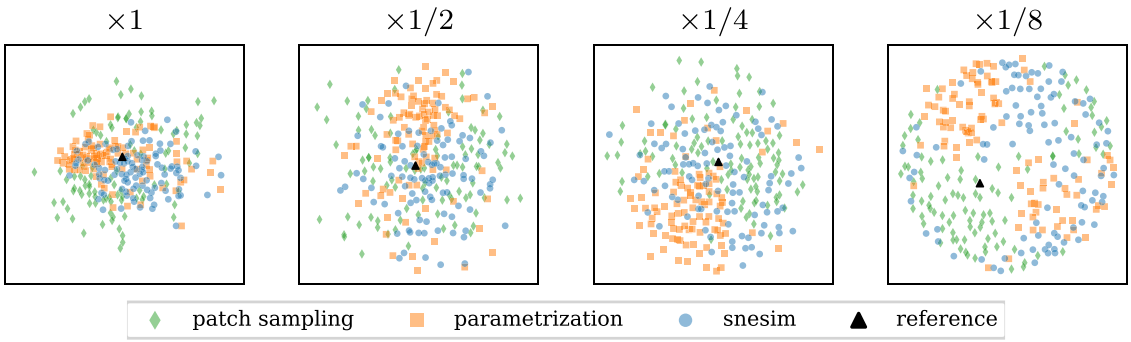
The inference network $I_\phi : \mathbb{R}^{30} \to \mathbb{R}^{30}$ is a fully connected neural network with several layers. We naturally use $n_w = n_z = 30$ and $p_w = p_z = \mathcal{N}(\mathbf{0}, \mathbf{I})$ (so that if no conditioning were present, I_ϕ should learn a distribution-preserving



(a) $G \circ I(\mathbf{w})$



(b) `snesim`



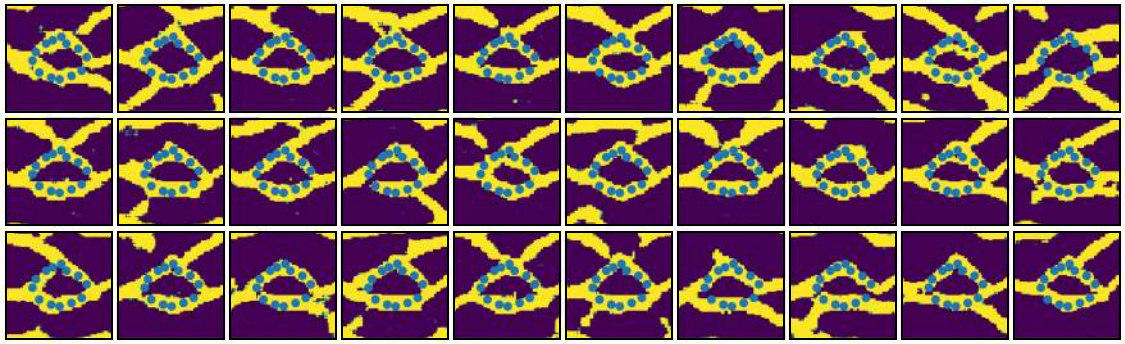
(c) Multidimensional scaling visualization.

Fig. 10 Example E. Conditional realizations. **a** $G \circ I(\mathbf{w})$. **b** `snesim`. **c** Multidimensional scaling visualization

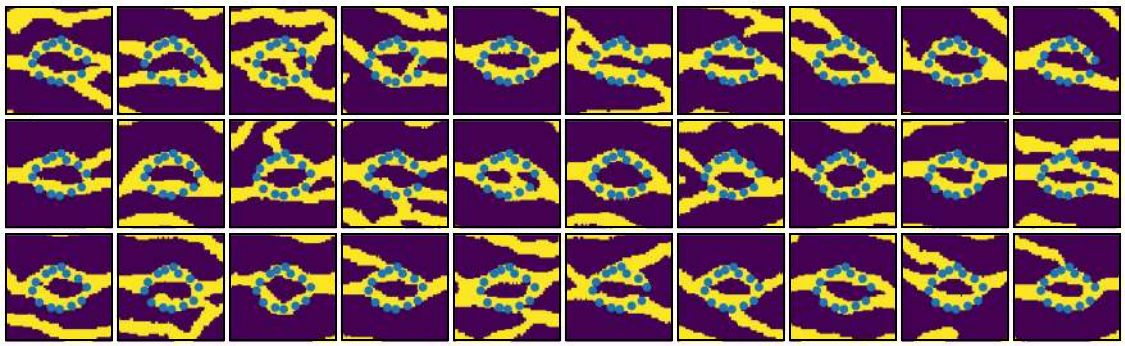
function such as the identity function). To simplify the presentation, we perform minimal hyperparameter tuning—that is, we use the same neural network architecture and optimization parameters for all 9 test cases. In practice, one should perform hyperparameter optimization for each problem at hand. For the non-linearity, we use scaled exponential linear units [56]. No non-linearity is applied in the output layer. Further details of the architecture and training are given in Appendix A.2. Once I is trained, we use $G \circ I$ to generate conditional realizations.

4.2.2 Quality assessment—conditional realizations

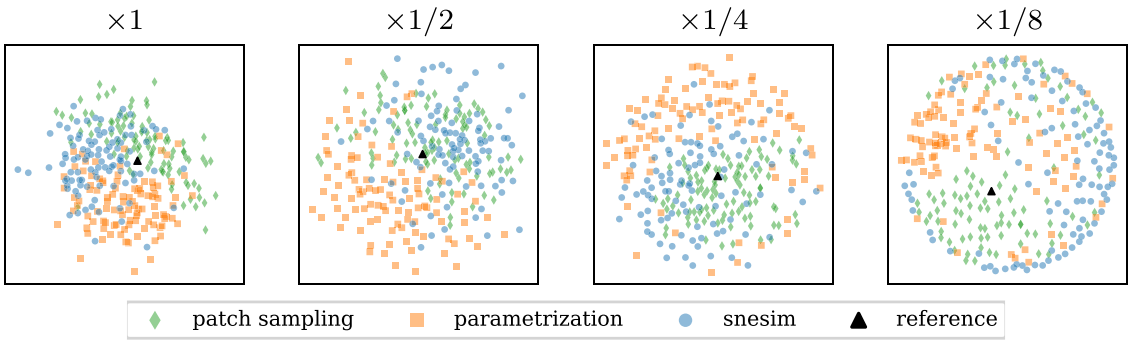
We show conditional realizations generated by $G \circ I$ for each test case in Figs. 6, 7, 8, 9, 10, 11, 12, 13, and 14. We also include conditional realizations obtained using `snesim` for



(a) $G \circ I(\mathbf{w})$



(b) `snesim`

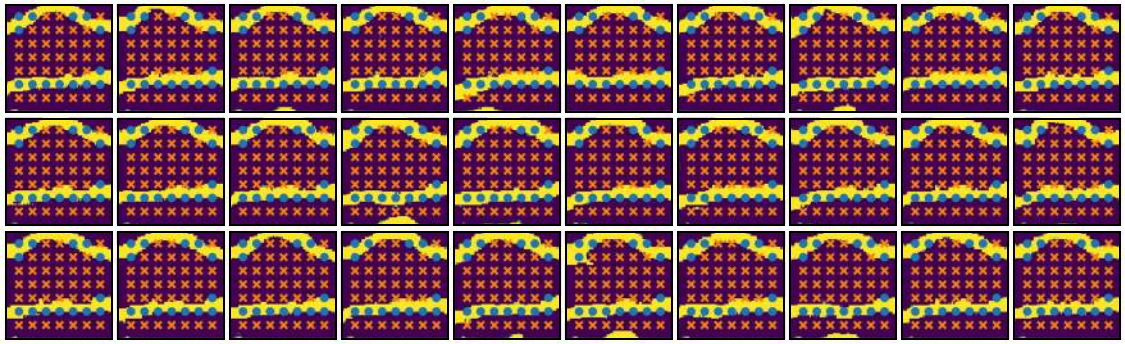


(c) Multidimensional scaling visualization.

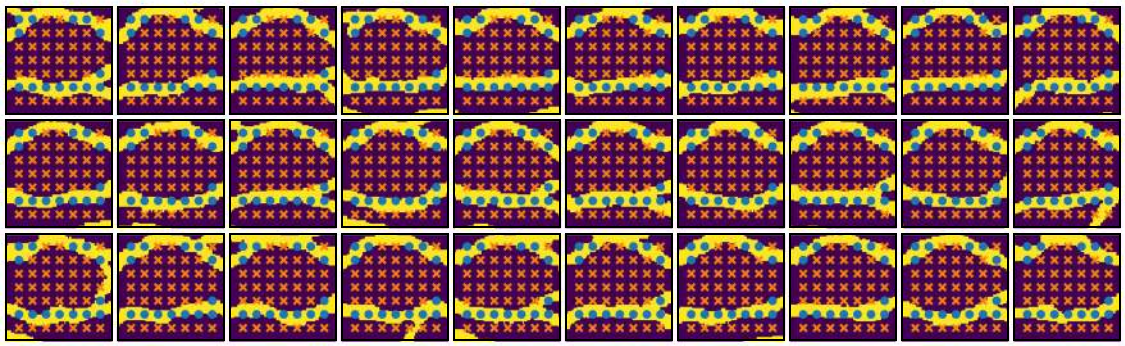
Fig. 11 Example F. Conditional realizations. **a** $G \circ I(\mathbf{w})$. **b** `snesim`. **c** Multidimensional scaling visualization

comparison. Over the images, we indicate the conditioning using blue dots to denote channel material and orange crosses to denote background material. Overall, we observe that $G \circ I$ generates good conditioning results maintaining the plausibility of the realizations. We also show in Fig. 15 the output of the inference network I for each test case to visualize the distribution change (for no conditioning, the distribution is normal). Since it is cumbersome to visualize the distribution for the 30 components of $\mathbf{z}$, we show pairwise scatter plots only for the first and last two components.

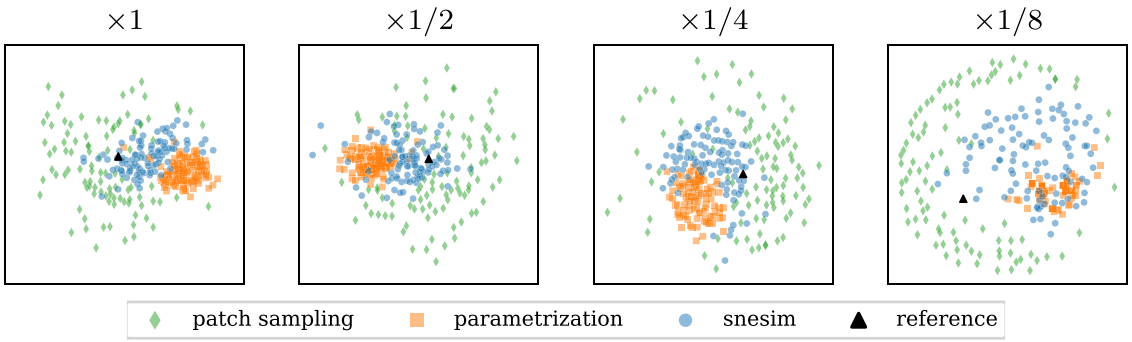
Assessment using analysis of distances We perform a quantitative assessment as in the unconditional case, using the ANODI method and multidimensional scaling on sets of 100 realizations. We keep the "patch sampling" method



(a) $G \circ I(\mathbf{w})$



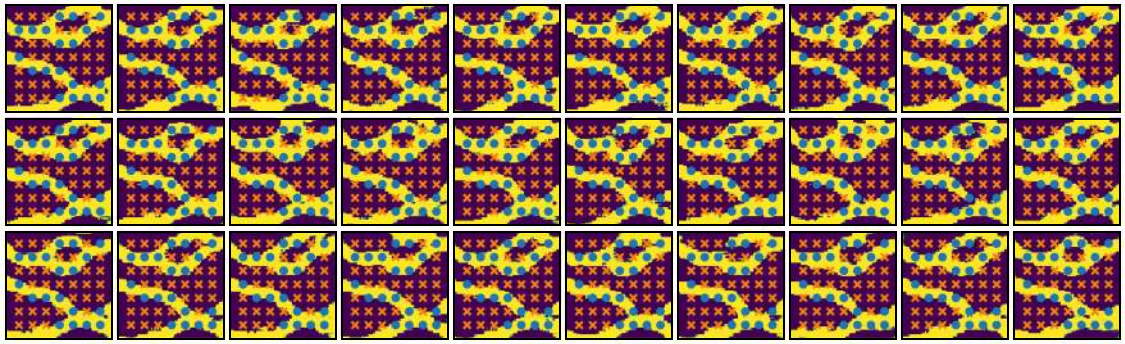
(b) `snesim`



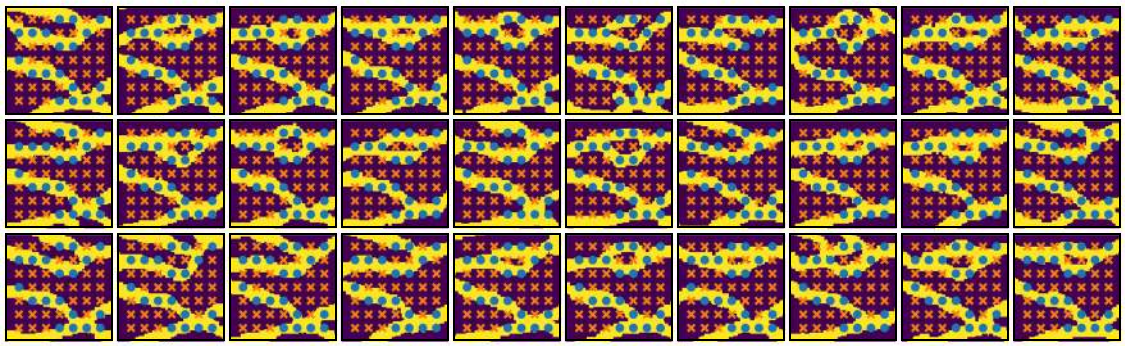
(c) Multidimensional scaling visualization.

Fig. 12 Example G. Conditional realizations. **a** $G \circ I(\mathbf{w})$. **b** `snesim`. **c** Multidimensional scaling visualization

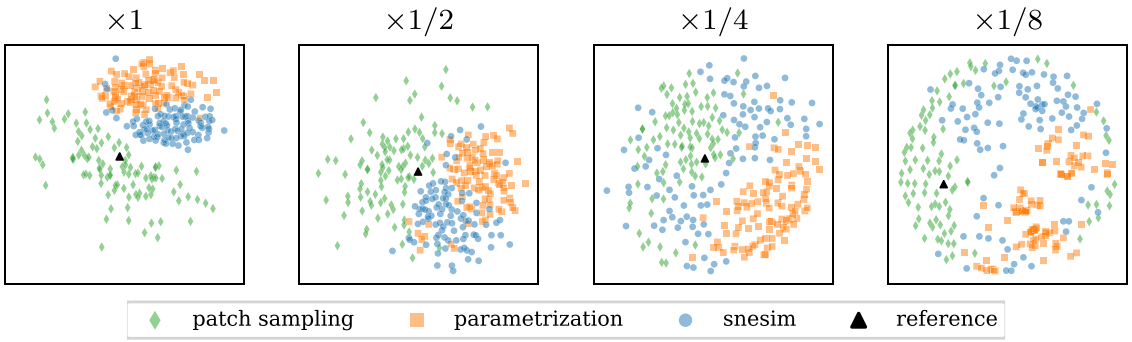
for the multidimensional scaling visualization. The results are shown in Figs. 6, 7, 8, 9, 10, 11, 12, 13, and 14 and Table 2. In terms of the ANODI scores, we find that whenever one method generates more plausible images (lower inconsistency), it also tends to be less diverse, and vice versa—this is the usual trade-off in image synthesis. Overall, we find that `snesim` produces more diverse realizations whereas $G \circ I$ emphasizes on plausibility. This is reasonable since $G \circ I \subset G$, i.e., an output of $G \circ I$ is an output of G; therefore, the conditional realizations cannot deviate too much from the reference spatial statistics. Also for this reason, we find that the outputs of $G \circ I$ and `snesim` are most different when the conditioning statistics are in less agreement with the reference spatial statistics. This is evident in Fig. 14, and to a lesser extent Figs. 6 and 8. In Fig. 8 we enforce a



(a) $G \circ I(\mathbf{w})$



(b) `snesim`

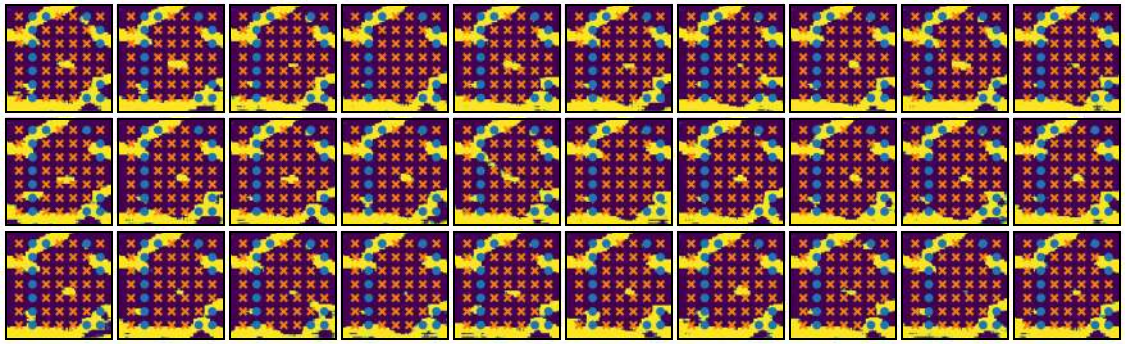


(c) Multidimensional scaling visualization and.

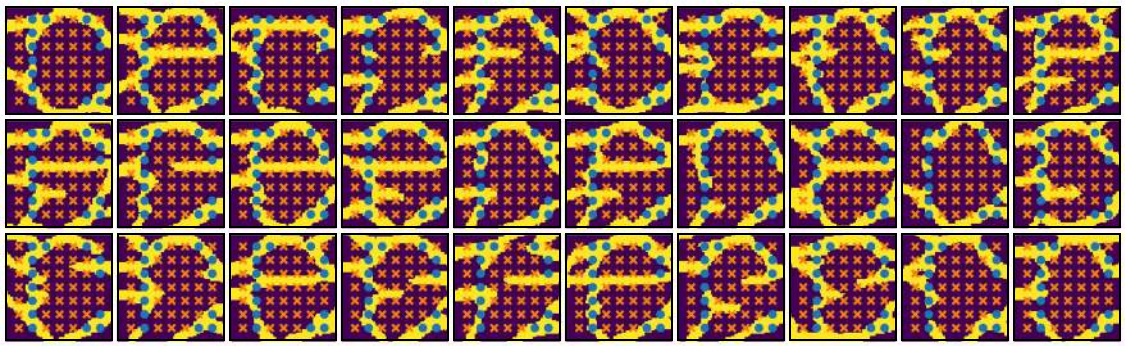
Fig. 13 Example H. Conditional realizations. **a** $G \circ I(\mathbf{w})$. **b** `snesim`. **c** Multidimensional scaling visualization

diagonal channel, finding that $G \circ I$ generates plausible but noticeably less diverse realizations compared to `snesim`. The difference is more pronounced in Fig. 14 where we densely enforce vertical channels and find a failure case for $G \circ I$, whereas `snesim` can handle this case despite the implausibility of this conditioning (there are no vertical channels in the reference image). In other words, if the conditioning is in far disagreement with the reference spatial statistics, effective conditional parametrization may be difficult since G is tied to the reference statistics. In the `snesim` algorithm, deliberate conditioning and diversity can be achieved regardlessly since the conditioning is trivially imposed and the stochasticity is intrinsic to the synthesis process.

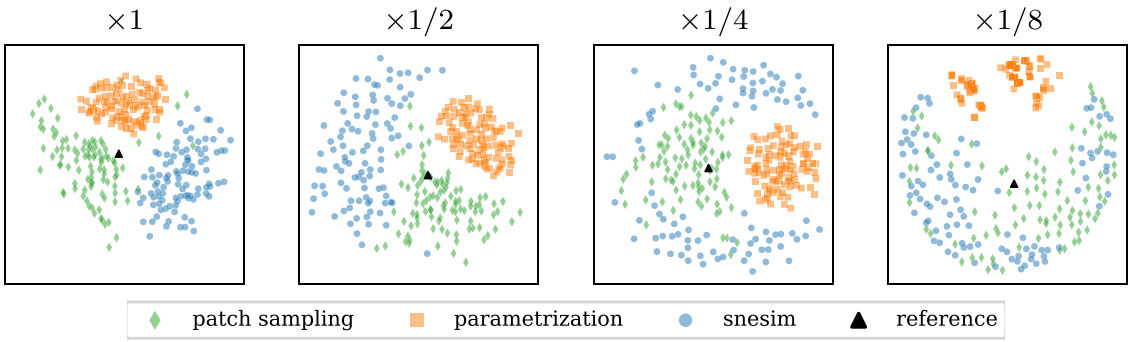
Finally, when the conditioning is in good agreement with the reference spatial statistics as in the remaining cases, we



(a) $G \circ I(\mathbf{w})$



(b) `snesim`



(c) Multidimensional scaling visualization.

Fig. 14 Example I. Conditional realizations. **a** $G \circ I(\mathbf{w})$. **b** `snesim`. **c** Multidimensional scaling visualization

observe that $G \circ I$ generates realizations that are visually comparable with `snesim`. Note that in practice, we always aim to use a reference image whose spatial statistics are in good agreement with the spatial observations; otherwise, the reference image may not be representative of the area under study. Also note that although we compare our method against a multipoint geostatistical simulator, our emphasis is on parametrization. Lastly, we mention that the present results could be further improved with hyperparameter tuning for each individual test case.

Assessment using the discriminator We demonstrate an alternative approach to assess the quality of the generated realizations using the discriminator D that is made available after training generative adversarial networks to obtain G. Recall that the discriminator outputs a score that estimates the probability of a realization being "real" (see Section 2.2), with higher scores corresponding to higher probability. We can therefore use the discriminator to assess the quality of the generated realizations. We evaluate the discriminator on the same sets of 100 realizations used

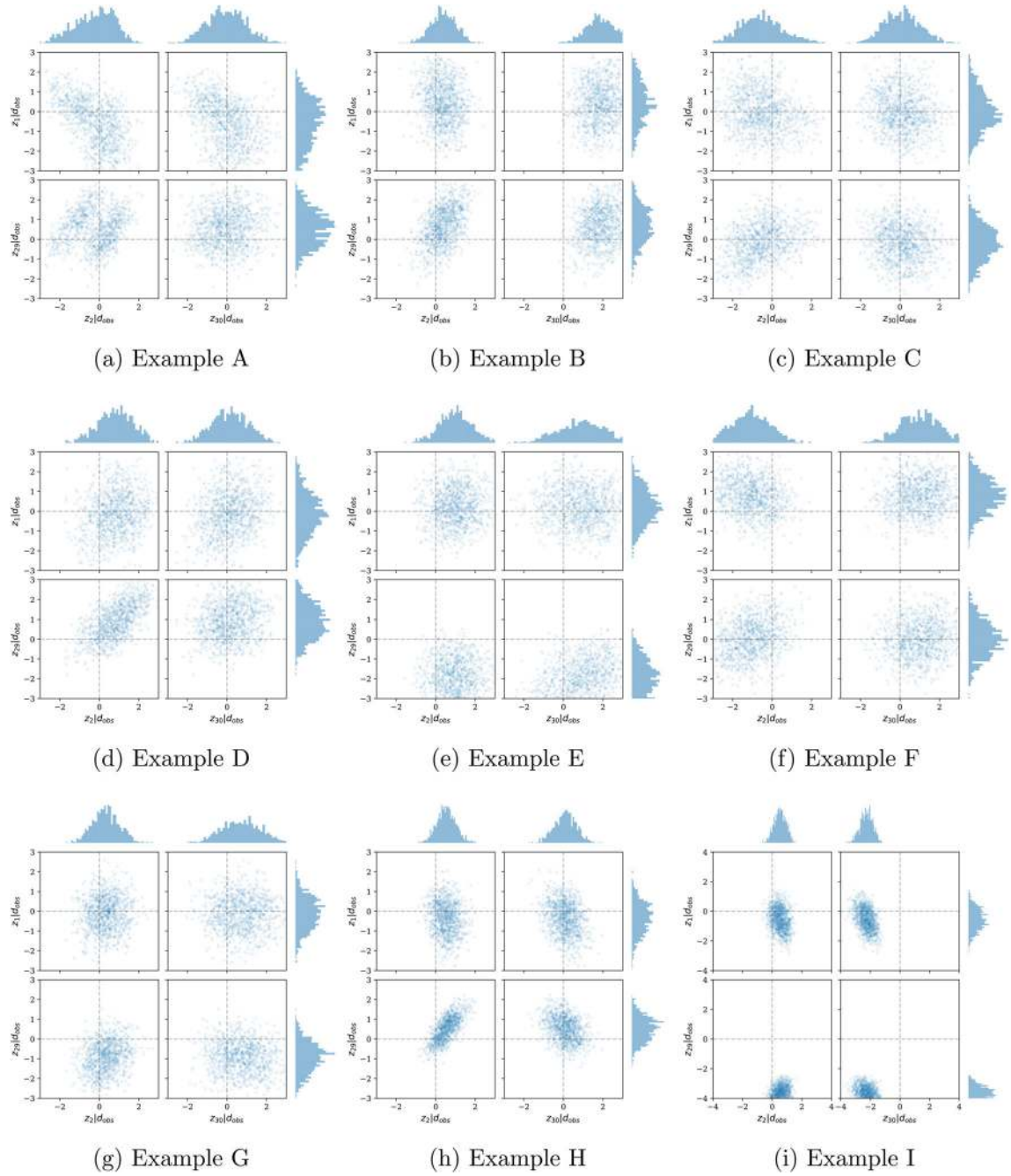


Fig. 15 Visualizing the distribution of $\mathbf{z}|d_{\text{obs}}$. **a** Example A. **b** Example B. **c** Example C. **d** Example D. **e** Example E. **f** Example F. **g** Example G. **h** Example H. **i** Example I.

before in the ANODI assessment, and plot the histogram of the scores in Fig. 16. Overall, we verify that this assessment at least arrives at the same qualitative conclusions as the ANODI assessment. For a quantitative summary report, one can consider summary statistics of the scores such as the mean and variance as measures of plausibility and diversity, respectively. Another option is to report the Jensen-Shannon divergence with respect to some reference histogram.

5 Related work

There is increasing interest in applying deep learning techniques in geological applications to leverage recent advances in the field as well as the increasing availability of data and computational resources that make these techniques effective. In particular, we expect to see more applications of generative adversarial networks

Table 2 Example A. ANODI scores (inconsistency/diversity)

	×1	×1/2	×1/4	×1/8
$G \circ I$	*0.0353*/0.0314	*0.0822*/0.0961	*0.3347*/0.3974	0.6857/0.6466
`snesim`	0.0374/*0.0359*	0.0868/*0.1104*	0.3474/*0.4539*	*0.6703*/*0.6827*
$G \circ I$	0.0309/*0.0325*	0.0773/*0.0952*	0.3184/*0.3707*	0.6654/*0.6500*
`snesim`	*0.0246*/0.0268	*0.0578*/0.0721	*0.2804*/0.3363	*0.5995*/0.5981
$G \circ I$	*0.0263*/0.0337	*0.0619*/0.0857	*0.2467*/0.3097	0.6446/0.6332
`snesim`	0.0278/*0.0369*	0.0670/*0.1035*	0.2815/*0.3809*	*0.6394*/*0.6758*
$G \circ I$	*0.0353*/*0.0297*	*0.0834*/0.0861	*0.3573*/0.3721	*0.6694*/0.6433
`snesim`	0.0381/0.0286	0.0948/*0.0921*	0.3693/*0.3972*	0.6703/*0.6632*
$G \circ I$	*0.0252*/*0.0286*	*0.0583*/0.0738	0.2553/0.2866	*0.6316*/0.6105
`snesim`	0.0264/0.0275	0.0599/*0.0743*	*0.2496*/*0.3084*	0.6510/*0.6546*
$G \circ I$	0.0306/*0.0365*	0.0785/*0.1089*	0.3464/*0.4237*	*0.6612*/0.6410
`snesim`	*0.0283*/0.0327	*0.0686*/0.0928	*0.2844*/0.3632	0.6733/*0.6787*
$G \circ I$	0.0232/0.0170	0.0487/0.0345	*0.1929*/0.1464	*0.5003*/0.1698
`snesim`	*0.0207*/*0.0211*	*0.0446*/*0.0508*	0.1967/*0.2009*	0.5273/*0.4325*
$G \circ I$	0.0365/*0.0289*	0.0716/*0.0749*	0.3092/0.3129	*0.6570*/0.5101
`snesim`	*0.0331*/0.0243	*0.0710*/0.0732	*0.2943*/*0.3599*	0.6656/*0.6201*
$G \circ I$	*0.0299*/0.0310	*0.0817*/0.0830	*0.3051*/0.2339	*0.6396*/0.3698
`snesim`	0.0364/*0.0351*	0.0964/*0.1288*	0.3512/*0.4308*	0.6518/*0.6645*

We highlight in italics the best scores between G and `snesim`

(GAN) [15] in geology following successful results from recent works [16–21]. In [19], conditioning is addressed using a Bayesian formulation and performing Markov chain Monte Carlo to sample the corresponding posterior distribution. In [20, 21], the authors address conditioning using the inpainting technique from [57], which is equivalent to a Bayesian formulation using a "neural network prior" (see Appendix D for more details), and the sampling is done using local optimization. Our approach is closer to [19] in that we use a simple prior and aim to sample the full posterior, with the difference that the sampling is carried out by a neural network and we obtain a parametrization for the sampling process. Our approach is motivated by [58] where the authors trained a neural network to perform texture synthesis. Such authors used the sample entropy estimator for case $k = 1$ (nearest neighbor, see Eq. 9). The entropy estimator used in our work is a generalization introduced in [47]. A similar estimator based on random distances is used in [59] in the context of texture synthesis. In the context of generative modeling, [60] used a closed-form expression of the entropy term when using batch normalization [61]. Other alternatives to train neural samplers include normalizing flow [62], autoregressive flow [63], and Stein discrepancy [64]. These are all alternatives worth exploring in future work. Also related to our work include [65, 66] where the authors optimize the latent space to condition on labels/classes.

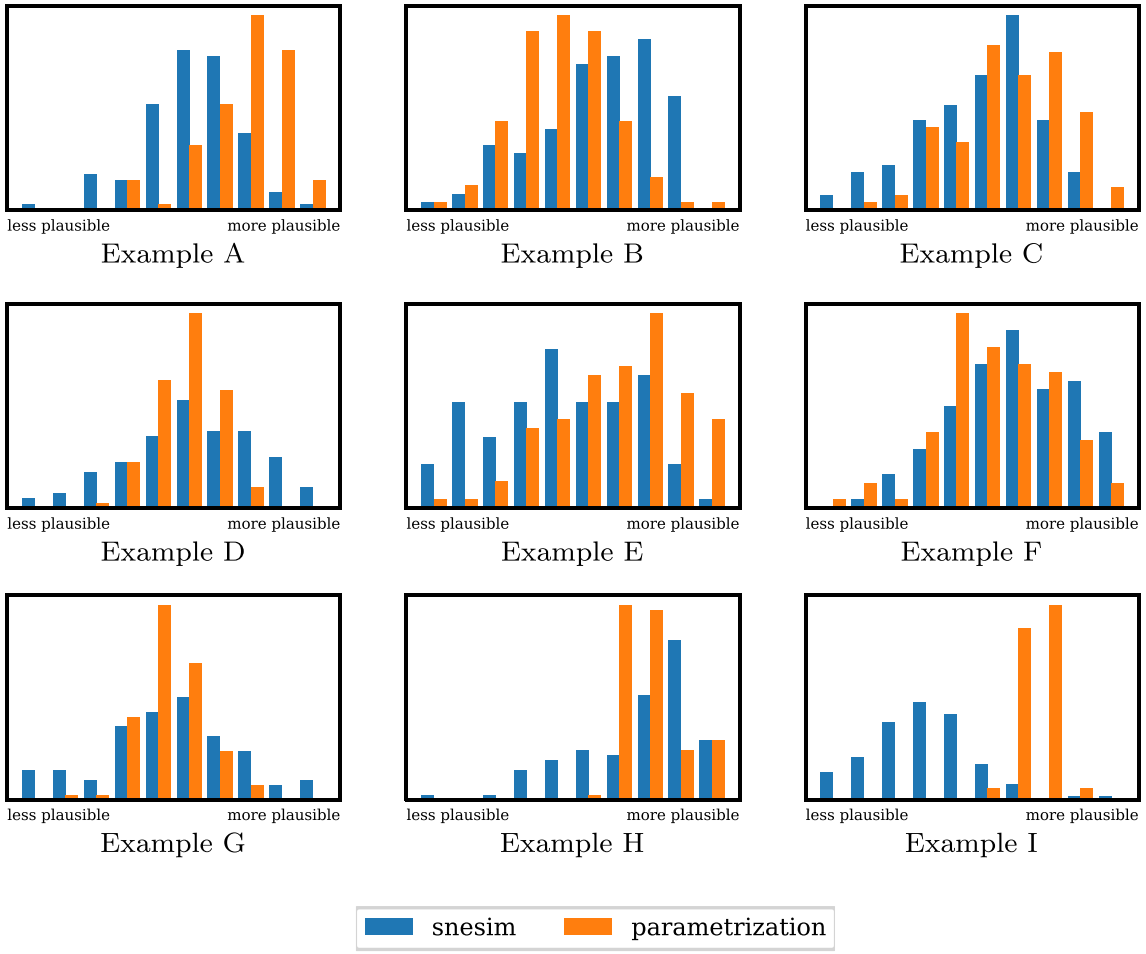


Fig. 16 Histograms of discriminator scores

6 Conclusion

We introduced a method to obtain a conditional parametrization by extending an existing unconditional parametrization, enabling reusability as well as direct and parametric sampling of conditional realizations. The parametrization considered in this work was based on deep neural networks motivated by their ability to express complex high-dimensional data such as natural images, including geological subsurface images. The unconditional parametrization G was obtained using generative adversarial networks (GAN) [15], and the post hoc conditioning was done by training a second neural network I to sample a Bayesian posterior, resulting in $G \circ I$ as the conditional parametrization.

We applied the method to parametrize binary channelized images using the benchmark image of [30]. In previous works, unconditional parametrization based on GAN was assessed using mostly two-point statistics tools. Here we added to the assessment using the analysis of distances method [53] which captures multipoint statistics. We found very positive results for the unconditional case, supporting previous results showing that G can effectively replicate the data generating process (in our case, the `snesim` [30] algorithm) while achieving dimensionality reduction of two orders of magnitude. Post-hoc conditional parametrization was explored for a variety of configurations. We found that $G \circ I$ produces very plausible realizations with good conditioning results, but the effectiveness may depend on the conditioning. Specifically, if the observations are in far disagreement with the reference

spatial statistics, effective conditioning may be difficult. For observations that agree with the reference spatial statistics, we found that the parametrization produces comparable results.

Possible future works include studying alternative training methods for the inference network as mentioned in Section 5, and further assessments with other images and in large scale settings.

Appendix A: Implementation details

This section describes training and hyperparameters of the neural network models. See [67] for a practical guide on training neural networks.

A.1 Generator neural network

The generator $G\colon \mathbb{R}^{30} \to \mathbb{R}^{64\times 64}$ is a deep convolutional neural network based on the template provided in [34]. The generator architecture consists of stacks of (transposed) convolutional layers (see Appendix E) together with batch normalization layers [61]. Batch normalization is the operation of normalizing the intermediate layer results to have zero mean and unit variance, which drastically improves optimization of deep neural networks [61]. For the non-linearity, we use rectified linear units (ReLU, $\sigma(x) = \max(0, x)$) in the intermediate layers, and $\sigma(x) = \tanh(x)$ in the last layer to constrain the output in $[-1, 1]$. The architecture is summarized in Table 3a. We train G using the Wasserstein formulation of GAN introduced in [41] with the proposed default hyperparameters. The optimization is performed using the Adam [48, 68] method with learning rate of 10^{-4} and batch size of 32. Our generator converges in approximately 20,000 iterations, taking around 30 minutes using a Nvidia GeForce GTX Titan X GPU. For deployment, it can generate approximately 5500 realizations per second using the GPU.

A.2 Inference neural network

We use the same inference network architecture $I\colon \mathbb{R}^{30} \to \mathbb{R}^{30}$ for all our conditioning experiments. The architecture is simply a stack of fully connected layers with constant-size intermediate layers. More specifically, we first transform the input from size 30 to size 512, then apply several more intermediate transformations preserving the size, and finally apply a transformation to bring the size back from 512 to 30 in the output layer. For the non-linearity, we use scaled exponential linear units (SeLU) [56], which are the current default option for deep fully connected networks: $\sigma(x) = \lambda x$ if $x > 0$, otherwise $\sigma(x) = \lambda\alpha(e^x - 1)$, where constants λ, α are given in [56]. No non-linearity is applied in the output layer (we do not need to bound the output as in the case of the generator). We experimented with different numbers of layers. Perhaps not surprisingly, we found that deeper architectures tended to produce better results in general. In our work, we settled with 5 intermediate layers. The architecture is summarized in Table 3b. We optimize I using the Adam method with learning rate of 10^{-4} and batch size of 64 for all the test cases. The network converges in between 1000 and 10,000 iterations, depending on the conditioning, taking between seconds and a few minutes to train using a Nvidia GeForce GTX Titan X GPU. For deployment, the conditional generator $G \circ I$ can generate approximately 5500 realizations per second using the GPU—we do not see significant increase in generation time from G to $G \circ I$.

Table 3 Neural network parametrization. `ConvT`, transposed convolution, the triplet indicates (filter size, stride, padding); `BN`, batch normalization; `FC`, fully connected

State size	Layer
(a) Generator architecture	
$30 \times 1 \times 1$	`ConvT(4,1,0), BN, ReLU`
$512 \times 4 \times 4$	`ConvT(4,2,1), BN, ReLU`
$256 \times 8 \times 8$	`ConvT(4,2,1), BN, ReLU`
$128 \times 16 \times 16$	`ConvT(4,2,1), BN, ReLU`
$64 \times 32 \times 32$	`ConvT(4,2,1), Tanh`
$1 \times 64 \times 64$	–
(b) Inference network architecture	
30	`FC, SeLU`
512	`FC, SeLU`
⋮	⋮
512	`FC, SeLU`
512	`FC`
30	–

Appendix B: Conditioning settings

The conditioning settings are summarized in Table 4.

Table 4 Conditioning configuration for each test case. The pair (i, j) denotes cell indices (row and column, respectively), and val $= 1$ indicates channel material, while val $= 0$ indicates background material

A			B			C			D			E			F		
i	j	val	i	j	val	i	j	val	i	j	val	i	j	val	i	j	val
12	12	0	12	12	1	12	12	1	0	50	1	0	20	1	33	44	1
12	25	0	25	12	0	25	12	0	10	50	1	5	22	1	28	42	1
12	38	1	38	12	0	38	12	0	20	50	1	10	23	1	24	40	1
12	51	1	51	12	1	51	12	0	30	50	1	15	25	1	18	35	1
25	12	1	12	25	0	12	25	0	40	50	1	20	26	1	16	30	1
25	25	0	25	25	1	25	25	1	50	50	1	30	30	1	20	23	1
25	38	0	38	25	1	38	25	0	60	50	1	40	33	1	27	20	1
25	51	0	51	25	0	51	25	0	0	15	1	45	34	1	32	19	1
38	12	0	12	38	0	12	38	0	10	21	1	50	36	1	39	21	1
38	25	1	25	38	1	25	38	0	20	26	1	55	37	1	45	24	1
38	38	1	38	38	1	38	38	1	30	32	1	60	39	1	48	32	1
38	51	1	51	38	0	51	38	0	40	37	1	60	20	1	43	37	1
51	12	0	12	51	1	12	51	0	50	43	1	55	22	1	36	40	1
51	25	0	25	51	0	25	51	0	60	48	1	50	23	1			
51	38	0	38	51	0	38	51	0	10	15	1	45	25	1			
51	51	1	51	51	1	51	51	1	20	15	1	40	26	1			
									30	15	1	35	30	1			
									40	15	1	20	33	1			
									50	15	1	15	34	1			
									60	15	1	10	36	1			
												5	37	1			
												0	39	1			

(a) A-F

	$j = 8$	$j = 16$	$j = 24$	$j = 32$	$j = 40$	$j = 48$	$j = 56$
$i = 8$	0	0	0	0	0	0	0
$i = 16$	1	1	1	1	1	1	1
$i = 24$	0	0	0	0	0	0	1
$i = 32$	0	0	0	0	0	0	0
$i = 40$	0	0	0	0	0	0	0
$i = 48$	1	0	0	0	0	0	1
$i = 56$	1	1	0	0	1	1	0

(b) G

	$j = 8$	$j = 16$	$j = 24$	$j = 32$	$j = 40$	$j = 48$	$j = 56$
$i = 8$	0	0	0	1	1	1	1
$i = 16$	0	0	0	0	1	0	1
$i = 24$	0	1	1	1	0	0	0
$i = 32$	1	0	0	0	0	0	0
$i = 40$	0	0	0	1	1	0	0
$i = 48$	1	1	1	0	0	1	0
$i = 56$	0	0	0	1	1	0	1

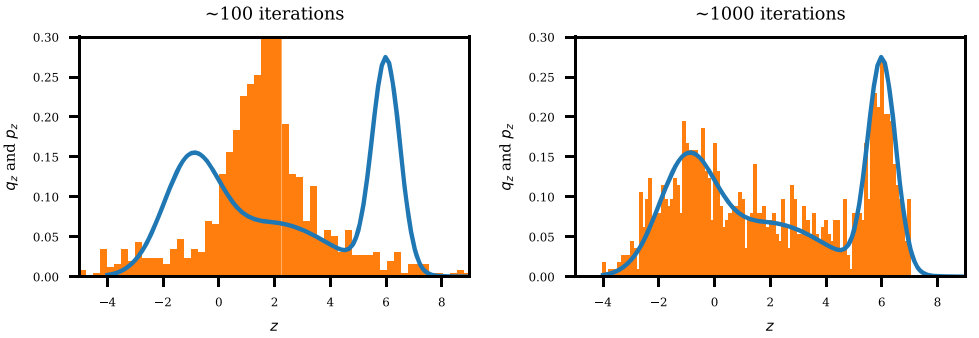
(c) H

	$j = 8$	$j = 16$	$j = 24$	$j = 32$	$j = 40$	$j = 48$	$j = 56$
$i = 8$	0	1	0	0	0	1	1
$i = 16$	0	1	0	0	0	0	1
$i = 24$	0	1	0	0	0	0	0
$i = 32$	0	1	0	0	0	0	0
$i = 40$	0	1	0	0	0	0	1
$i = 48$	0	1	0	0	0	0	1
$i = 56$	0	1	1	0	0	1	0

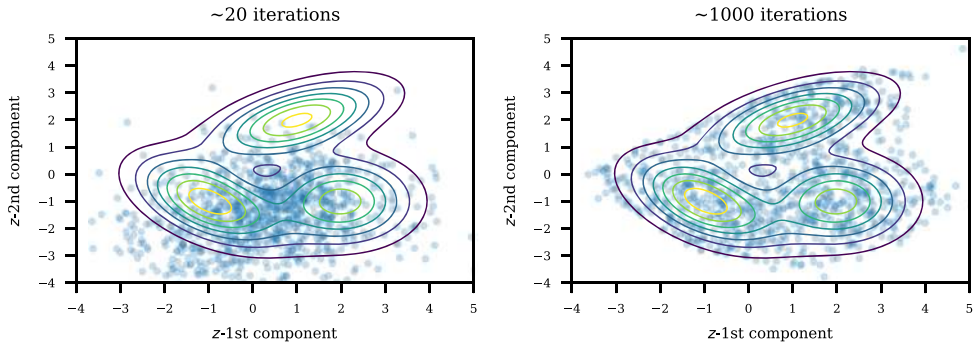
(d) I

Appendix C: Mixture of Gaussians

The proposed method described in Section 3 can be used to train a general *neural sampler*. In this side section, we perform a simple sanity check by assessing the method on a toy problem where we train neural networks to sample mixture of Gaussians. Concretely, we train fully connected neural networks $I_\phi \colon \mathbb{R}^{n_w} \to \mathbb{R}^{n_z}$ to sample simple 1D and 2D mixture of Gaussians, with $n_z = n_w = 1$ in the 1D case, and $n_z = n_w = 2$ in the 2D case. The source distribution p_w



(a) Mixture of three 1D Gaussians. The blue line indicates the target distribution, and the normalized histogram corresponds to generated values.



(b) Mixture of three 2D Gaussians. The contour lines indicate the target distribution, and the scattered points correspond to generated values.

Fig. 17 Results of I_ϕ trained to generate mixture of Gaussians. **a** Mixture of three 1D Gaussians. The blue line indicates the target distribution, and the normalized histogram corresponds to generated values. **b** Mixture of three 2D Gaussians. The contour lines indicate the target distribution, and the scattered points correspond to generated values

is the standard normal in both cases. Results are summarized in Fig. 17.

The first example (Fig. 17a) is a mixture of three 1D Gaussians, with centers $\mu_1 = -1$, $\mu_2 = 2$, and $\mu_3 = 6$, and standard deviations $\sigma_1 = 1$, $\sigma_2 = 2$, and $\sigma_3 = 0.5$, respectively. The density of the Gaussian mixture is indicated along with a histogram for 1000 points generated by the neural network at an early stage of the training (100 iterations), and at convergence (1000 iterations). The second example (Fig. 17b) is a mixture of three 2D Gaussians, with centers $\mu_1 = (-1, -1)$, $\mu_2 = (1, 2)$ and $\mu_3 = (2, -1)$, and covariances $\Sigma_1 = \begin{pmatrix} 1 & -0.5 \\ -0.5 & 1 \end{pmatrix}$, $\Sigma_2 = \begin{pmatrix} 1.5 & 0.6 \\ 0.6 & 0.8 \end{pmatrix}$, and $\Sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, respectively. We plot the contour lines of the density of the Gaussian mixture. We also show a scatterplot of 4000 points generated by the neural network at an early stage of the training (20 iterations), and at convergence (1000 iterations). In both test cases, we can verify that the neural network effectively learns to transport points from the standard normal distribution to the mixture of Gaussians.

Appendix D: Comparison with related work based on inpainting

In image processing, image inpainting is used to fill incomplete images or replace a subregion of an image (e.g., a face with eyes covered). The recent GAN-based inpainting technique employed in [20, 21] uses an optimization approach with the following loss:

$$\mathcal{L}(z) = \|G(z)_{\text{obs}} - d_{\text{obs}}\|^2 + \lambda \log(1 - D(G(z))) \tag{10}$$

The second term in this loss function is referred to as the *perceptual loss* and is the same second term in the GAN loss in Eq. 2, which is the classification score on synthetic realizations. Compare Eq. 10 with Eq. 6: While our Bayesian posterior uses a simple Gaussian prior, the prior in Eq. 10 (the perceptual loss) involves the discriminator D used during the GAN training. We argue that the Gaussian prior can be equally effective, as long as the GAN training has converged successfully: If G and D are at convergence, then $G(\mathbf{z})$ always produces plausible realizations for $\mathbf{z} \sim p_z$ where p_z is the chosen latent distribution, and D is $1/2$ for all realizations of $G(\mathbf{z})$. In such scenario, the perceptual

loss should then act as a regularization term that drives z towards regions of high density of the latent distribution p_z, therefore having a similar effect to using p_z as the prior.

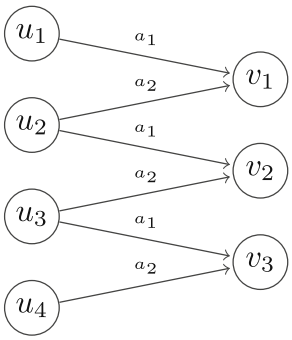
For example, let us consider $\mathbf{z} \sim \mathcal{U}[0, 1]$ and $\mathbf{y} \sim \mathcal{U}[1, 3]$. An optimal generator would be $G(z) = 2z + 1$ and an optimal discriminator $D(y) = 1/2$ for $y \in [1, 3]$ and $D(y) = 0$ otherwise. Then, $D(G(z)) = 1/2$ for $z \in [0, 1]$, and $D(G(z)) = 0$ otherwise, which is precisely the density function of $\mathbf{z} \sim \mathcal{U}[0, 1]$ scaled by $1/2$. Therefore, in this example the perceptual loss and p_z as prior would have the same effect. Nevertheless, in practice the perceptual loss can be very useful when G and D are not exactly optimal and there exist bad realizations from G. In that case, the perceptual loss can help the optimization to find good solutions. In our work, we found our Gaussian prior to be sufficient while removing a layer of complexity in the optimization.

Appendix E: Convolutional neural networks

We provide a very brief description by example of convolutional neural networks (see [69, 70] for further details or [71] for a more practical treatment). Let $u = (u_1, u_2, u_3, u_4)$ and $a = (a_1, a_2)$. Let us call a a *filter*. To convolve the filter a on u is to compute the output vector v with components $v_i = u_i a_1 + u_{i+1} a_2$ for $i = 1, \cdots, 3$. The operation is illustrated as a neural network layer in Fig. 18a. In this example, the convolution has a stride of 1 (at which the filter is swept), but in general it can be any positive integer.

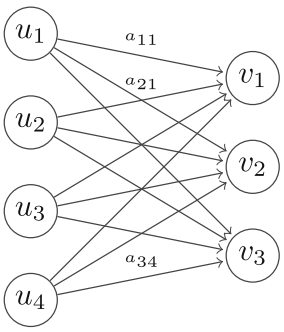
We also show the matrix A associated with this operation—it is easy to verify that $v = Au$. We see that the associated matrix is sparse and diagonal-constant, which is the appeal of using convolutional layers. This structural constraint achieves two things: it drastically reduces the number of free weights, and it does so by assuming a locality prior. This locality prior turns out to be useful in practice, since nearby events in natural phenomena (natural images, speech, text, etc.) tend to be correlated.

Compare the convolutional layer with the fully connected layer shown in Fig. 18b: In the fully connected case, the associated matrix is dense, resulting in 12 free weights whereas the convolution layer has only 2 for the same layer sizes. This difference is greatly amplified in practice where inputs/outputs are large (e.g., images), making convolutional layers a much more efficient architecture. Note that in practice we use *deep* architectures, i.e., several stacks of convolutional layers, therefore the full connectivity can be recovered if necessary, although now



$$A = \begin{pmatrix} a_1 & a_2 & 0 & 0 \\ 0 & a_1 & a_2 & 0 \\ 0 & 0 & a_1 & a_2 \end{pmatrix}$$

(a) A convolutional layer.



$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix}$$

(b) A fully connected layer.

Fig. 18 Neural network layers and respective transformation matrices

with an embedded locality prior along with a hierarchy in the influence of the weights.

Note that the example considered above would always result in a smaller output vector size. If the opposite effect is desired, a simple solution is to transpose the matrix A. For this reason, this operation is called a transposed convolution. Several stacks of transposed convolutions are typically used in generators and decoders to upsample the small latent vector to the full-size output image. In classifier neural networks, normal convolutions are used instead to downsample the large image to a single number indicating a probability.

Our brief description can be readily extended to 2D and 3D arrays with corresponding multidimensional filters. For example, for a 2D input the filters are of rectangular shape and can be swept horizontally and vertically. See [71] for further practical details.

Appendix F: Computational complexity

Let N denote the dataset size and d the dimension of each realization. Fast PCA methods based on singular value decomposition can achieve a complexity of $\mathcal{O}(N^2 d)$. This complexity is favorable in geology where $N \ll d$, i.e., we have a small number of very large realizations (although it still grows quadratically with the number of realizations). For our present method, reporting the computational complexity is less straightforward since it is highly problem-dependent. To illustrate the difficulties, we discuss in the following the computational complexity of a classifier neural network—similar arguments apply to encoders, generators and decoders.

Computing the computational complexity of neural network models is cumbersome since it fully depends on the architecture, which in turn depends on the *learning difficulty* of the problem at hand. For example, in the simple case that the dataset is linearly separable, a classifier neural network of the form $f(x) = \sigma(w^T x + b)$, with w, b to be determined, is enough to correctly classify all points of the dataset. The evaluation cost of this neural network is simply $\mathcal{O}(d)$, hence the training cost is $\mathcal{O}(Td)$ (when using stochastic gradient descent as normally done), where T is the number of update iterations. Note that this expression does not depend on N, although in practice T is at most linear in N, e.g., when performing multiple passes through the dataset until convergence, but note that the training can also converge even before a single pass through the dataset (which happens on massive datasets). Hence, neural networks are very favorable in the big data setting, i.e., when N is very large.

The estimated evaluation cost of $\mathcal{O}(d)$ is overly optimistic since in practice we use *deep* architectures to deal with complex datasets that are not linearly separable. If the architecture is instead $f(x) = \sigma(A_l(\cdots\sigma(A_2(\sigma(A_1 x + b_1) + b_2))\cdots) + b_l)$, where each A_i is a $d \times d$ matrix, then the evaluation cost of this architecture is roughly $\mathcal{O}(d^2)$ (we omit the number of layers l since this is a constant factor and $l \ll d$). However, this estimate is now overly pessimistic: First, in practice the A_i are not shape-preserving, instead they decrease very quickly in size while exponentially compressing the input (e.g., A_1 is of size $d \times \frac{d}{2}$, A_2 is of size $\frac{d}{2} \times \frac{d}{4}$). Second, the matrices A_i are rarely full since convolutional layers are used instead (see Appendix E), resulting in very sparse matrices that are several orders of magnitude lighter. Modern architectures use several stacks of exponentially decreasing convolutional layers, while fully connected layers are avoided or used only sparingly (and for small inputs/outputs). The overall effect is a drastic reduction in the computational complexity, from $\mathcal{O}(d^2)$ to $\mathcal{O}(kd)$ where k is a factor that is determined by the architecture. The corresponding training complexity is then $\mathcal{O}(Tkd)$. Note that although $k < d$ in practice, k can still be sizable. On the other hand, $k = 1$ is also possible as just mentioned. Ultimately, k will depend on the learning difficulty of the problem. In most models encountered in the literature, k grows sublinearly with d.

Perhaps more importantly is the human time, rather than computational time, that is involved in optimizing the dozens of hyperparameters—in particular the architecture design—for which automation is currently limited. As mentioned before, designing the architecture is heavily based on experience, heuristics, and experimentation which incur high costs in terms of engineering time. The justification of such costs will ultimately depend on the lifespan of the model, since the model needs to be constructed only once but can be deployed for a long time (e.g., history matching) or virtually indefinitely (e.g., most applications in internet companies such as recommender systems, visual and voice recognition, language translation). Automatic architecture search is an ongoing area of research (see e.g. [72, 73] and references therein).

References

1. Jacquard, P.: Permeability distribution from field pressure data. Soc. Pet. Eng. https://doi.org/10.2118/1307-PA (1965)
2. Jahns, H.O.: A rapid method for obtaining a two-dimensional reservoir description from well pressure response data. Soc. Pet. Eng. https://doi.org/10.2118/1473-PA (1966)
3. Sarma, P., Durlofsky, L.J., Aziz, K.: Kernel principal component analysis for efficient, differentiable parameterization of multipoint geostatistics. Math. Geosci. **40**(1), 3–32 (2008)
4. Ma, X., Zabaras, N.: Kernel principal component analysis for stochastic input model generation. J. Comput. Phys. **230**(19), 7311–7331 (2011)

5. Vo, H.X., Durlofsky, L.J.: Regularized kernel PCA for the efficient parameterization of complex geological models. J. Comput. Phys. **322**, 859–881 (2016)
6. Shirangi, M.G., Emerick, A.A.: An improved TSVD-based Levenberg–Marquardt algorithm for history matching and comparison with Gauss–Newton. J. Pet. Sci. Eng. **143**, 258–271 (2016)
7. Tavakoli, R., Reynolds, A.C.: Monte Carlo simulation of permeability fields and reservoir performance predictions with SVD parameterization in RML compared with EnKF. Comput. Geosci. **15**(1), 99–116 (2011)
8. Jafarpour, B., McLaughlin, D.B.: Reservoir characterization with the discrete cosine transform. Soc. Petrol. Eng. https://doi.org/10.2118/106453-PA (2009)
9. Jafarpour, B., Goyal, V.K., McLaughlin, D.B., Freeman, W.T.: Compressed history matching: exploiting transform-domain sparsity for regularization of nonlinear dynamic data integration problems. Math. Geosci. **42**(1), 1–27 (2010). ISSN 1874-8953. https://doi.org/10.1007/s11004-009-9247-z
10. Moreno, D., Aanonsen, S.I.: Stochastic facies modelling using the level set method. In: EAGE Conference on Petroleum Geostatistics (2007)
11. Dorn, O., Villegas, R.: History matching of petroleum reservoirs using a level set technique. Inverse Prob. **24**(3), 035015 (2008). http://stacks.iop.org/0266-5611/24/i=3/a=035015
12. Chang, H., Zhang, D., Lu, Z.: History matching of facies distribution with the EnKF and level set parameterization. J. Comput. Phys. **229**(20), 8011–8030 (2010). ISSN 0021-9991. https://doi.org/10.1016/j.jcp.2010.07.005. http://www.sciencedirect.com/science/article/pii/S0021999110003748
13. Khaninezhad, M.M., Jafarpour, B., Li, L.: Sparse geologic dictionaries for subsurface flow model calibration: part i. Inversion formulation. Adv. Water Resour. **39**, 106–121 (2012)
14. Khaninezhad, M.M., Jafarpour, B., Li, L.: Sparse geologic dictionaries for subsurface flow model calibration: part ii. Robustness to uncertainty. Adv. Water Resour. **39**, 122–136 (2012)
15. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Bing, X.u., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in Neural Information Processing Systems, pp. 2672–2680 (2014)
16. Mosser, L., Dubrule, O., Blunt, M.J.: Reconstruction of three-dimensional porous media using generative adversarial neural networks. arXiv:1704.03225 (2017)
17. Mosser, L., Dubrule, O., Blunt, M.J.: Stochastic reconstruction of an oolitic limestone by generative adversarial networks. arXiv:1712.02854 (2017)
18. Chan, S., Elsheikh, A.H.: Parametrization and generation of geological models with generative adversarial networks. arXiv:1708.01810 (2017)
19. Laloy, E., Hérault, R., Jacques, D., Linde, N.: Training-image based geostatistical inversion using a spatial generative adversarial neural network. Water Resour. Res. **54**(1), 381–406 (2018)
20. Dupont, E., Zhang, T., Tilke, P., Liang, L., Bailey, W.: Generating realistic geology conditioned on physical measurements with generative adversarial networks. arXiv:1802.03065 (2018)
21. Mosser, L., Dubrule, O., Blunt, M.J.: Conditioning of three-dimensional generative adversarial networks for pore and reservoir-scale models. arXiv:1802.05622 (2018)
22. Chan, S., Elsheikh, A.H.: Parametrization of stochastic inputs using generative adversarial networks with application in geology. arXiv:1904.03677 (2019)
23. Marçais, J., de Dreuzy, J.-R.: Prospective interest of deep learning for hydrological inference. Groundwater **55**(5), 688–692 (2017)
24. Nagoor Kani, J., Elsheikh, A.H.: DR-RNN: a deep residual recurrent neural network for model reduction. arXiv:1709.00939 (2017)
25. Klie, H., et al.: Physics-based and data-driven surrogates for production forecasting. In: SPE Reservoir Simulation Symposium. Society of Petroleum Engineers (2015)
26. Stanev, V.G., Iliev, F.L., Hansen, S., Vesselinov, V.V., Alexandrov, B.S.: Identification of release sources in advection–diffusion system by machine learning combined with Green's function inverse method. Appl. Math. Model. **60**, 64–76 (2018)
27. Sun, W., Durlofsky, L.J.: A new data-space inversion procedure for efficient uncertainty quantification in subsurface flow problems. Math. Geosci. **49**(6), 679–715 (2017)
28. Zhu, Y., Zabaras, N.: Bayesian deep convolutional encoder-decoder networks for surrogate modeling and uncertainty quantification. J. Comput. Phys. **366**, 415-447 (2018)
29. Valera, M., Guo, Z., Kelly, P., Matz, S., Cantu, A., Percus, A.G., Hyman, J.D., Srinivasan, G., Viswanathan, H.S.: Machine learning for graph-based representations of three-dimensional discrete fracture networks. arXiv:1705.09866 (2017)
30. Strebelle, S.B., Journel, A.G.: Reservoir modeling using multiple-point statistics. In: SPE Annual Technical Conference and Exhibition. Society of Petroleum Engineers (2001)
31. Brock, A., Donahue, J., Simonyan, K.: Large scale gan training for high fidelity natural image synthesis. arXiv:1809.11096 (2018)
32. Karras, T., Aila, T., Laine, S., Lehtinen, J.: Progressive growing of gans for improved quality, stability, and variation. arXiv:1710.10196 (2017)
33. Schmidhuber, J.: Learning factorial codes by predictability minimization. Neural Comput. **4**(6), 863–879 (1992)
34. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv:1511.06434 (2015)
35. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved techniques for training gans. In: Advances in Neural Information Processing Systems, pp. 2234–2242 (2016)
36. Arjovsky, M., Bottou, L.: Towards principled methods for training generative adversarial networks. arXiv:1701.04862 (2017)
37. Arora, S., Ge, R., Liang, Y., Ma, T., Zhang, Y.: Generalization and equilibrium in generative adversarial nets (GANs). arXiv:1703.00573 (2017)
38. Müller, A.: Integral probability metrics and their generating classes of functions. Adv. Appl. Probab. **29**(2), 429–443 (1997)
39. Gretton, A., Borgwardt, K.M., Rasch, M., Schölkopf, B., Smola, A.J.: A kernel method for the two-sample-problem. In: Advances in Neural Information Processing Systems, pp. 513–520 (2007)
40. Dziugaite, G.K., Roy, D.M., Ghahramani, Z.: Training generative neural networks via maximum mean discrepancy optimization. arXiv:1505.03906 (2015)
41. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein GAN. arXiv:1701.07875 (2017)
42. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.C.: Improved training of Wasserstein GANs. In: Advances in Neural Information Processing Systems, pp. 5769–5779 (2017)
43. Mroueh, Y., Sercu, T.: Fisher GAN. In: Advances in Neural Information Processing Systems, pp. 2510–2520 (2017)
44. Mroueh, Y., Li, C.-L., Sercu, T., Raj, A., Cheng, Y.: Sobolev GAN. arXiv:1711.04894 (2017)
45. Mroueh, Y., Sercu, T., Goel, V.: Mcgan: mean and covariance feature matching GAN. arXiv:1702.08398 (2017)
46. Kozachenko, L.F., Leonenko, N.N.: Sample estimate of the entropy of a random vector. Problemy Peredachi Informatsii **23**(2), 9–16 (1987)
47. Goria, M.N., Leonenko, N.N., Mergel, V.V., Inverardi, P.LN.: A new class of random vector entropy estimators and its applications

in testing statistical hypotheses. J. Nonparametr. Stat. **17**(3), 277–297 (2005)
48. Kingma, D., Ba, J.: Adam: a method for stochastic optimization. arXiv:1412.6980 (2014)
49. Tieleman, T., Hinton, G.: Lecture 6.5-RMSprop: divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning 4(2). https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (2012)
50. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch. NIPS Autodiff Workshop (2017)
51. Strebelle, S.: Conditional simulation of complex geological structures using multiple-point statistics. Math. Geol. **34**(1), 1–21 (2002)
52. Remy, N., Boucher, A., Wu, J.: Sgems: Stanford geostatistical modeling software. Software Manual (2004)
53. Tan, X., Tahmasebi, P., Caers, J.: Comparing training-image based algorithms using an analysis of distance. Math. Geosci. **46**(2), 149–169 (2014)
54. Borg, I., Groenen, P.: Modern multidimensional scaling: theory and applications. J. Educ. Meas. **40**(3), 277–280 (2003)
55. Otsu, N.: A threshold selection method from gray-level histograms. IEEE Trans. Syst. Man Cybern. **9**(1), 62–66 (1979)
56. Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S.: Self-normalizing neural networks. In: Advances in Neural Information Processing Systems, pp. 971–980 (2017)
57. Yeh, R., Chen, C., Lim, T.Y., Hasegawa-Johnson, M., Do, M.N.: Semantic image inpainting with perceptual and contextual losses. arXiv:1607.07539 (2016)
58. Ulyanov, D., Vedaldi, A., Lempitsky, V.: Improved texture networks: maximizing quality and diversity in feed-forward stylization and texture synthesis. In: Proceedings of CVPR (2017)
59. Li, Y., Fang, C., Yang, J., Wang, Z., Lu, X., Yang, M.-H.: Diversified texture synthesis with feed-forward networks. In: Proceedings of CVPR (2017)
60. Kim, T., Bengio, Y.: Deep directed generative models with energy-based probability estimation. arXiv:1606.03439 (2016)
61. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167 (2015)
62. Rezende, D.J., Mohamed, S.: Variational inference with normalizing flows. arXiv:1505.05770 (2015)
63. Kingma, D.P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., Welling, M.: Improved variational inference with inverse autoregressive flow. In: Advances in Neural Information Processing Systems, pp. 4743–4751 (2016)
64. Wang, D., Liu, Q.: Learning to draw samples: with application to amortized mle for generative adversarial learning. arXiv:1611.01722 (2016)
65. Nguyen, A., Yosinski, J., Bengio, Y., Dosovitskiy, A., Clune, J.: Plug & play generative networks: conditional iterative generation of images in latent space. arXiv:1612.00005 (2016)
66. Engel, J., Hoffman, M., Roberts, A.: Latent constraints: learning to generate conditionally from unconditional generative models. arXiv:1711.05772 (2017)
67. Bengio, Y.: Practical recommendations for gradient-based training of deep architectures. In: Neural Networks: Tricks of the Trade, pp. 437–478. Springer (2012)
68. Reddi, S.J., Kale, S., Kumar, S.: On the convergence of Adam and beyond. International Conference on Learning Representations (2018)
69. Fukushima, K., Miyake, S.: Neocognitron: a self-organizing neural network model for a mechanism of visual pattern recognition. In: Competition and Cooperation in Neural Nets, pp. 267–285. Springer (1982)
70. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural Comput. **1**(4), 541–551 (1989)
71. Dumoulin, V., Visin, F.: A guide to convolution arithmetic for deep learning. arXiv:1603.07285 (2016)
72. Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., De Freitas, N.: Taking the human out of the loop: a review of Bayesian optimization. Proc. IEEE **104**(1), 148–175 (2016)
73. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv:1611.01578 (2016)