npj | computational materials

www.nature.com/npjcompumats

ARTICLE OPEN

Check for updates

End-to-end differentiability and tensor processing unit computing to accelerate materials' inverse design

Han Liu[1,2]✉, Yuhan Liu[3,4], Kevin Li[3,4], Zhangji Zhao[3], Samuel S. Schoenholz[5], Ekin D. Cubuk[5], Puneet Gupta[6] and Mathieu Bauchy[3]✉

Numerical simulations have revolutionized material design. However, although simulations excel at mapping an input material to its output property, their direct application to inverse design has traditionally been limited by their high computing cost and lack of differentiability. Here, taking the example of the inverse design of a porous matrix featuring targeted sorption isotherm, we introduce a computational inverse design framework that addresses these challenges, by programming differentiable simulation on TensorFlow platform that leverages automated end-to-end differentiation. Thanks to its differentiability, the simulation is used to directly train a deep generative model, which outputs an optimal porous matrix based on an arbitrary input sorption isotherm curve. Importantly, this inverse design pipeline leverages the power of tensor processing units (TPU)—an emerging family of dedicated chips, which, although they are specialized in deep learning, are flexible enough for intensive scientific simulations. This approach holds promise to accelerate inverse materials design.

npj Computational Materials (2023)9:121 ; https://doi.org/10.1038/s41524-023-01080-x

INTRODUCTION

Numerical simulations have transformed the way we design materials[1]. For instance, density functional theory and molecular dynamics excel at predicting the properties of materials based on the knowledge of their composition and atomic structure[2,3]. This makes it possible to replace costly trial-and-error experiments by simulations so as to screen in silico promising materials[4]. However, numerical simulations are of limited help to tackle inverse design problems (i.e., identifying an optimal material featuring optimal properties within a given design space)[5–7]. Indeed, although numerical simulations are typically faster and cheaper than experiments, their computational burden usually prevents a thorough exploration of the design space (e.g., the systematic exploration of all possible materials' compositions)[8]. In addition, traditional numerical simulations are usually not differentiable, which prevents their seamless integration with gradient-based optimization methods[9,10]. These limitations—which are reminiscent of the state of machine learning before automatic differentiation became popular[11]—have limited the use of numerical simulations in inverse design pipelines[12].

To address this issue, it is common to replace simulations by a differentiable surrogate predictor machine learning model, which aims to approximately interpolate the mapping between design space parameters (e.g., the material's structure) and the target property of interest[7,12,13]. Following this approach, Generative Networks (GNs)[5] have been used for inverse design application using, for instance, autoencoders[14], generative adversarial networks[15], or generative inverse design networks[12]. The generator can then be combined with the differentiable surrogate predictor in the same pipeline so as to be trained by gradient backpropagation[12,16,17]. However, this approach can result in difficulties associated with the fact that the generator and predictor must both be trained, either simultaneously or sequentially. In addition, the ability of the generator to discover new unknown, potentially non-intuitive material designs (i.e., which are very different from those in the training set) is often limited by the accuracy and generalizability of the surrogate predictor[5–7]. Then the question is, can we avoid using surrogate predictor? With the recent expansion of automatic differentiation technologies[18,19], differentiable programming platforms—such as TensorFlow[20], JAX[21], and TaiChi[22]—are rapidly developing and getting attention for differentiable simulation applications[23,24], including molecular dynamics[10,11] and robotic dynamics[25]. However, as differentiable programming remains largely unexplored in material simulations, the potential of directly training a generator based on a differentiable predictor has received less attention.

Here, to address the challenges facing surrogate predictor, we introduce a deep generative pipeline that combines an end-to-end differentiable simulator with a generator model. Note that the present work is a fully developed version of our recent report in a conference proceeding[26]. We illustrate the power of this approach by taking the example of the inverse design of a porous matrix featuring targeted sorption isotherm—wherein the sorption isotherm corresponds here to the amount of adsorbed liquid water in the porous structure as a function of relative humidity. This is enabled by the implementation of an end-to-end differentiable lattice-based density functional theory code in TensorFlow[20,21]. We show that the trained generative model is able to successfully generate porous structures with arbitrary sorption curves. Moreover, this generator-simulator pipeline leverages the power of tensor processing units (TPU)—an emerging family of dedicated chips[27], which, although they are specialized in deep learning, are flexible enough for intensive scientific simulations. This approach holds promise to accelerate the inverse design of materials with tailored properties and functionalities.

[1]SOlids inFormaTics AI-Laboratory (SOFT-AI-Lab), College of Polymer Science and Engineering, Sichuan University, Chengdu 610065, China. [2]AIMSOLID Research, Wuhan 430223, China. [3]Physics of AmoRphous and Inorganic Solids Laboratory (PARISlab), Department of Civil and Environmental Engineering, University of California, Los Angeles, CA 90095, USA. [4]Department of Computer Science, University of California, Los Angeles, CA 90095, USA. [5]Google Research, Brain Team, Mountain View, CA, USA. [6]Department of Electrical and Computer Engineering, University of California, Los Angeles, CA 90095, USA. ✉email: happylife@ucla.edu; bauchy@ucla.edu

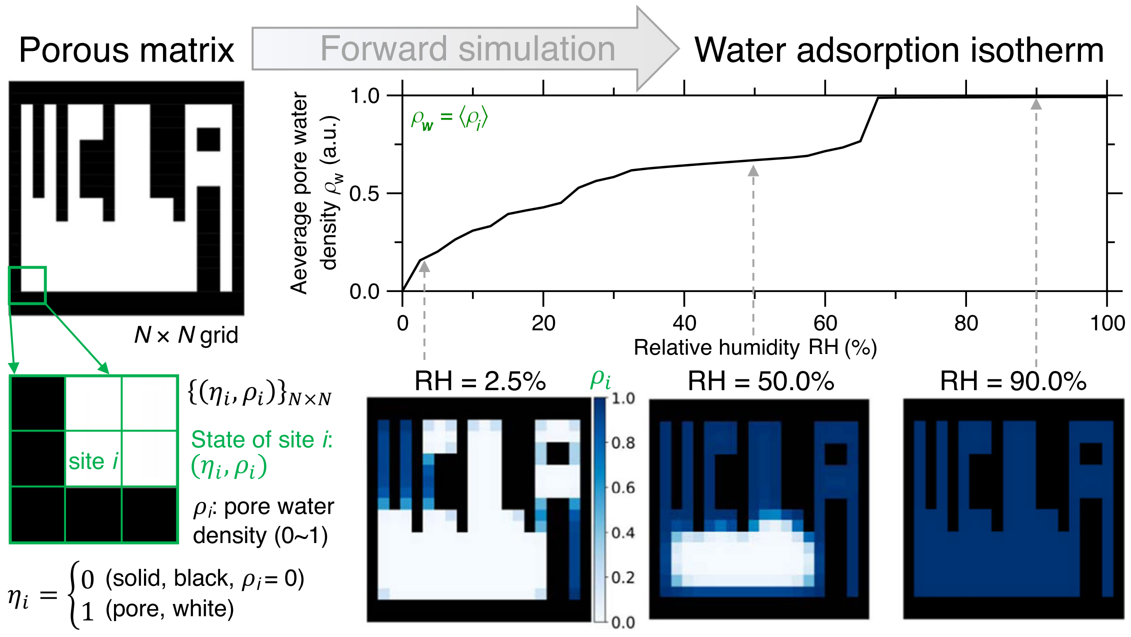

Fig. 1 Numerical simulation of water sorption in a porous matrix. The porous matrix is represented by an N-by-N grid, wherein each pixel i of the grid can be filled with solid ($\eta_i = 0$) or be a pore ($\eta_i = 1$). ρ_i is the density of water in the pore and is calculated by iteratively applying Eq. (1) until convergence at each relative humidity (RH), where RH increases from 0 to 100% to yield the sorption isotherm, and $\rho_i = 0$ and 1 denote that the pore is fully empty or saturated with water, respectively.

RESULTS

Numerical simulation of water sorption in porous matrices

To establish our conclusions, we take a close inspection on the conventional numerical simulation exemplified by water sorption in a target porous matrix. Figure 1 illustrates the water sorption simulation for a toy porous matrix represented by a square N-by-N lattice, where each pixel i of the grid can either be filled with solid ($\eta_i = 0$) or be a pore ($\eta_i = 1$), and the initially empty pores can be gradually filled with water upon increasing relative humidity (RH), with the pore water density $\rho_i = 0$ and 1 representing that the pore is fully empty or saturated with water, respectively. At each RH, the equilibrium density of water in each pore is computed by lattice density functional theory (LDFT)[28,29]. Details about the LDFT formalism are provided in the Methods section. Based on this formalism, the water density ρ_i at a given pixel i is given by Eq. (1) (see Methods section), where ρ_i depends on the state of its 4 neighbors, which is essentially a convolution operation. At each RH, the equilibrium fraction of water is determined by iteratively applying Eq. (1) on each pixel until a convergence in the $\{\rho_i\}$ values are obtained. The sorption isotherm of the porous matrix is then determined by computing the equilibrium values of $\{\rho_i\}$ for all RH values ranging from 0 to 100% with an increment dRH = 2.5%. At each increment step K, namely, at RH = $K \times$ dRH (herein, $K = \{0, 1, 2, 3, \ldots, 39\}$), the equilibrium water density values $\{\rho_i\}^{K\text{th}}$ serve as the starting configuration to calculate $\{\rho_i\}^{K+1}$ at the subsequent step $K + 1$. More details of the numerical simulations can be found in the Methods section.

End-to-end differentiable reformulation of the sorption simulation

Such (LDFT) simulations are traditionally not differentiable. Here, to address this limitation, we decompose Eq. (1) into a series of mathematical operations that can be implemented as differentiable computation layers in TensorFlow. Figure 2 shows the differentiable simulation architecture, where we decompose Eq. (1) into three layers, namely, (i) the input layer, (ii) the CONV layer, and (iii) the output layer. Note that the CONV layer represents the convolution operation in Eq. (1)—i.e., one of the operations that can be efficiently performed by TPUs. The input layer consists of three parallel layers associated with three input matrices, respectively, where one input matrix $\{\eta_i\}_{N\times N}$ is fed into the output layer, and the other two matrices $\{\rho_i\}_{N\times N}$ and $\{1 - \eta_i\}_{N\times N}$ are fed into two parallel CONV layers. Then the two CONV layers conduct the convolution operation $\sum_{j/i}[w_{\mathrm{ff}}\rho_j]$ and $\sum_{j/i}[w_{\mathrm{mf}}(1 - \eta_j)]$ (see Eq. (1) in Methods section), respectively, wherein j/i indicates 4 neighbors of pixel i. Finally, the two convolution outcomes (denoted as C1 and C2) together with the input matrix $\{\eta_i\}_{N\times N}$ is fed into the output layer that conducts the remaining mathematical operation of Eq. (1), namely, $\rho_i = \frac{\eta_i}{1+e^{-(\mu+C1+C2)/(kT)}}$. At each RH, we repeat this three-layer block (i.e., the decomposed layers of Eq. (1)) for M times in series, which is equivalent to iteratively solving Eq. (1) until a convergence in the water density is achieved. Importantly, since all layers share the feature of automatic differentiation in TensorFlow, the gradient of each layer can back propagate to enable end-to-end differentiation.

Accuracy of the differentiable simulator

We now evaluate the accuracy of the reformulated differentiable simulator. Figure 3a shows a comparison between the sorption curve of a porous matrix computed by the reference (undifferentiable) simulator and its reformulated differentiable counterpart, where the area between the two curves defines the percentage loss L of the differentiable simulation. We then evaluate the average percentage loss of the differentiable simulator using a large validation set of 8769 porous matrices. Figure 3b shows the validation set of grids featuring a diverse population of reference sorption curves (computed by the reference simulator), as characterized by a wide distribution of the reference curves' sinuosity index S_r, where S_r is calculated as the ratio of the curvilinear length along the curve over the straight-line length between end points of the curve. Figure 3c shows the percentage loss L for the validation set as a function of the number of convolution layers M. As expected, larger M yields more iterations of Eq. (1) to facilitate the convergence of water density, thus enhancing the simulation accuracy, and when $M = 100$, L reduces to a miniscule level so that the differentiable simulator is as

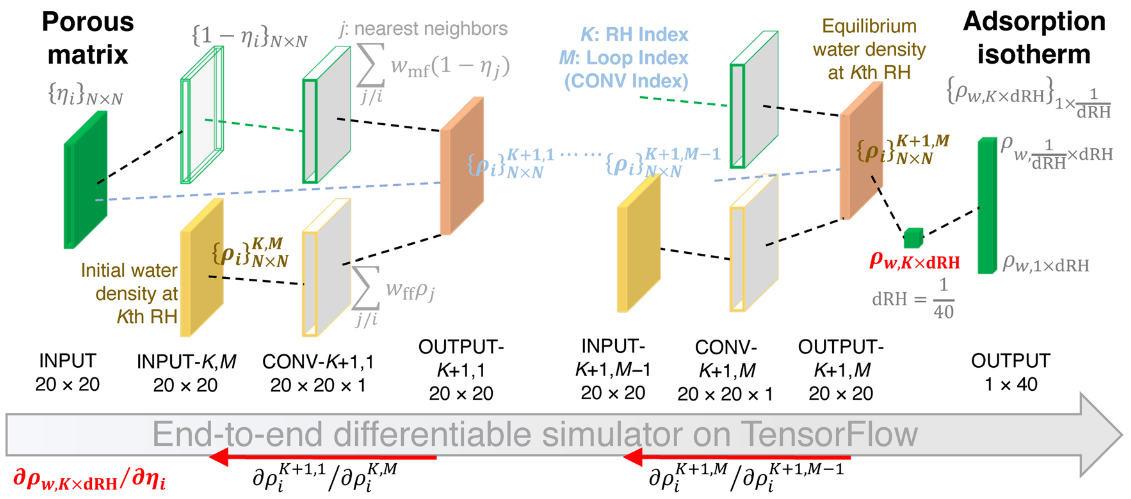
Porous matrix
$\{\eta_i\}_{N\times N}$
$\{1-\eta_i\}_{N\times N}$
j: nearest neighbors
$\sum_{j/i} w_{\mathrm{mf}}(1-\eta_j)$
K: RH Index
M: Loop Index (CONV Index)
Equilibrium water density at Kth RH
Adsorption isotherm
$\{\rho_{w,K\times \mathrm{dRH}}\}_{1\times\frac{1}{\mathrm{dRH}}}$
$\{\rho_i\}^{K+1,1}_{N\times N}$
$\{\rho_i\}^{K+1,M-1}_{N\times N}$
$\{\rho_i\}^{K+1,M}_{N\times N}$
$\rho_{w,\frac{1}{\mathrm{dRH}}\times \mathrm{dRH}}$
$\rho_{w,K\times \mathrm{dRH}}$
$\rho_{w,1\times \mathrm{dRH}}$
$\mathrm{dRH}=\frac{1}{40}$
Initial water density at Kth RH
$\{\rho_i\}^{K,M}_{N\times N}$
$\sum_{j/i} w_{\mathrm{ff}}\rho_j$
INPUT 20 × 20
INPUT-K,M 20 × 20
CONV-K+1,1 20 × 20 × 1
OUTPUT-K+1,1 20 × 20
INPUT-K+1,M–1 20 × 20
CONV-K+1,M 20 × 20 × 1
OUTPUT-K+1,M 20 × 20
OUTPUT 1 × 40
End-to-end differentiable simulator on TensorFlow
$\partial\rho_{w,K\times \mathrm{dRH}}/\partial\eta_i$
$\partial\rho_i^{K+1,1}/\partial\rho_i^{K,M}$
$\partial\rho_i^{K+1,M}/\partial\rho_i^{K+1,M-1}$

Fig. 2 End-to-end differentiable reformulation of the sorption simulation. The sorption simulation (viz., the iteration of Eq. (1)) is reformulated as a series of convolutional (CONV) layers in TensorFlow. Note that Eq. (1) is essentially a convolution operation, and M consecutive CONV layers are mathematically equivalent to iterating Eq. (1) M times until a convergence in the water density $\{\rho_i\}^{K\mathrm{th}}$ is achieved at the Kth relative humidity (RH), where RH increases from 0 to 100% with a constant increment dRH and an increment index K, namely, RH = K × dRH.

accurate as the reference simulator. Further, Fig. 3d shows the average percentage loss $\langle L\rangle$ of the differentiable simulator at $M = 100$ as a function of S_r for the validation set, where the differentiable simulator exhibits a satisfactory accuracy of $\langle L\rangle$ = ~0.36% for the whole range of S_r. Overall, these results demonstrate that, by reformulating the LDFT simulation into a succession of convolutional layers, the sorption simulator enables TPU-adaptive computing and end-to-end differentiability without accuracy deterioration.

Seamless integration of the differentiable simulator with an inverse design generator

Due to its end-to-end differentiability, the reformulated sorption simulator can be seamlessly integrated with an inverse design generator. Figure 4a shows the architecture of the generator-simulator pipeline. Taking as inputs a targeted sorption isotherm, the generator transforms the curve into a porous matrix, which is subsequently fed to the differentiable simulator to compute the real sorption curve of the generated porous matrix. Note that the generator is designed as a structure of dual, parallel deconvolution blocks, where each block is fed with half of the input curve. These two blocks aim to specifically generate small and large pores, which are saturated with water at low and large RH, respectively. More details about the architecture of the generator-simulator pipeline are provided in the Methods section. Since each layer of the pipeline is differentiable, the generator can then be optimized by gradient backpropagation in TensorFlow so as to minimize the difference between the input and output sorption curves. Note that, here, the convolutional layers of the simulator are hard-coded with fixed weights and, hence, are not optimized. This is key advantage of our approach since it avoids difficulties arising from the simultaneous optimization of the generator and predictor in traditional implementations of generative pipelines.

Training the inverse design generator by differentiable knowledge

We now focus on the training of the generator-simulator pipeline. Upon their seamless integration in gradient backpropagation, the generator is essentially trained by differentiable knowledge encoded into the simulator, rather than a preset training set, thus promoting the generator to learn the underlying physics. During the training process, a grid size $N = 20$ (i.e., 20 × 20 lattice) yields about 7 million parameters to be optimized for the generator, while the simulator comprises about 4000 convolution layers to compute. The loss function L is defined as the percentage loss between the input and output sorption curves, as illustrated in Fig. 4b. The generator is trained based on a training set of 6,400,000 sorption isotherm curves and then subsequently evaluated based on a test set of 8769 curves (see Fig. 3b). More details of the training and test sets can be found in the Methods section. Figure 4c shows the evolution of test set loss L as a function of the number of training epochs, where the batch size is set as 64 and each epoch contain 1000 batches. We find that the accuracy of the generator plateaus after 50 epochs (which corresponds to a training size of 3,200,000). Figure 4d shows the average loss $\langle L\rangle$ as a function of the reference curve sinuosity S_r in the test set after 100 training epochs. We find that the generator exhibits an average prediction loss of 3%, which is here considered very good (see below). More details of the pipeline training can be found the Methods section.

Training acceleration by tensor processing unit (TPU) computing

Considering the large depth of the simulator and the number of parameters to be optimized in the generator, the training process comes with a significant computational cost. To mitigate this issue, as a pioneering experiment, the training is conducted on TPUs[27]. TPU is a family of dedicated chips that assemble different computing units for machine learning applications[30]. Figure 5a shows a schematic of the TPU computing system composed of both software and hardware architecture, where TensorFlow is a software used to compile program ready for TPU computing on TPU chip. In contrast to general purposes processors (i.e., CPUs and GPUs), TPUs are specifically designed as matrix processors thanks to their matrix unit (MXU)[31,32]. Although TPUs have been extensively used for deep learning, their application to numerical simulations has thus far remained limited[33]. However, TPUs exhibit enough flexibility to have the potential to accelerate a broader range of computations. Figure 5b shows the training time per batch as a function of both grid size and batch size on a TPU-v3-8 chip with 8 cores and 16 GB memory per core[27]. The computational performance is compared with the training time yielded by a NVIDIA A100 GPU with 40 GB memory. All benchmarks are conducted on Google Colab using the same

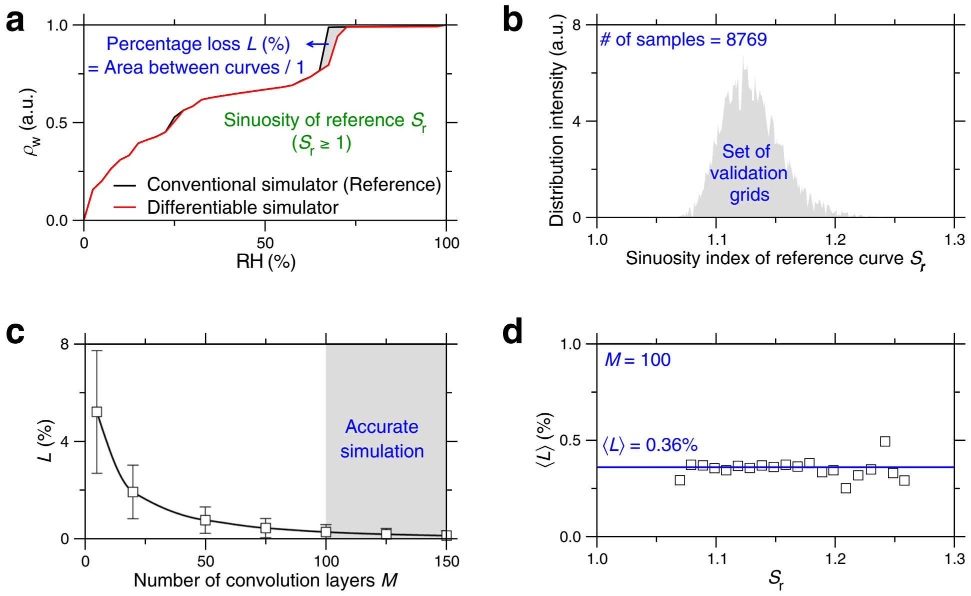

Fig. 3 Accuracy of the differentiable simulator. a Comparison between the sorption curve of a porous matrix computed by the reference (undifferentiable) sorption simulator and its reformulated differentiable counterpart. The area between curves (grey area) defines the percentage loss L. The sinuosity index of reference (black) sorption curve S_r is characterized as the ratio of the curvilinear length along the curve over the straight-line length between end points of the curve. **b** Distribution of S_r for a validation set of 8769 porous matrices. **c** Percentage loss L for the validation set as a function of the number of convolution layers M. The grey window ($M \geq 100$) indicates the range where the differentiable simulator is as accurate as the reference simulator. **d** Average percentage loss ⟨L⟩ as a function of S_r at $M = 100$. The blue line represents the average percentage loss for the validation set.

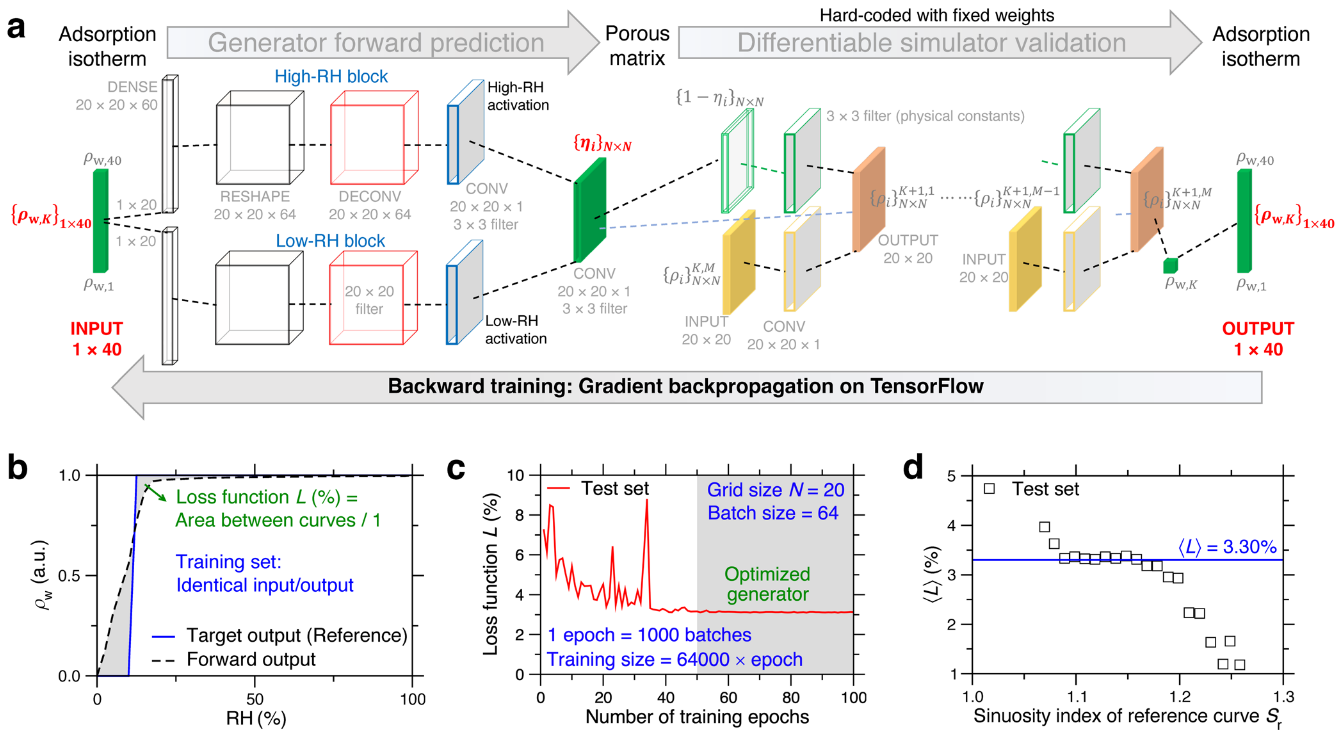
a
Adsorption isotherm
Generator forward prediction
Porous matrix
Hard-coded with fixed weights
Differentiable simulator validation
Adsorption isotherm
High-RH block
Low-RH block
DENSE 20 × 20 × 60
RESHAPE 20 × 20 × 64
DECONV 20 × 20 × 64
CONV 20 × 20 × 1 3 × 3 filter
High-RH activation
Low-RH activation
20 × 20 filter
$\{\rho_{w,K}\}_{1\times40}$
INPUT 1 × 40
OUTPUT 1 × 40
$\{\eta_i\}_{N\times N}$
$\{1-\eta_i\}_{N\times N}$
3 × 3 filter (physical constants)
$\{\rho_i\}^{K,M}_{N\times N}$
$\{\rho_i\}^{K+1,1}_{N\times N}$
$\{\rho_i\}^{K+1,M-1}_{N\times N}$
$\{\rho_i\}^{K+1,M}_{N\times N}$
$\rho_{w,K}$
Backward training: Gradient backpropagation on TensorFlow
b
Loss function L (%) = Area between curves / 1
Training set: Identical input/output
Target output (Reference)
Forward output
ρ_w (a.u.)
RH (%)
c
Test set
Grid size N = 20
Batch size = 64
Optimized generator
1 epoch = 1000 batches
Training size = 64000 × epoch
Loss function L (%)
Number of training epochs
d
Test set
⟨L⟩ = 3.30%
⟨L⟩ (%)
Sinuosity index of reference curve S_r

Fig. 4 Training an inverse design generator by the differentiable simulator. a Architecture of the generator-simulator training pipeline. The generator is designed as a structure of dual, parallel deconvolution blocks, where each block is fed with half of the input curve $\{\rho_{w,K}\}$ that represents low- and high-RH range signal, respectively. The associated porous matrix $\{\eta_i\}$ predicted by the generator is subsequently fed to the differentiable simulator for validation. The forward output of the simulator is then compared with the targeted output—which is the same as generator input $\{\rho_{w,K}\}$—to calculate the loss function used for backward training on TensorFlow. **b** Loss function L (grey area) for a target output (blue line). **c** Evolution of the test set loss L as a function of the number of training epochs. The test set contains 8769 reference sorption curves (see Fig. 3b). The plateau in the grey window indicates that the generator has reached optimal prediction performance. **d** Average test set loss ⟨L⟩ as a function of the sinuosity index of reference curve S_r at epoch = 100. The blue line represents the average loss for the test set.

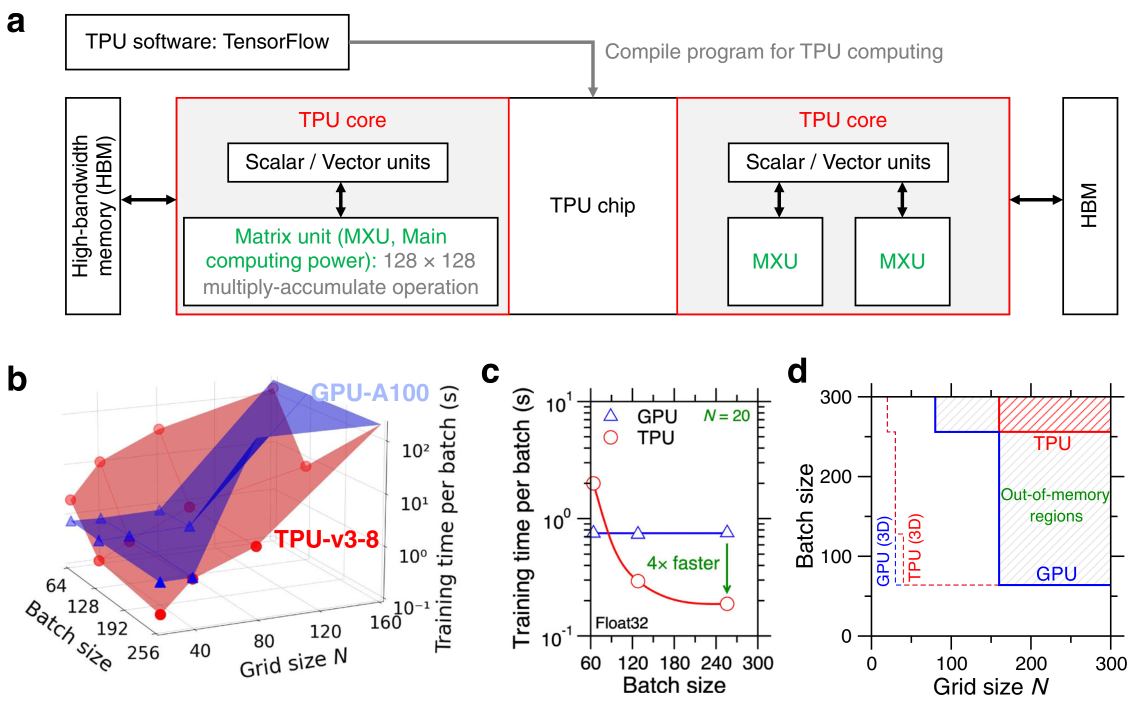

Fig. 5 Training acceleration by Tensor Processing Unit (TPU) computing. **a** Schematic of the TPU computing system composed of both software and chip hardware, where TensorFlow is a software used to compile program ready for TPU computing on TPU chip. The TPU chip contains 2 cores, and each core is an assembly of different computing units specific for machine learning, where the main computing power arises from the matrix unit (MXU) capable of 128 × 128 multiply-accumulate operation. **b** Comparison of the training time per batch as a function of the grid size *N* and batch size offered by Google's TPU-v3-8 (with 8 cores and 16 GB memory per core) and an NVIDIA A100 GPU with 40 GB memory. All benchmarks are conducted on Google Colab using the same TensorFlow code and single precision (float32). **c** Comparison of the training time per batch between TPU and GPU as a function of batch size for *N* = 20. **d** Out-of-memory regions of TPU (red line) and GPU (blue line) training for 2D (solid line) and 3D (dash line) grids in the parameter space of grid size *N* and batch size, where the lines represent the out-of-memory boundary.

TensorFlow code and single precision (float32). Figure 5c further describes the TPU and GPU training time as a function of batch size for grid size *N* = 20. We find that, especially for large grid size and batch size, the deliciated TPU hardware results in a training time that is several times faster than that offered by the GPU hardware considered herein (more than 4× faster, see Fig. 5c). These results highlight the exciting, largely untapped potential of TPU computing in accelerating computationally-intensive scientific simulations (i.e., besides traditional deep learning applications).

Note that we conduct the comparison using an up-to-date version of TPU (TPU-v3-8) and GPU (GPU-Nvidia-A100). During this comparison between TPU and GPU, their hardware architectures are rapidly updated on the way to achieve better performance[34], where TPU and GPU excel at matrix multiplication and parallel computing, respectively. Relying on their different strategies to accelerate numerical computing, we nevertheless find that the training pipeline does not gain any acceleration contribution from TPU computing with respect to GPU at the batch size of 64, which suggests that small batch size does not leverage the computing power of MXU operation and parallel compilation in TPU, as compared to the highly paralleled GPU computing. Notably, despite the fast execution speed of GPU-A100, TPU-v3-8 can outperform GPU-A100 at certain parameter settings of grid size and batch size (see Fig. 5c). In practice, we adopt a batch size of 64 for GPU training, while a batch size of 256 for TPU computing to expedite the training, and the prediction accuracy of generators under both the training settings exhibit an excellent agreement with each other.

Besides the execution speed, we further investigate the TPU and GPU memory usage under various training settings to evaluate the accessible design space of the gird model. Figure 5d shows the out-of-memory regions in the parameter space of grid size and batch size for the training pipeline in both the format of 2D grids (present format) and 3D grids (see the format below). Importantly, we find that the TPU computing generally exhibits smaller out-of-memory regions than its GPU counterpart, which echoes the fact that, for the same training pipeline, TPU hardware is delicately designed to reduce the demanding requirements of computing resources[30]. As such, TPU computing offers an attractive opportunity to access the design space that runs out of memory on GPU, promising to accelerate the training pipeline and extend its accessible design space.

Accuracy of the inverse design generator

Finally, we evaluate the accuracy of the trained generator on the test set (which comprises more than 8000 target sorption isotherms). After training, we find that the generator exhibits an average prediction loss of 3% (see Fig. 4d), which is here considered very good. Figure 6a offers an illustration of three porous matrices that are generated so as to present three archetypical sorption isotherms wherein: (i) full water saturation occurs at very low RH (which arises in the presence of very small pores), (ii) water saturation is delayed and occurs at very large RH (which is a consequence of large pores), and (iii) an intermediate case (with medium-size pores). Overall, we find that the generator model is able to predict realistic porous matrices, with expected length scales for the pores. Importantly, the simulated sorption curves of the generated porous structures exhibit all the features (in terms of trend, convexity, and value) as the target sorption curves.

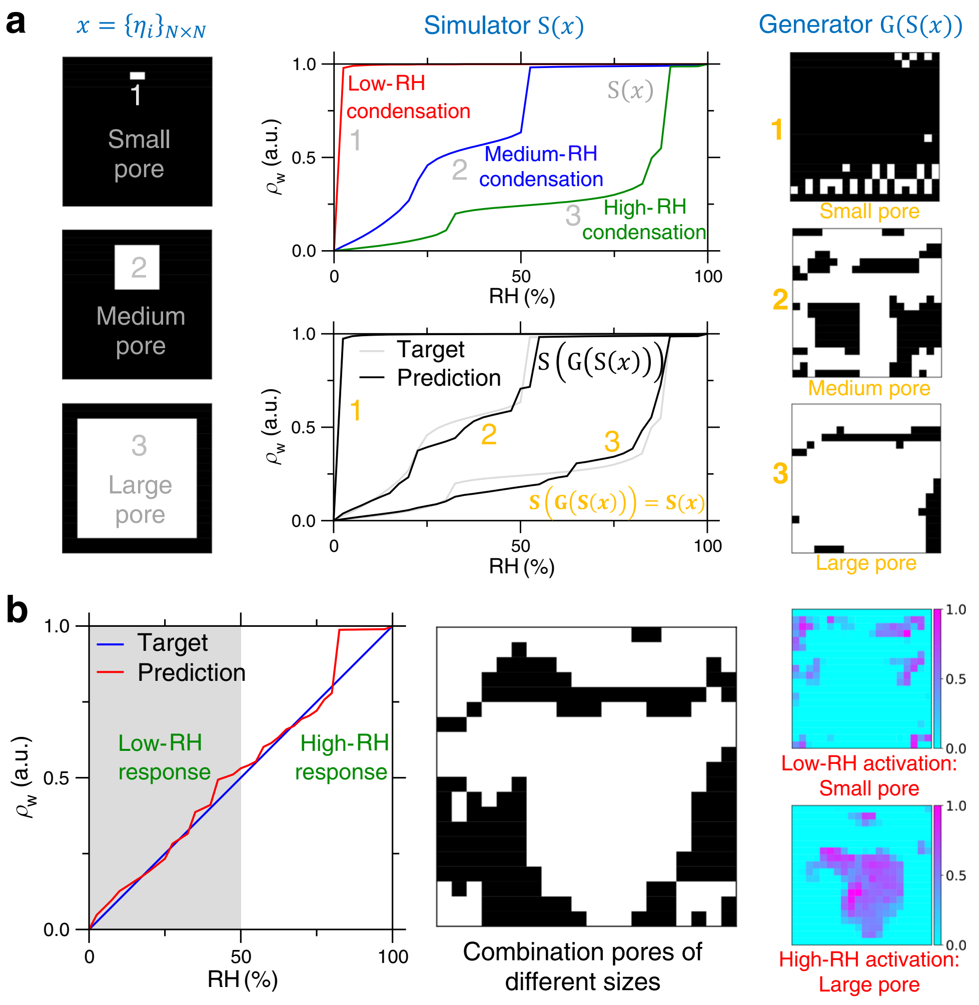

Fig. 6 Accuracy of the inverse design generator. a Illustration of three porous matrices that are generated so as to present three archetypical sorption isotherms associated with small, medium, and large pores. **b** Porous matrix generated for a target sorption curve $y = x$. The activation patterns of low- and high-RH block are also provided.

As a final test of the generator, we assess the ability of the generator to predict a porous structure featuring a target identity sorption curve $y = x$. This is an especially challenging test set case since (i) the sorption curve is not included in the training set, (ii) such a smooth sorption curve (with no sudden jump in water sorption) requires a complex, continuous pore size distribution, and (iii) this case corresponds to maximum degeneracy—unlike the cases of a 1-pixel or $(N–1) \times (N–1)$ pores, which present a limited number of possible solutions. Once again, we find that the generator yields a very realistic generated porous matrix, which, as expected, exhibits a combination of small, medium, and large pores (see Fig. 6b). Notably, the real sorption curve (computed by the simulator) of the generated porous matrix indeed exhibits a very close match with the $y = x$ target, where the curve deviation at high RH is likely ascribed to the limited capacity at grid size $N = 20$ to create giant pores that can delay the condensation up to RH = 1. This confirms that the generative model has learned the basic physical rules governing water sorption in porous media (e.g., small and large pores get saturated as low and high RH, etc.) and can successfully predict new unknown porous structures featuring tailored arbitrary sorption curves. In that regard, the fact that the generator is directly trained based on the simulator (rather than on surrogate model that approximates reality by learning from finite training set examples) is key to ensure that the generator is not limited by the accuracy of the predictor, or its ability to extrapolate predictions to grids it has never been exposed to during its training.

By designing as a dual, parallel-block structure (see Fig. 3a), the generator shows not only high prediction accuracy but also enough simplicity and interpretability out of the inductive bias (as compared to a single, giant-block structure). After training, we find that the activation patterns of these two blocks can specifically generate small and large pores that are saturated with water at low and large RH, respectively (see Fig. 6b), in agreement with the basic physics of fluid sorption that small and large pores exhibit early and delayed condensation behavior, respectively[28,29]. These results *a posterior* demonstrate that the physics-informed machine learning framework offers a reasonable balance between model accuracy, simplicity, interpretability, and extrapolability[35].

Mapping the 2D grids to 3D porous matrices

To associate the 2D grid to its 3D real material representation, we evaluate herein how much of these 2D grid designs will translate upon moving to the 3D matrices space. Note that the LDFT simulation has been well established in both the format of 2D and 3D lattice to describe capillary condensation of disordered porous materials[28,36], and in a recent study[37], we have demonstrated that the 2D lattice model can offer water sorption behaviors in excellent agreement with the water sorption isotherms in cement

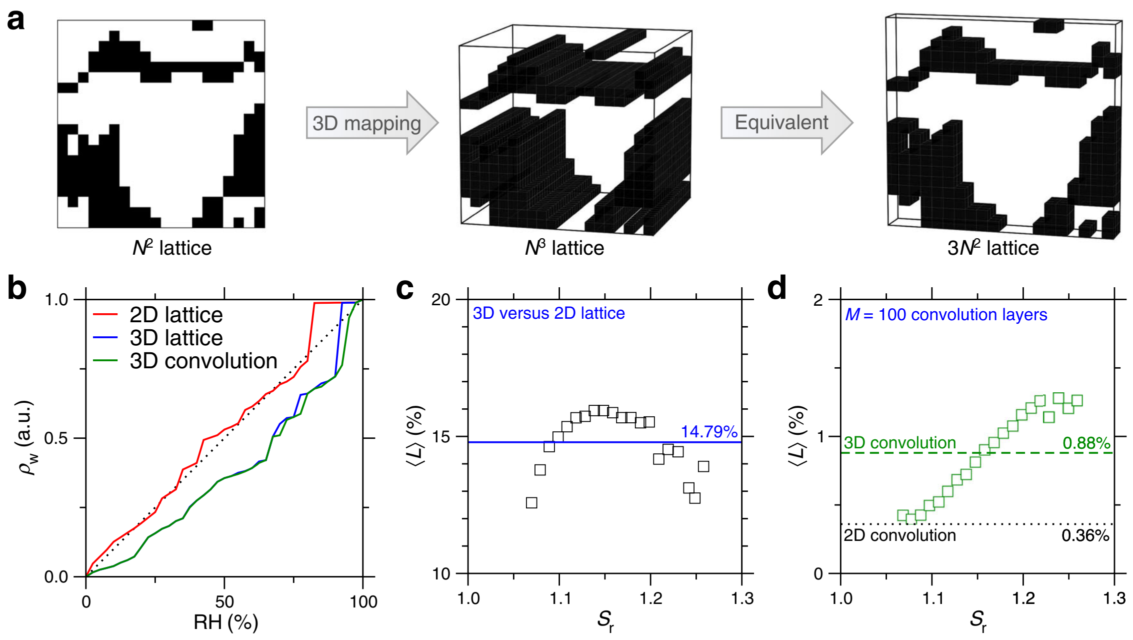

Fig. 7 Mapping the 2D grids to 3D matrices. a Illustration of 3D representation for the predicted 2D grid targeting the sorption curve $y = x$. The 3D cubic lattice has 6 nearest neighbors per voxel and is equivalent to a multilayer grid (≥3 layers) under periodic boundary condition. **b** Comparison of the sorption curve between the 2D and 3D lattice, where the sorption curves are computed by conventional LDFT simulation. The result of differentiable 3D lattice simulation using $M = 100$ consecutive 3D convolution layers is also added for comparison. **c** Average percentage loss ⟨L⟩ of sorption curve when translating from 2D grids to their 3D representations, as a function of the sinuosity index of reference curve S_r in the validation set. The blue line represents the average loss for the validation set. **d** Average percentage loss ⟨L⟩ of sorption curve between conventional and differentiable 3D lattice simulation, as a function of S_r at $M = 100$. The green dash represents the average loss for the validation set. The average loss for differentiable 2D lattice simulation (black dash) is added as a reference.

pastes. However, the abstract space of 2D grid is an extensive simplification of the real material space, while real materials are significantly complicated in terms of system size, dimensionality, and pixel state. These complexities make the simulation of real materials heavily rely on the capability of computing hardware, including execution speed and memory limitation.

Despite its simplicity, the 2D grid simulation nevertheless provides an indicative representation of its real material counterpart formulated by 3D porous matrix. Figure 7a provides an example of mapping an 2D grid to its 3D matrix representation, where the 2D convolution operation (with 4 nearest neighbor pixels) in LDFT simulation is modified as 3D convolution (with 6 nearest neighbor voxels), and the sorption isotherms for the 2D and 3D lattice are shown in Fig. 7b. Figure 7c further provides their average percentage loss ⟨L⟩ as a function of the reference curve sinuosity S_r in the validation set. We find that the percentage loss of translation from 2D to 3D lattices' sorption isotherms is about 14.8% for the whole range of S_r. This translation loss is not trivial but indicates that the sorption isotherms exhibit very similar trend when translated from 2D to 3D lattices. Overall, these results illustrate the close correlation between the toy 2D model and its 3D real material representation.

Generalization to 3D porous matrices' inverse design

Besides mapping 2D grid to its 3D representation, we investigate herein the direct applicability of this inverse design approach to the real material space formulated by 3D porous matrix. To this end, relying on the same generator–simulator pipeline (see Fig. 4a), the sorption simulation of 3D porous matrices is readily implemented by modifying the 2D convolution layer to 3D convolution layer, where we find that the differentiable 3D lattice simulator is as accurate as the conventional LDFT simulator and exhibits a miniscule average percentage loss of 0.88% for the validation set (see Fig. 7d). Then the generator is designed to predict a cubic grid with grid size $N = 10$, as illustrated in Fig. 8a, which is then fed into the differentiable 3D lattice simulator for validation. More details about the architecture of the generator-simulator pipeline are provided in the Methods section. Using the same training scheme as that of 2D grid model, Fig. 8b shows the average loss ⟨L⟩ as a function of the reference curve sinuosity S_r in the test set after 100 training epochs. We find that the generator exhibits an average prediction loss of 2.9%, close to the prediction loss of 3.3% in 2D grid model. These results demonstrate that the 3D generator excels at inverse design in 3D space to generate 3D porous matrices with target sorption isotherms. As such, the inverse design approach can flexibly extend to 3D grid model in real material space.

However, compared to 2D model training, 3D model training at the same gird size N incorporates a new dimension's computation that theoretically increases the computational cost N times. This constitutes a significant burden on the execution speed and memory for this 3D workflow. To rationalize the applicability of this approach to 3D design space, we evaluate herein the execution speed and memory usage of 3D grid model training on both TPU and GPU hardware. Using the same benchmark scheme as that of 2D grid model (see Fig. 5b), Fig. 8c shows the comparison of the training time per batch as a function of the grid size N and batch size offered by TPU-v3-8 and GPU-A100, and Fig. 8d further describes the TPU and GPU training time as a function of batch size for grid size $N = 10$. We find that, at large batch size of 256, the TPU computing can modestly expedite the training to be slightly faster than GPU training (1.1× faster, see Fig. 8d), in harmony with the faster TPU training for 2D grid model. Despite their similarity in TPU acceleration, 2D and 3D grid model exhibits a huge difference of computational burden at the same grid size, which renders the 3D model easily run out of memory on the current version of TPU and GPU (see Fig. 5d), with the

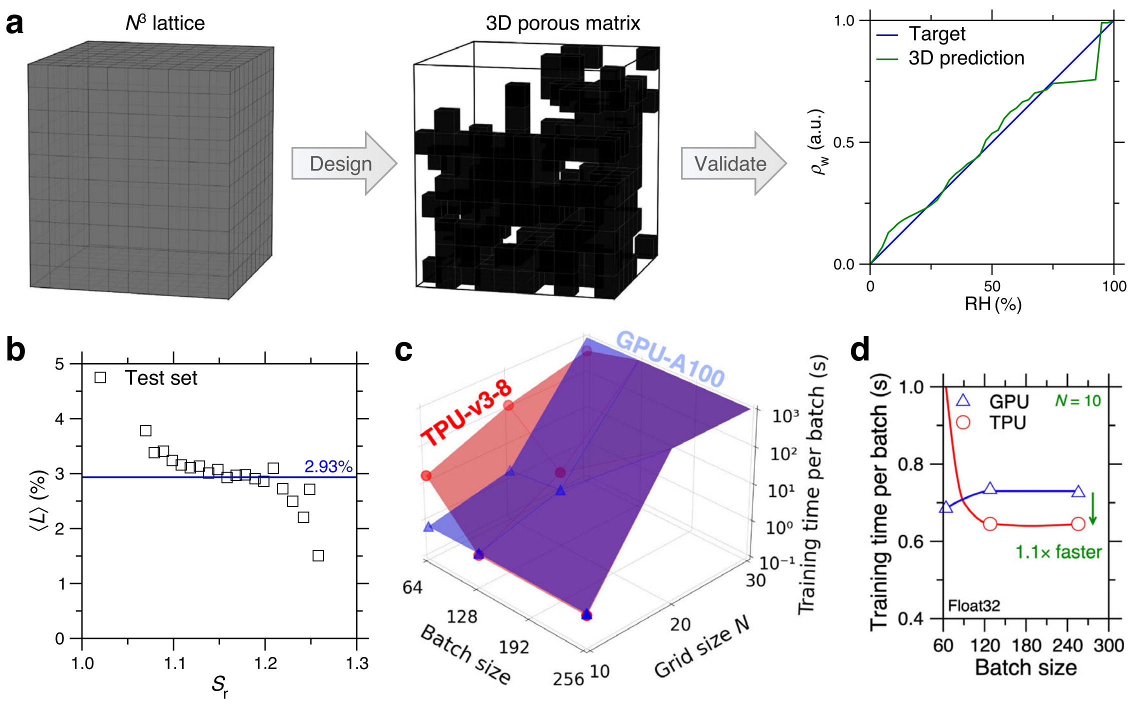

Fig. 8 Generalization to 3D porous matrices' inverse design. a Illustration of applying the generator-simulator pipeline to inversely design a 3D porous matrix targeting the sorption curve $y = x$. The 3D lattice generator predicts a cubic grid with grid size $N = 10$, which is then fed into the differentiable 3D lattice simulator for validation. **b** Average test set loss $\langle L \rangle$ as a function of the sinuosity index of reference curve S_r after 100 training epochs. The blue line represents the average loss for the test set. **c** Comparison of the training time per batch as a function of the grid size N and batch size offered by TPU-v3-8 and GPU-A100. **d** Comparison of the training time per batch between TPU and GPU as a function of batch size for $N = 10$.

maximum grid size $N = 20$ for GPU and $N = 30$ for TPU in the accessible design space, which, however, illustrates the TPU potentiality in requiring less computing resource than GPU to extend the 3D design space thereof.

Inverse design of porous solids with target hysteresis behavior

Finally, relying on the 3D model, we extend this approach to design porous solids with target hysteresis behavior. Figure 9a offers an illustration of the adsorption and desorption process in a 3D porous matrix, and its hysteresis curve is provided in Fig. 9b. By inputting the entire hysteresis curve into the generator–simulator pipeline, the loss function L is defined as the sum of the percentage loss for both the adsorption and desorption isotherm, as illustrated in Fig. 9c. The training and test sets are the same as before except that each sorption isotherm is followed by a desorption isotherm to form the hysteresis curve. Details about the preparation of training and test sets are provided in the Methods section. Using the same training scheme as before, Fig. 9d shows the average loss $\langle L \rangle$ as a function of the sinuosity index of adsorption isotherm S_r in the test set after 100 training epochs. We find that the generator exhibits an average prediction loss of 4.7% for the whole range of S_r, which is here considered very good.

Figure 10 offers an illustration of porous matrices generated for various target hysteresis behaviors. By inputting a linear curve $y = x$ with zero hysteresis, the generator provides a porous matrix exhibiting nearly linear adsorption isotherm but a pronounced hysteresis (see Fig. 10a)—which turns out an intrinsic phenomenon rooted in the complex metastability of disordered porous solids[28,36], so that the hysteresis cannot be an arbitrary design. By inputting the output (real) hysteresis curve, Fig. 10a shows the porous matrix generated to present this target input hysteresis, and the predicted hysteresis offers a close match to its input curve. As a further validation, Fig. 10b offers an illustration of porous matrices generated to present the target hysteresis behaviors associated to small and large pore matrix, respectively, where, indeed, the input and output hysteresis agree well with each other. Overall, these results highlight the flexibility of this approach to extend to complex curve properties such as hysteresis behavior.

DISCUSSION

Overall, this work establishes a robust pipeline to enable the inverse design of materials by leveraging an end-to-end differentiable simulation as predictor. The fact that the generator is directly trained based on a simulator rather than on a surrogate machine learning model is key to ensure that the generator is not limited by the accuracy or extrapolation ability of the predictor. Compared to previous inverse design approaches using optimization or traditional generative models[5–7], the present approach eliminates the prerequisite of a predefined dataset by integrating differentiable knowledge into the inverse design pipeline, so that a material's inverse design is unbiased to the initial dataset and can accurately extrapolate to a design space away from the training space. As a key enabler of this approach, we adopt TPUs to accelerate the training of the generator by gradient backpropagation in TensorFlow. This illustrates the exciting possibilities of TPU computing to accelerate scientific numerical simulations. Although our results are built upon a toy sorption model, this research has several scientific and societal implications. First, this work illustrates the benefits of integrating differentiable simulations in machine learning pipelines—which is key to accelerate the discovery of new materials. Second, our results establish TPU computing as a promising route to accelerate scientific simulations, which are ubiquitous in various applications (drug discovery by molecular dynamics, architectural design by finite element method, weather forecast predictions, etc.)[1–3]. Finally, the ability to design new porous structures with tailored sorption isotherms

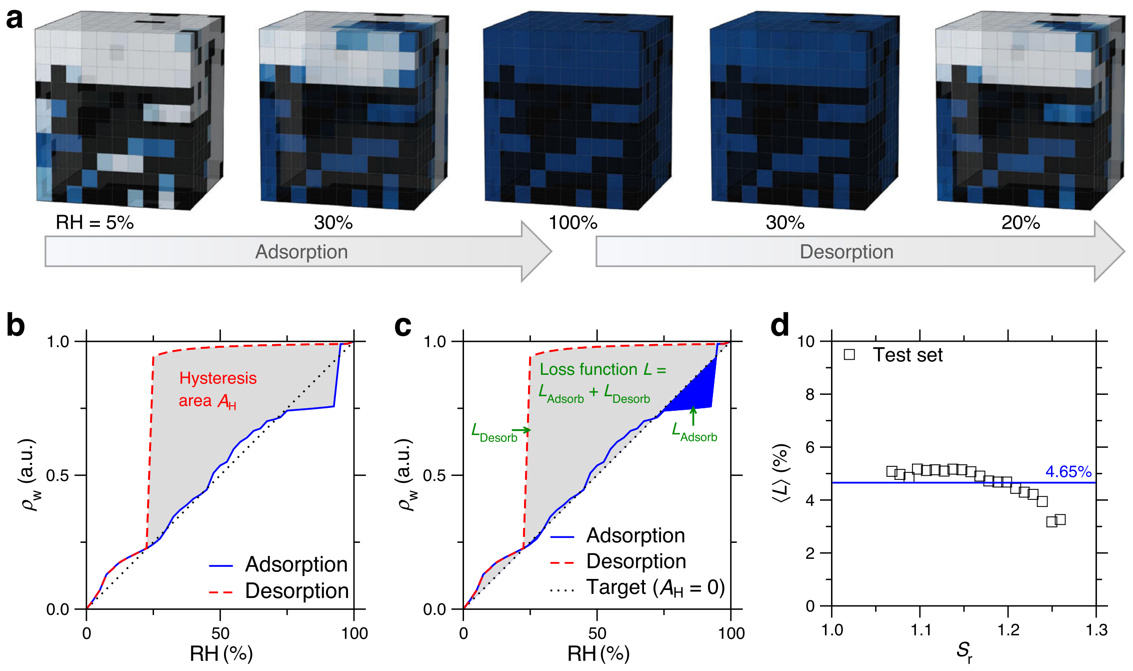

Fig. 9 Inverse design of porous solids with target hysteresis behavior. a Illustration of the adsorption and desorption process in a 3D porous matrix targeting the adsorption curve $y = x$. **b** Sorption hysteresis behavior of the 3D matrix, where the area between adsorption and desorption isotherm is defined as the hysteresis area A_H. **c** Loss function L for a target output, where L consists of both the percentage loss of adsorption isotherm L_{Adsorb} and desorption isotherm L_{Desorb}. The target output is herein set as a linear curve $y = x$ with $A_H = 0$. **d** Average test set loss $\langle L \rangle$ as a function of the sinuosity index of adsorption isotherm S_r after 100 training epochs. The blue line represents the average loss for the test set.

could leapfrog several important applications, including for CO_2 capture[38,39] and gas separation[40,41]. In addition, designing new porous structures featuring a smooth, continuous sorption isotherm (i.e., as close as possible to the $y = x$ target used herein) is important for drug delivery applications, to ensure that drugs are continuously released at a constant rate in a given environment[42,43].

METHODS

Lattice density function theory (LDFT) of sorption

We consider a simplified model of porous matrix by using (i) a square N-by-N lattice (see Fig. 1a) and (ii) a cubic N^3 lattice (see Fig. 8a). In this lattice, the state of each pixel (or voxel) i is given by the knowledge of (η_i, ρ_i), where $\eta_i = 0$ and 1 indicate that the pixel is filled with solid or is a pore, respectively, and ρ_i is the density of water in the pore upon increasing relative humidity (RH). In the scenario of water sorption in nanoporous grids, the pixel size is approximately the size of one single water molecules, i.e., 3 Å per dimension, and the sorption isotherm is dependent on pore pixel distribution and the resultant pore size (rather than grid size N)[37]. $\rho_i = 0$ and 1 denote that the pore is fully empty or saturated with water, respectively. Based on the LDFT formalism, the water density ρ_i at a given pixel i is given by:

$$\rho_i = \frac{\eta_i}{1 + e^{-\{\mu + \sum_{j/i}[w_{ff}\rho_j + w_{mf}(1-\eta_j)]\}/(kT)}} \tag{1}$$

where μ is the chemical potential (which depends on RH), k is the Boltzmann constant, T is the temperature, w_{ff} is interaction energy between two neighboring pixels that are filled with water, w_{mf} is the interaction energy between a pixel filled with water and a substrate (i.e., a neighboring pixel filled with solid), and j/i are the pixel IDs of the 4 neighbors of pixel i (note that, to avoid any surface effect, periodic boundary conditions are applied)[29]. Details of the LDFT formulism behind Eq. (1) are provided in the following.

According to LDFT[28,29], the equilibrium $\{\rho_i\}$ at a given RH is computed by minimizing the configuration's grand potential function $\Omega(\{\rho_i\})$:

$$\begin{aligned}\Omega(\{\rho_i\}) =& kT\sum\nolimits_i[\rho_i \ln(\rho_i) + (\eta_i - \rho_i)\ln(\eta_i - \rho_i)] \\ & - w_{ff}\sum\nolimits_{\langle ij\rangle}\rho_i\rho_j - w_{mf}\sum\nolimits_{\langle ij\rangle}[\rho_i(1-\eta_j) + \rho_j(1-\eta_i)] \\ & - \mu\sum\nolimits_i \rho_i\end{aligned} \tag{2}$$

where $\langle ij\rangle$ indicates the sums are restricted to distinct nearest neighbor pairs, μ is calculated by $\mu = \mu_{sat} + kT\ln(\text{RH})$, where $T = 298$ K, μ_{sat} is the chemical potential of water at saturated state and can be estimated as $\mu_{sat} = w_{ff} \times c/2$ (here, the coordination number $c = 4$ for 2D lattice and $c = 6$ for 3D lattice). w_{ff} is derived from the critical temperature of water ($T_c = 647$ K) and $w_{ff} = 4kT_c/c$. w_{mf} is calculated from the interaction ratio parameter $y = w_{mf}/w_{ff}$, where $y > 1$ indicates a hydrophilic substrate and $y < 1$ a hydrophobic substrate (here, $y = 1.5$). By minimizing Eq. (2) with respect to $\{\rho_i\}$, the solution of equilibrium $\{\rho_i\}$ is rewritten as an iteration loop of Eq. (1) until convergence, where the convergence condition is set as $1/N^2\sum_i(\rho_i^{(t+1)} - \rho_i^{(t)}) < 10^{-10}$ between two consecutive loop t and $t+1$. ρ_i is calculated at each RH for RH = 0-to-100% with an increment dRH (here, dRH = 2.5%). Initially, the equilibrium $\rho_i = 0$ when RH = 0. At each increment step K, the equilibrium water density values $\{\rho_i\}^{K\text{th}}$ at RH = $K \times$ dRH serve as the starting configuration to calculate $\{\rho_i\}^{K+1}$ at the subsequent step $K + 1$ by iteratively applying Eq. (1) until a convergence in the $\{\rho_i\}$ values is obtained. Finally, the water sorption isotherm $\{\rho_w\}_{1\times 40}$ is obtained by calculating the average pore water density $\rho_w = \langle\rho_i\rangle = 1/N^2\sum_i(\rho_i)$ at each of the 40 RH increments. Relying on the same computation process, we flip the 40 RH values to obtain the desorption isotherm and the hysteresis curve thereof $\{\rho_w\}_{1\times 80}$. More detailed descriptions of LDFT of sorption can be found in Ref. [29].

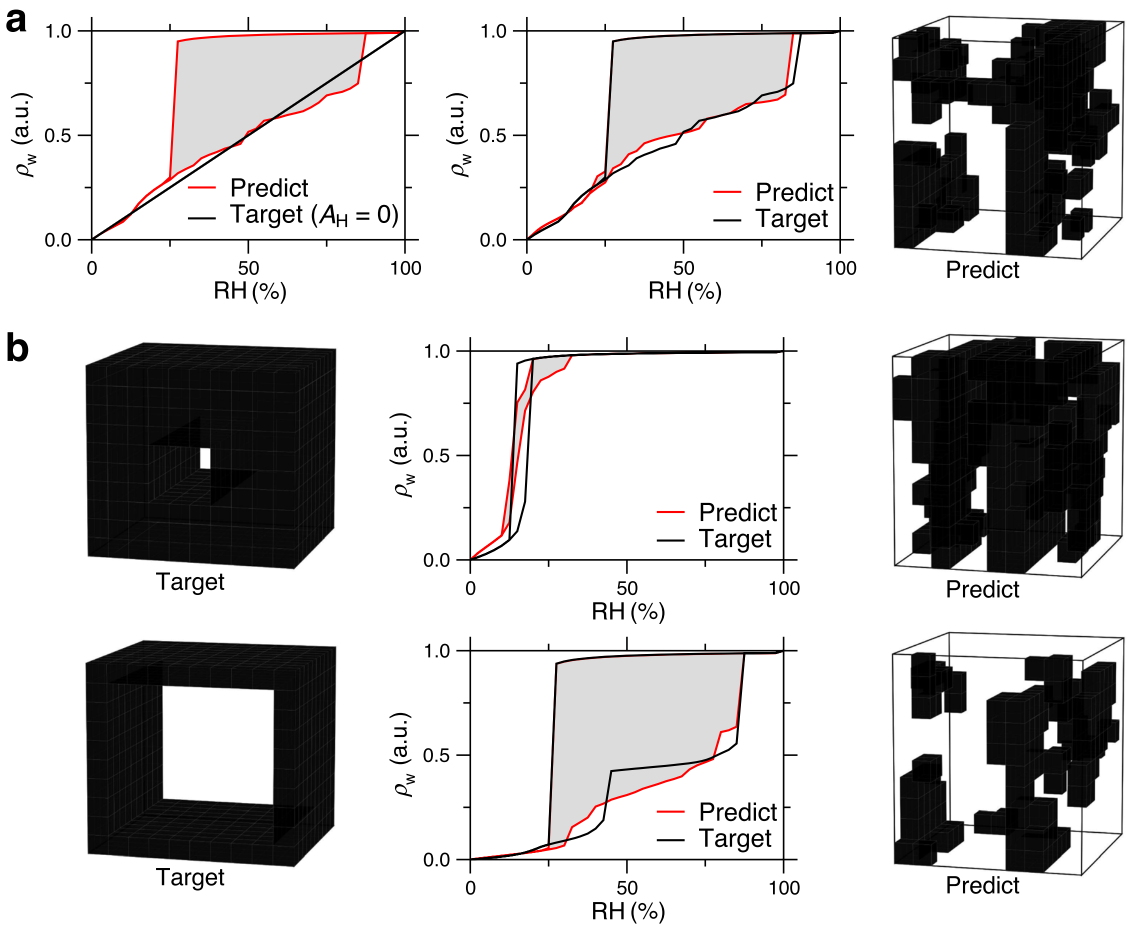

Fig. 10 Accuracy of the generator using hysteresis behavior input. a Porous matrix generated for a target hysteresis behavior that is associated to a porous matrix pre-generated by inputting a linear curve $y = x$ with $A_H = 0$. **b** Porous matrices generated for target hysteresis behaviors that are associated to a small and large pore matrix, respectively.

Architecture of the generator-simulator pipeline

The generator is designed as a structure of dual, parallel deconvolution blocks (see Fig. 4a), where each block is fed with half of the input sorption isotherm $\{\rho_w\}_{1\times40}$ or hysteresis curve $\{\rho_w\}_{1\times80}$. In detail, the low- and high-RH block is fed with the low- and high-RH half signal of the input $\{\rho_w\}$, i.e., a 1-by-20 array each for adsorption isotherm or 1-by-40 array each for hysteresis curve. Then the two blocks show the same structure, which consists of 4 layers in series, that is, (i) a fully connected dense neural layer (DENSE) that contains $20 \times 20 \times 64 = 25600$ neurons and outputs a 1-by-25600 array for 2D lattice or that contains $10 \times 10 \times 10 \times 8 = 8000$ neurons and outputs a 1-by-8000 array for 3D lattice; (ii) a reshape layer (RESHAPE) that transforms the one dimensional 1-by-25600 for 2D lattice or 1-by-8000 for 3D lattice array into a three dimensional 20-by-20-by-64 for 2D lattice or a four dimensional array 10-by-10-by-10-by-8 for 3D lattice array; (iii) a deconvolution layer (DECONV) that contains 64 channels with a 20×20 filter size and outputs a three dimensional 20-by-20-by-64 array for 2D lattice or that contains 8 channels with a 10×10 filter size and outputs a four dimensional 10-by-10-by-10-by-8 array for 3D lattice; (iv) a convolution layer (CONV) that contains 1 channel with a 3×3 filter size and outputs a two-dimensional 20-by-20 array for 2D lattice or a three-dimensional 10-by-10-by-10 array for 3D lattice. The activation function of each layer is set as 'ReLU' function, and batch normalization has been applied to the output of each layer to accelerate the training process[44]. Finally, the two 20-by-20 array for 2D lattice or 10-by-10-by-10 array for 3D lattice obtained from the low- and high-RH block—which are denoted here as low- and high-RH activation, respectively—are concatenated in parallel and are fed into the generator's output layer, namely, a convolution layer that contains 1 channel, uses a 3×3 filter size and a 'binary sigmoid' activation function, and outputs a 20-by-20 prediction gird $\{\eta_i\}_{20\times20}$ for 2D lattice or a 10-by-10-by-10 prediction grid $\{\eta_i\}_{10\times10\times10}$ for 3D lattice. The generator output $\{\eta_i\}$ is then fed into the differentiable simulator for validation. This configuration would go through $M = 100$ consecutive blocks of TensorFlow-based layers (i.e., the decomposed operations of Eq. (1) programmed in TensorFlow, see Fig. 2) at each of the 40 RH increments to obtain the output sorption isotherm $\{\rho_w\}_{1\times40}$ and, if needed, continue the computation at each of the flipped 40 RH points to obtain the desorption isotherm and the output hysteresis curve thereof $\{\rho_w\}_{1\times80}$.

Preparation of the training and test sets

The training set contains 6,400,000 target sorption curves. These curves are generated automatically from a self-defined generative function. This function aims to produce as many as possible curves that are monotonically non-decreasing but vary differently in terms of trend, convexity, and value. Although this generative curve are not real sorption isotherms, they possess most important features of real sorption curves and cover all possible variations of real sorption isotherms. There are different ways to define the generative function. Here we propose one type of generative function that satisfy the above requirements. This function generates 20% 'stepwise' curves and 80% 'anchor-based' curves. According to the general classification of sorption isotherms[45,46], the 'stepwise' and 'anchor-based' generative function enable an efficient description of, respectively, the saturation and monotonicity feature in real sorption isotherms. By discretizing the curve as a one dimensional 1-by-40 array $\{\rho_w\}_{1\times 40}$, the 'stepwise' curves are designed as an array where the first n elements $= 0$ and the last $(40-n)$ elements $= 1$, where the integer n is uniformly randomized from 1 to 39. The 'anchor-based' curves are designed by first defining an 'anchor' element from the 1-by-40 array, where the anchor is the n-th element, and we uniformly randomize its index n from 1 to 39 and its value A from 0 to 1. Then, regarding all the elements before the anchor, the increment D of their value between two consecutive elements can be expressed as $D = \mathrm{e}^R/\sum_n \mathrm{e}^R \times A$, where R is a random number sampled from a normal distribution with a zero mean and a standard deviation of σ (here, $\sigma = 4$). Similarly, regarding all the elements after the anchor, the increment D of their value between two consecutive elements can be expressed as $D = \mathrm{e}^R/\sum_{40-n} \mathrm{e}^R \times (1 - A)$. Both the 'stepwise' and 'anchor-based' curves can be generated efficiently to create a large training set covering a diverse population of sorption curves. The hysteresis curves are generated using the same 'stepwise' and 'anchor-based' generative function by setting the desorption curve ahead of the adsorption curve. The stepwise desorption curve inherits all features from the stepwise adsorption curve except that the maximum index of zero elements is no larger than n. The anchor-based desorption curve inherits all features from the anchor-based adsorption curve except that the anchor value is no less than A.

Finally, the test set are real sorption curves to evaluate the generator's prediction accuracy. Here, we create a test set that contains 8769 real curves. These curves are generated by the sorption simulator using a large set of grids (see Fig. 3b), which includes 8769 diverse and random grid patterns. These random grids are generated using the following strategy: By providing some reference curves and initial grid configurations, the test set grids are generated by varying the initial grids to approach the reference sorption curves using particle swarm optimization—where the grid variation rule is based on the competition between local best grid and global best grid (see Refs. [47–49] for the mathematical formulation), and the variations end up when the local and global best grids become the same. Note that the end grids are highly dependent on the selection of initial grids and may exhibit sorption curves away from the reference curves. Since the variation paths are biased to their initial grids—which are generated on purpose to exhibit pronounced differences, the grids on the different variation paths would inherently remain distinctive in pore patterns (including different symmetries). All these grids during the variation are collected to constitute the test set of 8769 girds.

Training of the generator-simulator pipeline

In the training process, we first set the grid size $N = 20$ for 2D lattice and $N = 10$ for 3D lattice, and the batch size $= 64$ for GPU training and 256 for TPU training acceleration. Then we train the pipeline for 100 epochs and each epoch contains 1000 batches. The loss function used herein is the percentage loss L between the forward output and the reference target curve (see Fig. 4b), that is, the area between the forward curve and the reference curve. Note that, since both the solid phase and pore phase in a porous matrix shows some continuity within their phase, some regularization term can be applied to the training process to simultaneously accelerate the training and improve the prediction accuracy[44]. Here, the regularization term designed for the generator output is defined as $\sum_i^{N^2} \sum_{j/i} |\eta_i - \eta_j| / 4N^2$ for 2D lattice and $\sum_i^{N^3} \sum_{j/i} |\eta_i - \eta_j| / 6N^3$ for 3D lattice, which panelizes a solid site neighbored by a pore, or vice versa. In other words, the generator's output gird would favor continuous solid phase or continuous pore phase. Then we select the Stochastic Gradient Descent (SGD) optimizer to minimize the loss function[44]. The momentum is set as 0.9 to accelerate gradient descent[20]. The learning rate is initially set as 10^{-2} and gradually decays by a factor of 0.1 after a patience of 10 epochs[20]. Finally, a validation step is applied to the pipeline after each training epoch using the test set of 8769 sorption curves.

DATA AVAILABILITY

All data needed to evaluate the conclusions of this study are present in the paper, and all relevant data are available from Dr. Han Liu upon reasonable request.

CODE AVAILABILITY

All codes needed to evaluate the conclusions in the paper are provided at GitHub/SOFT-AI-Lab and available from Dr. Han Liu upon reasonable request.

Received: 31 December 2022; Accepted: 2 July 2023;
Published online: 13 July 2023

REFERENCES

1. Levchenko E. V., Dappe Y. J. & Ori G. Theory and simulation in physics for materials applications: cutting-edge techniques in theoretical and computational materials science. In *Springer Series in Materials Science*, Vol. 296, 1–286 (Springer Cham, 2020).
2. Agrawal, A. & Choudhary, A. Perspective: materials informatics and big data: Realization of the "fourth paradigm" of science in materials science. *APL Mater* **4**, 053208 (2016).
3. Mauro, J. C. Decoding the glass genome. *Curr. Opin. Solid State Mater. Sci.* **22**, 58–64 (2018).
4. Pyzer-Knapp, E. O., Suh, C., Gómez-Bombarelli, R., Aguilera-Iparraguirre, J. & Aspuru-Guzik, A. What is high-throughput virtual screening? A perspective from organic materials discovery. *Annu. Rev. Mater. Res.* **45**, 195–216 (2015).
5. Sanchez-Lengeling, B. & Aspuru-Guzik, A. Inverse molecular design using machine learning: generative models for matter engineering. *Science* **361**, 360–365 (2018).
6. Liao, T. W. & Li, G. Metaheuristic-based inverse design of materials – A survey. *J. Materiomics* **6**, 414–430 (2020).
7. Noh, J., Gu, G. H., Kim, S. & Jung, Y. Machine-enabled inverse design of inorganic solid materials: promises and challenges. *Chem. Sci.* **11**, 4871–4881 (2020).
8. Liu, H., Fu, Z., Yang, K., Xu, X. & Bauchy, M. Machine learning for glass science and engineering: a review. *J. Non-Cryst* **4**, 100036 (2019).
9. Plimpton, S. Fast parallel algorithms for short-range molecular dynamics. *J. Comput. Phys.* **117**, 1–19 (1995).
10. Wang, W., Axelrod, S. & Gómez-Bombarelli, R. Differentiable molecular simulations for control and learning. In *Machine Learning for Molecules Workshop at NeurIPS 2020*. No. 32 (2020).
11. Schoenholz, S. S. & Cubuk, E. D. JAX, M.D. A framework for differentiable physics. In *Advances in Neural Information Processing Systems*. Vol. 33, 11428–11441 (2020).
12. Chen, C.-T. & Gu, G. X. Generative deep neural networks for inverse materials design using backpropagation and active learning. *Adv. Sci.* **7**, 1902607 (2020).
13. Dan, Y. et al. Generative adversarial networks (GAN) based efficient sampling of chemical composition space for inverse design of inorganic materials. *NPJ Comput. Mater.* **6**, 1–7 (2020).
14. Kingma, D. P. & Welling, M. An introduction to variational autoencoders. *Found. Trends Mach. Learn.* **12**, 307–392 (2019).
15. Goodfellow, I. et al. Generative Adversarial Nets. In advances in neural information processing systems 27 (2014).
16. Kim, B., Lee, S. & Kim, J. Inverse design of porous materials using artificial neural networks. *Sci. Adv.* **6**, eaax9324 (2020).
17. Liu, Z., Zhu, D., Rodrigues, S. P., Lee, K.-T. & Cai, W. Generative model for the inverse design of metasurfaces. *Nano Lett* **18**, 6570–6576 (2018).
18. Paszke, A. et al. PyTorch: an imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*. Vol. 32, No. 4399 (2019).

19. Griewank, A. & Walther, A. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation,* 2nd edn. (Society for Industrial and Applied Mathematics, 2008).
20. Abadi, M. et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. In *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation* 265–283 (2016).
21. Frostig, R., Johnson, M. J. & Leary, C. Compiling machine learning programs via high-level tracing. In SysML Conference 2018, February 2018, Stanford, CA USA.
22. Hu, Y. et al. DiffTaichi: Differentiable Programming for Physical Simulation. in International Conference on Learning Representations (2020).
23. Hernández, A. & Amigó, J. M. Differentiable programming and its applications to dynamical systems. Preprint at https://doi.org/10.48550/arXiv.1912.08168 (2020).
24. de Avila Belbute-Peres, F., Smith, K., Allen, K., Tenenbaum, J. & Kolter, J. Z. End-to-End Differentiable Physics for Learning and Control. In *Advances in Neural Information Processing Systems*. Vol. 31, No. 3572 (2018).
25. Hu, Y. et al. ChainQueen: A Real-Time Differentiable Physical Simulator for Soft Robotics. In 2019 International Conference on Robotics and Automation (ICRA) 6265–6271 (2019).
26. Liu, H. et al. End-to-End Differentiability and Tensor Processing Unit Computing to Accelerate Materials' Inverse Design. In Machine Learning for Engineering Modeling, Simulation, and Design Workshop @ NeurIPS (2020).
27. Google Cloud Tensor Processing Units (TPUs). https://cloud.google.com/tpu.
28. Kierlik, E., Monson, P. A., Rosinberg, M. L. & Tarjus, G. Adsorption hysteresis and capillary condensation in disordered porous solids: a density functional study. *J. Phys.: Condens. Matter* **14**, 9295–9315 (2002).
29. Kierlik, E., L. Rosinberg, M., Tarjus, G. & Viot, P. Equilibrium and out-of-equilibrium (hysteretic) behavior of fluids in disordered porous materials: theoretical predictions. *Phys. Chem. Chem. Phys.* **3**, 1201–1206 (2001).
30. Wang, Y. E., Wei, G.-Y. & Brooks, D. Benchmarking TPU, GPU, and CPU Platforms for Deep Learning. Preprint at https://doi.org/10.48550/arXiv.1907.10701 (2019).
31. Lu, T., Chen, Y.-F., Hechtman, B., Wang, T. & Anderson, J. Large-scale discrete fourier transform on TPUs. *IEEE Access* **9**, 93422–93432 (2021).
32. Huot, F., Chen, Y.-F., Clapp, R., Boneti, C. & Anderson, J. High-resolution imaging on TPUs. Preprint at https://doi.org/10.48550/arXiv.1912.08063 (2019).
33. Yang, K., Chen, Y.-F., Roumpos, G., Colby, C. & Anderson, J. High performance Monte Carlo simulation of ising model on TPU clusters. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (2019).
34. Jouppi, N. P. et al. TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. No. 82, 1–14 (2023).
35. Liu, H. et al. Predicting the dissolution kinetics of silicate glasses by topology-informed machine learning. *Npj Mater. Degrad.* **3**, 1–12 (2019).
36. Kierlik, E., Monson, P. A., Rosinberg, M. L., Sarkisov, L. & Tarjus, G. Capillary condensation in disordered porous materials: hysteresis versus equilibrium behavior. *Phys. Rev. Lett.* **87**, 055701 (2001).
37. Zhang, Y., Liu, H., Zhao, C., Ju, J. W. & Bauchy, M. Deconstructing water sorption isotherms in cement pastes by lattice density functional theory simulations. *J. Am. Ceram. Soc.* **104**, 4226–4238 (2021).
38. Sneddon, G., Greenaway, A. & Yiu, H. H. P. The potential applications of nanoporous materials for the adsorption, separation, and catalytic conversion of carbon dioxide. *Adv. Energy Mater.* **4**, 1301873 (2014).
39. Sumida, K. et al. Carbon dioxide capture in metal–organic frameworks. *Chem. Rev.* **112**, 724–781 (2012).
40. Boyd, P. G., Lee, Y. & Smit, B. Computational development of the nanoporous materials genome. *Nat. Rev. Mater.* **2**, 17037 (2017).
41. Nugent, P. et al. Porous materials with optimal adsorption thermodynamics and kinetics for CO_2 separation. *Nature* **495**, 80–84 (2013).
42. Anglin, E. J., Cheng, L., Freeman, W. R. & Sailor, M. J. Porous silicon in drug delivery devices and materials. *Adv. Drug Deliv. Rev.* **60**, 1266–1277 (2008).
43. Horcajada, P. et al. Porous metal–organic-framework nanoscale carriers as a potential platform for drug delivery and imaging. *Nature Mater* **9**, 172–178 (2010).
44. Bishop, C. M. Pattern Recognition and Machine Learning. (Springer, New York), (2006).
45. Thommes, M. et al. Physisorption of gases, with special reference to the evaluation of surface area and pore size distribution (IUPAC Technical Report): Pure Appl. *Chem* **87**, 1051–1069 (2015).
46. Cychosz, K. A. & Thommes, M. Progress in the physisorption characterization of nanoporous gas storage materials. *Engineering* **4**, 559–566 (2018).
47. Christensen, R. et al. Interatomic potential parameterization using particle swarm optimization: case study of glassy silica. *J. Chem. Phys* **154**, 134505 (2021).
48. Wang, D., Tan, D. & Liu, L. Particle swarm optimization algorithm: an overview. *Soft Comput* **22**, 387–408 (2018).
49. Perez, R. E. & Behdinan, K. Particle swarm approach for structural design optimization. *Comput. Struct.* **85**, 1579–1588 (2007).

ACKNOWLEDGEMENTS

H.L. acknowledges funding from the Fundamental Research Funds for the Central Universities under the Grant No. YJ202271. M.B. acknowledges the National Science Foundation under the Grant No. DMREF-1922167. TPU computing time was provided by a grant allocation from Google's TensorFlow Research Cloud (TFRC) program.

AUTHOR CONTRIBUTIONS

Conceptualization: M.B. Methodology, Writing (review and editing): H.L. and M.B. Investigation: H.L., Y.L., K.L. and Z.Z. Visualization H.L. Supervision: H.L., S.S., E.C., P.G. and M.B. Writing (original draft): H.L.

COMPETING INTERESTS

The authors declare no competing interests.

ADDITIONAL INFORMATION

Correspondence and requests for materials should be addressed to Han Liu or Mathieu Bauchy.

Reprints and permission information is available at http://www.nature.com/reprints

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.