Supplementary Information for

A Generative Model for Inverse Design of Metasurfaces

Zhaocheng Liu[1], Dayu Zhu[1], Sean P. Rodrigues[1,2], Kyu-Tae Lee[1], and Wenshan Cai[1,2]*

[1] School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332

[2] School of Materials Science and Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332

* Correspondence should be addressed to W.C. (wcai@gatech.edu)

Contents:

1. Detailed information of the network architecture

As we are processing the binary image data, we utilized the convolutional neural network (CNN) for all three parts of the network – the simulator, the critic, and the generator. Figure S1 presents all the structures within the network. To better understand the image, take the simulator as an example in Fig. S1(a). The input image is represented as a three-dimensional tensor in a tuple (depth, height, width). In our case, the patterns are 64 by 64-pixel binary (1-channel) images, so the input tensor is in the shape of a (1, 64, 64). We picture the tensor as blocks in **Figure** S1 and the size of all the tensors throughout the network are marked at the bottom of each block. Different layers of the CNN are connected by convolutional operations defined by kernels with randomly initialized weights. In Fig. S1(a), kernels are represented as a darker green, and the convolutional operations are illustrated as the gray lines: the kernel multiplies the values of the tensor in the region of that kernel and then sums them together as a new value in tensors in the next layer. At the end of the CNN, we attach fully connected networks (dimensions shown at the bottom) to approximate all four transmittances T_{ij}'s of the input structure patterns. The critic in Fig. S1(b) shares a similar structure, but at the end of the CNN we did not attach an additional network as indicated in the deep convolutional generative adversarial network (DCGAN) [1].

The network structure of the generator as shown in Fig. S1(c) is inverse compared to the simulator and the critic, i.e. the operations between each layer are processes of generating new information rather than extracting information. Mathematically, this operation in matrix representation is identical to the transpose of the convolutional operation in the CNN, so this operation is named as the transpose convolutional operation. The network structure of the generator is also named as the transpose convolutional neural network (TCNN). In our case, we take the spectrum represented as a vector and an additional 50-dimensional noise vector into the

generator. After several transpose convolutional layers, the final tensor shares the same dimension (1, 64, 64) as the images that represent the unit cell of the metasurface.

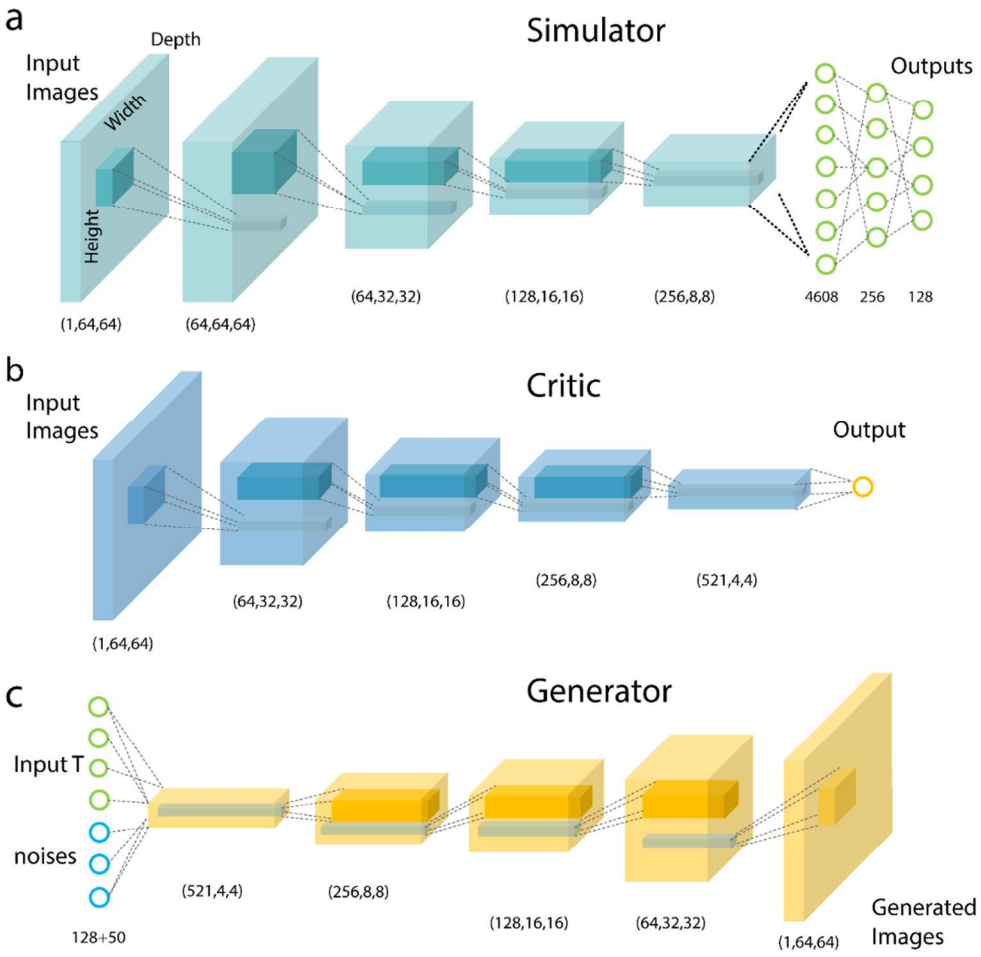

Figure S1 | The network architectures of the simulator, the critic, and the generator. (a) Illustration of the simulator network. Each shallow-colored block represents a tensor in the corresponding layer of the network. The size of the tensor in each layer is represented in a tuple (depth, height, width) under the block. The dark-colored block inside the tensor represents kernels of the convolutional operations, and the corresponding tensor after the convolutional operations are indicated as the gray cube in the tensor in the next layer. At the end of the CNN, a fully connected network with the number of neurons marked under each layer is attached for the approximation of the transmittances. (b) and (c) The detailed network structures of the critic and the generator with the same notations as in (a).

2. Algorithms and hyperparameters used in the training

In the following discussion, we follow the same definition for all parameters defined in the main article, i.e. we denote the simulator, the generator, and the critic as *S*, *G* and *D* respectively; we denote the image data representing the unit cell of the structures in the dataset as *s*, patterns produced by the generator as *s'*, and denote the corresponding FEM simulated transmittances of *s* and *s'* as *T* and *T'* respectively. We further define the geometric data as *x*, and the noise input to the generator as *z*. Then the following equations hold:

$$\hat{T} = S(s), \tag{1}$$

$$s' = G(T, z). \tag{2}$$

where $\hat{T}$ is the estimated transmittances by the simulator.

a. The simulator *S*.

The training of the simulator follows the common method for training a CNN. The loss of the network is defined as:

$$L_S = \frac{1}{N}\left\| T - \hat{T} \right\|^2, \tag{3}$$

where *N* is the number of the entries of *T*. Equation (3) defines the distance of the approximated transmittances and the input transmittances. During the training, as the total number of data is limited (6,500 in total), we augmented the dataset in the following ways so that it maximumly improves the performance of the simulator. First, we randomize the pixel values in the way that the regions with and without gold are represented with uniformly distributed random numbers in (0, 1] and [-1, 0] respectively. Second, we randomly shifted the patterns in the images with the

periodic boundary conditions in both *x*- and *y*-directions. This augmentation does not require any change in the FEM simulated spectra *T*. Third, we randomly rotate the patterns in the images with 90, 180 or 270 degrees. When the patterns are rotated 90 or 270 degrees, it is necessary to exchange the T_{xx} , T_{xy} to T_{yy} , T_{yx} respectively in the loss of the simulator. Note that it is forbidden to flip the patterns, in which the case the T_{xy} and T_{yx} of some asymmetric structures cannot be exchanged. Figure S2(a) presents the variation of the loss during the training of the simulator. Some hyperparameters of the training are listed in the **Table** S1.

b. The generator *G* and the critic *D*

The generator *G* and the critic *D* themselves construct a generative adversarial network (GAN). The weights of *G* and *D* should be updated simultaneously. In our implementation, we utilized the training method proposed in the Wasserstein GAN [2], in which the loss of the critic is defined as:

$$L_D = \frac{1}{m}\sum_{i=1}^{m} D(x^i) - \frac{1}{m}\sum_{i=1}^{m} D(G(T^i, z^i)) \tag{4}$$

where *m* is the batch size, and the superscript *i* indicates the *i*th data in the batch. The loss is to measure the distance of the two distributions formed by the geometric dataset *x* and the generated data $s' = G(T, z)$. Informally, this loss defines the similarity of the patterns between the geometric data and the generated patterns. We also implemented the gradient penalty introduced in Ref. 3 to improve the training performance.

The parameters of the generator are updated through back propagation subject to the losses from both the simulator and the critic:

$$L_G = L_S + \lambda L_D \tag{5}$$

where λ is the parameter to balance the weight of L_S and L_D. We choose $\lambda = 0.0001$ for all the experiments. The weights of the generator and the critic are updated in an alternating order. Since the generator accepts feedback from both the simulator and the critic, it may not always result in ideal patterns as needed if there are contradictions happening in *S* and *D*. The general dynamics of the loss of the critic L_D and the simulator L_S are shown in the Fig. S2(b). After some iterations, the losses of the two networks tend to present oscillations (equilibrium). We define the greatest distance of the two losses as *e* at this equilibrium stage. Since L_D defines the similarity between the generated patterns and the geometric data, and L_S defines the accuracy of the spectra of the generated patterns, we can regard *e* as the mismatch between the patterns *s* and the geometric data *x*. Note that as the simulator is not perfect in our experiments, sometimes it leads to the wrong approximated spectra. When the retrieved pattern has a topology too distinct compared to the training data of the simulator, the resulting structure is inaccurate. An example of this can be seen in the second column of Figure 2(c), which shows a densely distributed structure composed of small dots (similar to random noise). Therefore, satisfactory patterns are achieved in the training when L_S and L_D are both reasonably small. This is the crucial reason for having to document the possible patterns throughout the whole training process.

In different periods during the equilibrium, it is common to see various topologies occur during the training. This occurs when *e* is large causing discrepancies between the generated patterns and the geometric data. In our experiments, we observe that *e* always nears zero when the patterns of the input spectra do exist. This can also occur when the input spectra do not

commonly appear in physics, which can lead to an extraordinarily large *e*, and the curve of the two losses will oscillate violently.

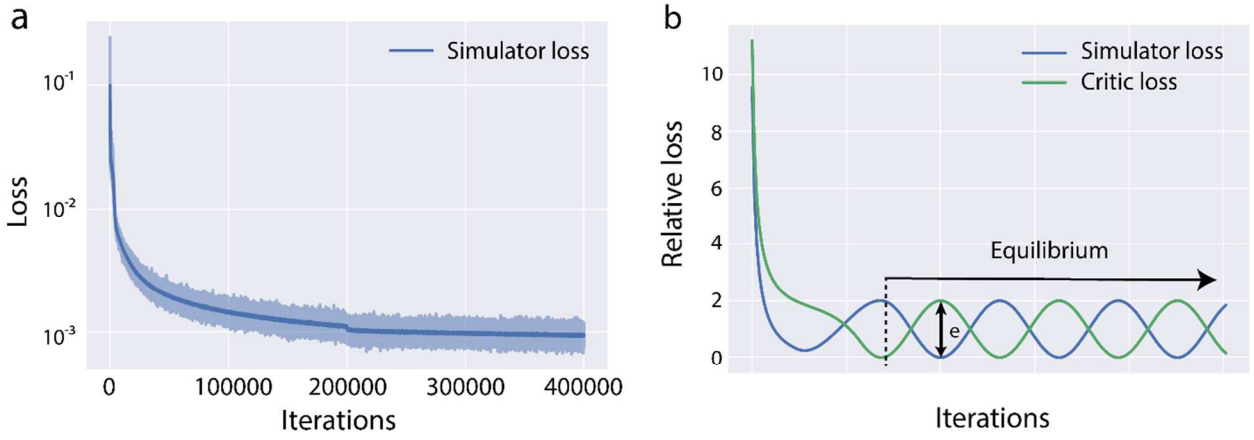

Figure S2 | Variation of losses during the training of the simulator and the generator (a) The loss of the simulator during the pre-training of the simulator, where the lighter-colored line represents the actual loss change and the dark line is the moving average of the loss. The break point around iterations of 200,000 is caused by the change of the learning rate. (b) The general loss variation of the simulator and the critic during the training of the generator. The maximum distance *e* of the simulator loss L_S and critic loss L_D in equilibrium indicates the mismatch of the geometric data and the actual patterns given input spectra. Valid patterns are documented in this equilibrium while both L_S and L_D are small.

	Simulator *S*	Critic *D*/ Generator *G*
Learning rate	from 5×10^{-5} to 1×10^{-5}	1×10^{-4}
Batch size	128	16
Optimizer	Adam $\beta_1 = 0.5, \beta_2 = 0.999$	Adam $\beta_1 = 0.5, \beta_2 = 0.999$
Nonlinear activations	Leaky ReLU $\alpha = 0.2$	Leaky ReLU $\alpha = 0.2$
Batch Normalization	Momentum = 0.1, $\varepsilon = 1\times 10^{-5}$	None

Table S1 | Hyperparameters used in the training of the simulator and the generator/critic.

3. Post processing of the image

The images of the patterns contain random-valued pixels representing the regions with and without gold. Hence, we need to post process the image to ensure it is binary and smooth for latter finite element method (FEM) simulation. Figure S3 illustrates the processing steps we adopted in our experiments. The leftmost image shows a pattern of a cross produced by the generator, where the darker region represents the gold structure. First, we transform the image into a binary image as shown in the middle image in Fig. S3. There are some isolated pixels in both the gold void regions. For the simulator, small perturbations like these do not affect the results of the approximation much, so we can remove those isolated pixels. To achieve this, for each pixel i in the image, we count the black and white pixels in its nearest neighbor and flip i if the pixels with opposite color count more. The final processed pattern is shown as the rightmost image in Fig. S3.

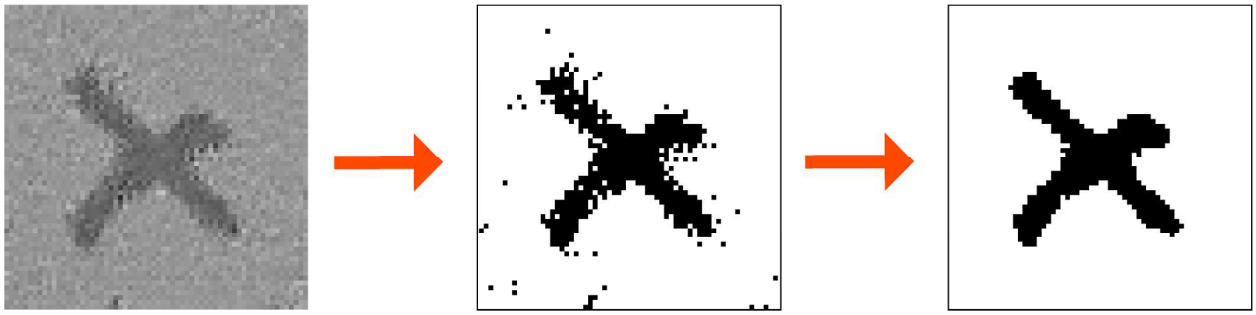

Figure S3 | Processing of the images from the generator. The left image is an example of the generated pattern from the network. It is transformed to a binary image (middle) and then smoothed to a valid pattern with as little deformation as possible (right).

4. Performance on partial input spectra

The input to the generator for the experiments in the main article are all four transmission coefficients, T_{ij}, where i, j can be x or y, which indicate the direction of the incident polarization and the detection polarization. In practice, we may only want to search the patterns that satisfy some components of the transmittance T_{ij} for a certain frequency range. This task is easier because more corresponding patterns of the input transmittances exist when the generator receives less restricted input spectra. Figure S4 shows two examples of the experiments with only T_{xx} or T_{xy} input to the generator. The experiment is carried out such that the critic only receives 1,000 arcs from the geometric data set. In Fig. S4(a), the blue and red lines are the FEM simulated T_{xx} of the test arc pattern *s* and searched pattern *s'*, respectively. The pattern of *s* (yellow) and the pattern of *s'* (black) are also presented on the right of Fig. S4(a). *s'* is topologically different as *s*, while the T_{xx} of *s'* only presents a small red shift compared to the T_{xx} of *s*. Fig. S4(b) shows another example with T_{xy} as the input to the generator. At this time a searched pattern *s'* is the same as the test pattern *s,* as indicated in the right part of the Fig. S4(b). All the examples above demonstrate that our model can be used as a general method to search patterns given any transmittance component and frequency range.

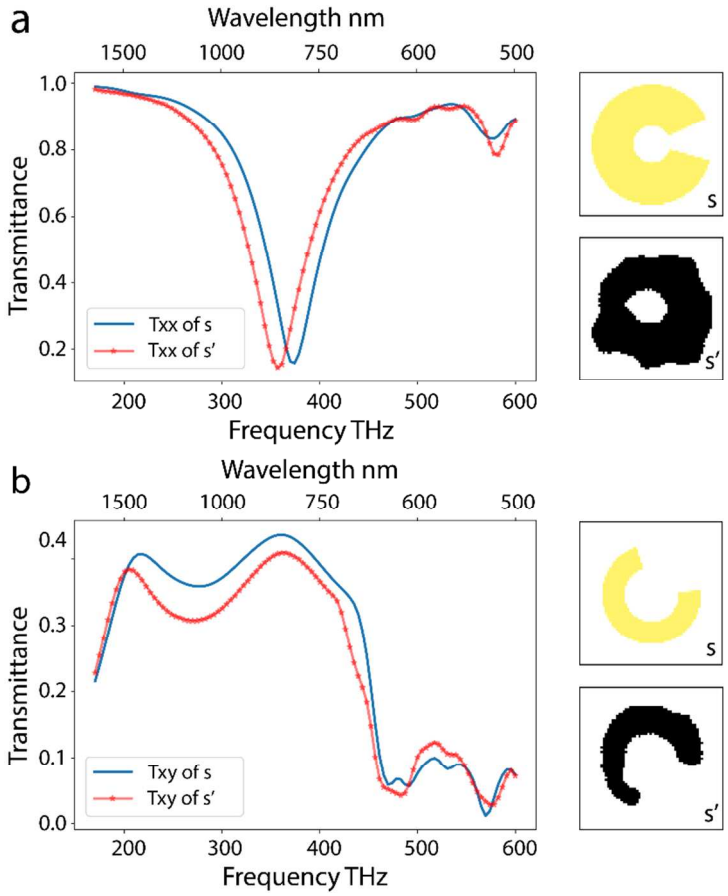

Figure S4 | Examples of the generated patterns with partial input spectra. (a) The blue line is the test T_{xx} input to the generator and the red line with dots is the FEM simulated T_{xx} of the generated pattern *s'*. The unit cell of the test pattern *s* and the generated pattern *s'* are also shown in the yellow and black images on the right respectively. (b) The comparison of an example with the same settings as in (a), but the input to the generator is T_{xy} of a different pattern *s*, the unit cell of which is shown in the image on the right.

5. Additional samples of results of the experiments in the main article

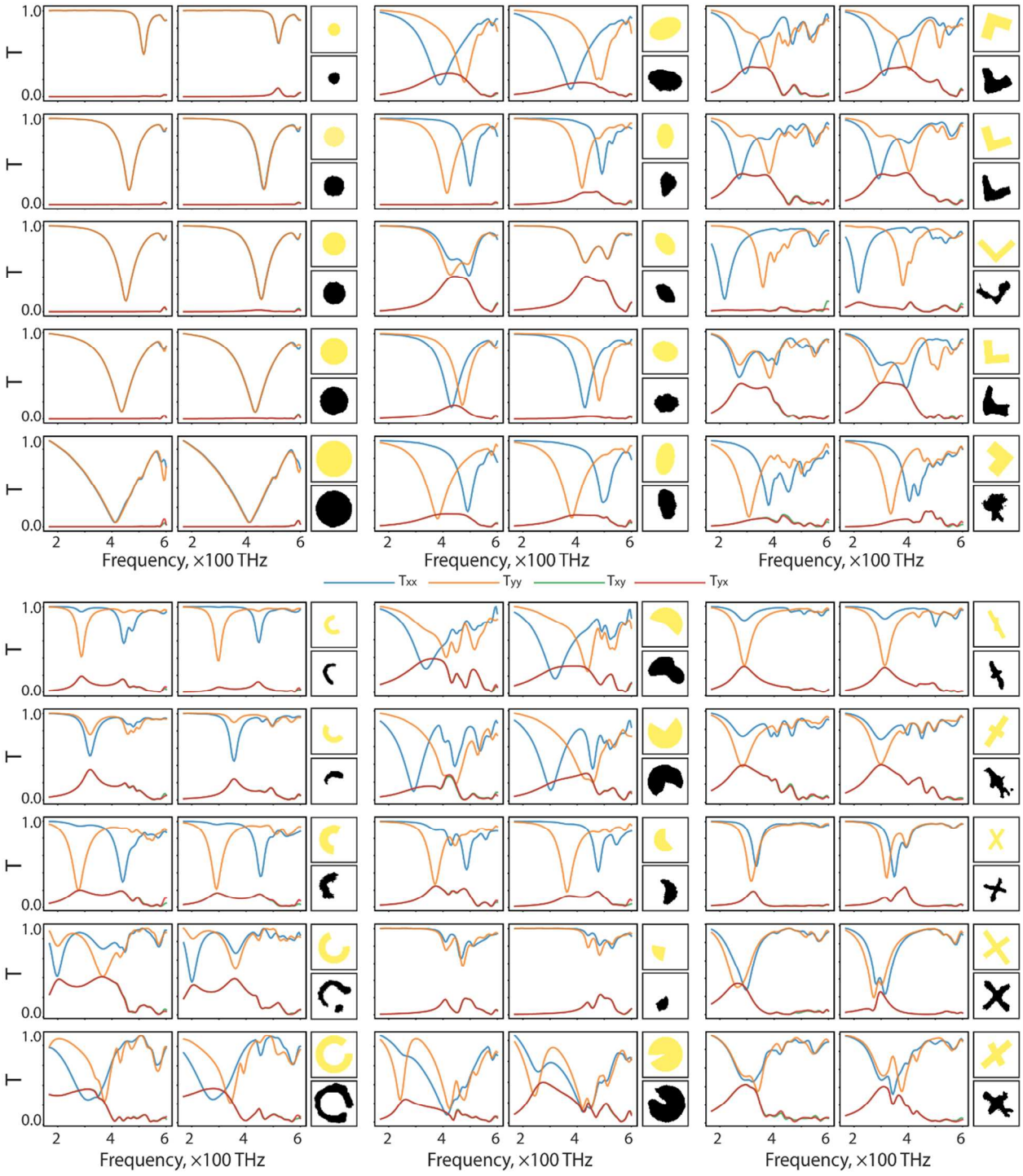

Figure S5 | Additional patterns generated from the network with comparisons to the test patterns. In each unit panel, the left and right figures are the FEM simulated transmittances of the test and generated patterns respectively. The corresponding unit cell of the test (yellow) and

generated (black) patterns are also shown on the right. All the samples shown above are derived when a single class of geometric data is fed into the critic.

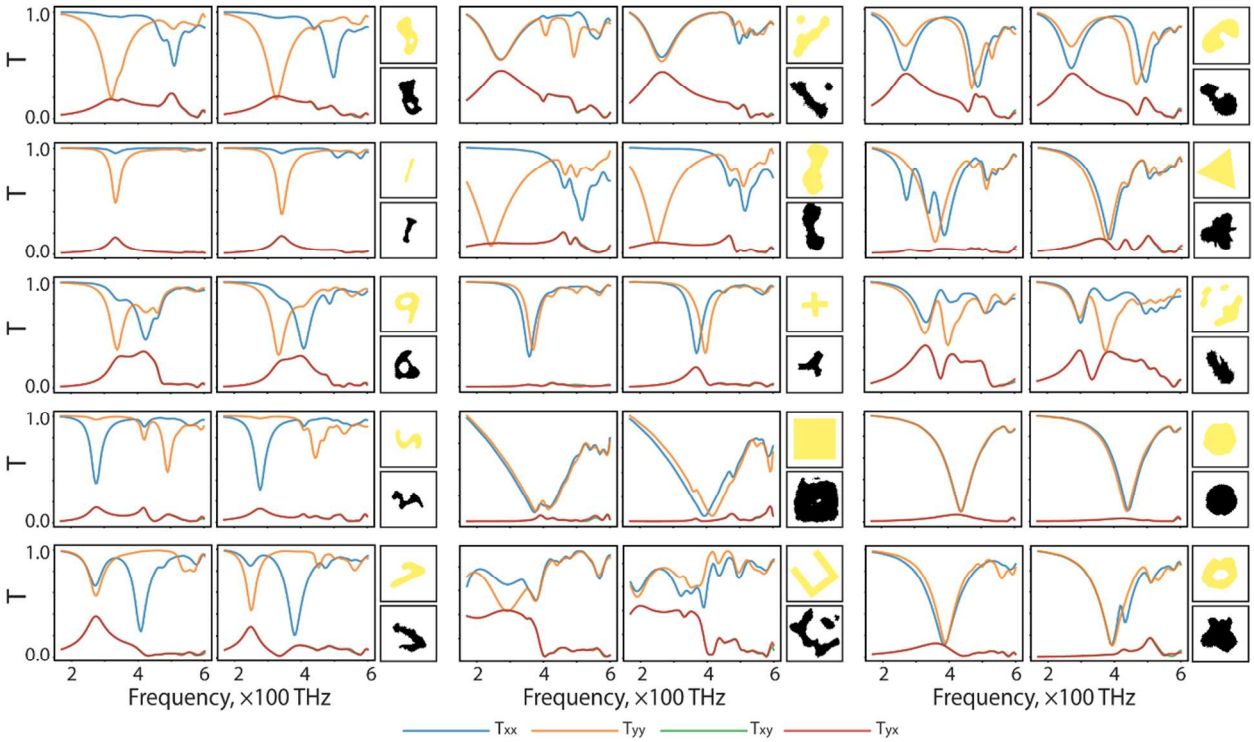

Figure S6 | More examples. Additional samples of results from the experiments described in the main article as show in Fig. S5. The figures in the first column are the samples derived from the network trained with the geometric data of the modified MNIST dataset, and the figures in the last two columns are the samples generated from the network trained with the entire mixed geometric data set.

6. Additional samples of results with manually defined input spectra.

In practice, the required spectra are determined by humans, and the existence of the required spectra is uncertain given the working parameters that are specified in the main text. Just as before, our network is able to seek the possible patterns such that the discrepancy of the spectra of the generated patterns *T'* and the input spectra *T* are minimized assuming the simulator is sufficiently robust. This can be achieved because the loss of the generator includes L_S , the Euclidean distance of the input spectra *T,* and the approximated spectra from the simulator $\hat{T}$. During the training, the generator probabilistically produces patterns with a minimum distance between *T* to $\hat{T}$. If the simulator is robust enough such that it is able to approximate any spectra of input patterns accurately, we can regard the approximated spectra $\hat{T}$ as *T'*. Therefore, training the generator will always result in the patterns with minimum error of the required spectra.

Figure S7 displays four examples of searched patterns when T_{xx} and T_{yy} are randomly generated Gaussian-like transmittances. The pattern searched in the Fig. S7(a) perfectly match the input spectra. However, in Fig. S7(b) and (c), the transmittances of the searched patterns are lower than the desired input, while the peaks are in the same frequency ranges. This reflects the fact that when no patterns exist for the input spectra, the generator will compromise to produce some other patterns while minimizing the loss of the simulator L_S . Fig. S7(d) is an example when the broadband property of T_{xx} cannot be satisfied. Here, to minimize L_S , the network generates a pattern with two peaks residing in the frequency range of the desired bandwidth of T_{xx} . As the geometric data does not contain the exact patterns that corresponds to the input spectra, the mismatch *e* discussed in section 2 occurs in the equilibrium. While the losses of the critic and the simulator alternate, the patterns searched vary along with the increasing iterations.

It is possible to achieve several patterns whose spectra *T'* have a similarly minimized distance to the input ones *T*.

In Figure S8, we show the experiments when the input spectra are manually defined filters. As expected, a single-layer metasurface cannot match the exact same spectra as the filter-like optical spectra prescribed at the input of the generator. In this case, the simulator still tries to restrict the generator to produce patterns with a minimized distance between *T* and *T'*. Fig. S8(a) shows an example when the desired *T* is a notch-filter with a narrow bandwidth. The network successfully searched out one pattern with a narrow *T*" at the exact desired frequency. Fig. S8(b) presents the example when the input patterns are broadband. The network cannot find the perfect absorber within the defined frequency range. However, it does evaluate for one with a broadband feature. Figure S8(c) and (d) are the two generated patterns in different stages of the training given the same broadband input spectrum *T*. To the knowledge of the network, it cannot find the patterns with *T'* perfectly matching *T*, but it produces patterns with either broadband (Fig. S8(c)) or high absorption (Fig. S8(d)).

The performance of the network can be systematically improved by training a simulator with increased capacity by approximating the spectra of the generated patterns. This can be achieved by training the simulator with much more data and adopting a more sophisticated network model. In particle, the shape of the input spectra also affects the performance of the network. For instance, the pattern of a Gaussian-like spectra, which resembles the real spectra of the metasurface, is much more easily found compared to a square-shaped spectrum. The designing of the input spectra is another interesting topic for further investigation.

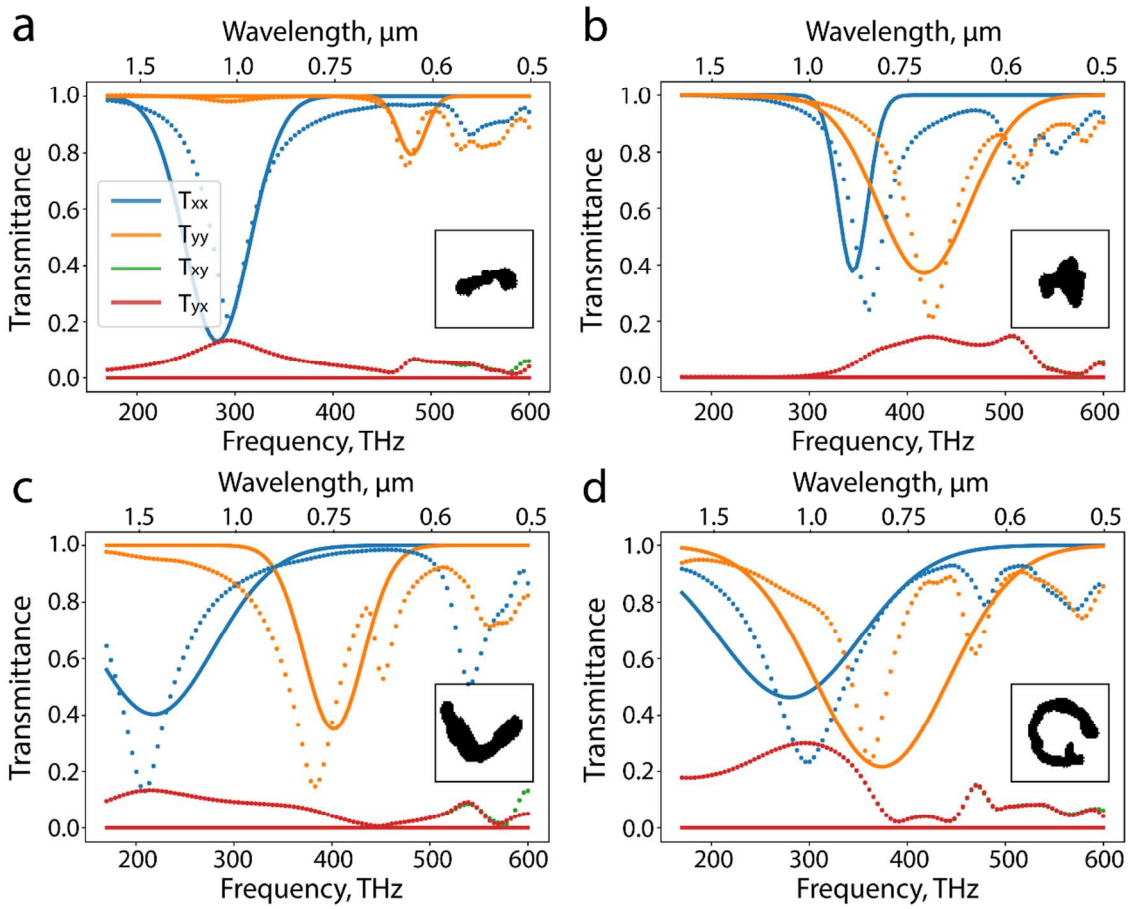

Figure S7 | Results of experiments with a Gaussian-like random input spectra. (a – d) Four examples of the generated patterns, when the input T_{xx} and T_{yy} are two spectra of Gaussian-like lineshape, and T_{xy} and T_{yx} are zero across the specified frequency range. Solid curves represent target spectra input to the network, and dashed lines represent FEM simulated results of the identified nanostructure shown in the lower right corner of each part. The data input to the critic are a mix of all classes of geometric data. When no pattern can be found with spectra that replicate the input ones, the generator will compromise and yield patterns with a penalty on the accuracy of the spectra T'.

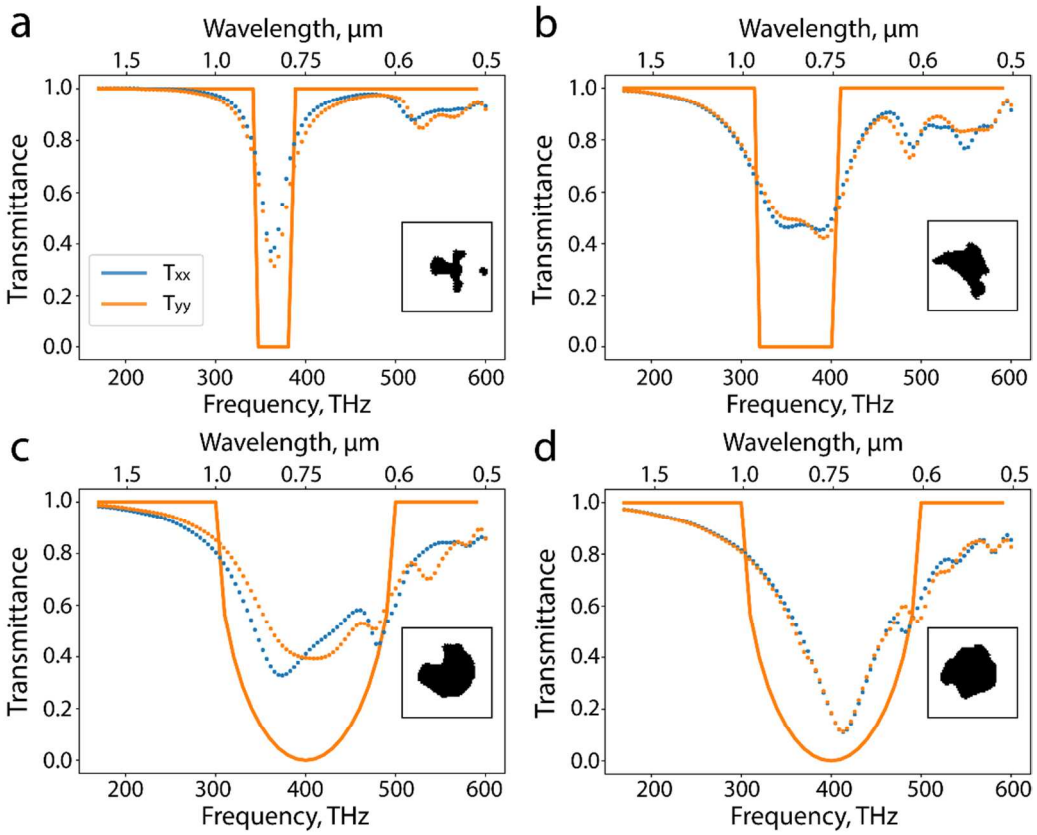

Figure S8 | Results of the experiment with manually defined input spectra. (a – d) Examples of on-demand designing. In this experiment, we only consider the T_{xx} and T_{yy}. The solid lines are required spectra, and the existence of the correspondent metasurface is almost impossible in the framework of a single-layer metasurface. The dashed lines are the spectra of the searched patterns. (a) and (b) are two notch filters. Even though the searched patterns can not meet the requirements defined as the input spectra, the network still searches two patterns with consistent peak frequency and bandwidth. (c) and (d) are results of two samples identified by the network while the input spectra are maintained the same. The broadband and the perfect absorption cannot be satisfied at the same time with the current trained simulator, the network still identifies two possibilities of the unit cell with either broadband or high absorption.

7. Detail information of the geometric dataset

We constructed a dataset with different classes of geometries as shown in the Figure S9. We selected part of them and simulated the transmittances of them using FEM for the training of the simulator. This part of the data is not included in the geometric dataset fed into the critic. For the MINIST dataset shown in Fig. S9(i), we thickened the stroke to meet the minimum requirements needed for fabrication. We also included failed samples generated from the network when the simulator is not well trained to maximize the variety of the dataset as in Fig. S9(l).

During the training of the GAN, the amount of data required depends on the complexity of the potential nanostructure. The minimal data required varies from 50 (for circle-shaped nanostructure as in Fig. S9(1)) to 5,000 (for arbitrarily-shaped nanostructures). However, the larger dataset will always result in higher fidelity. This is because augmented data expand the number of possible nanostructure shapes that can be chosen as the potential candidates. However, the increased dataset may incur slow convergence of the training process. Two potential approaches for reducing this requirement of prior knowledge include: 1) training a generator to replace the geometric data, and 2) incorporating an auxiliary network to identify the general class of geometry, given the desired spectra is provided. However, the detailed elaboration of these enhancements could be subject of a new study.

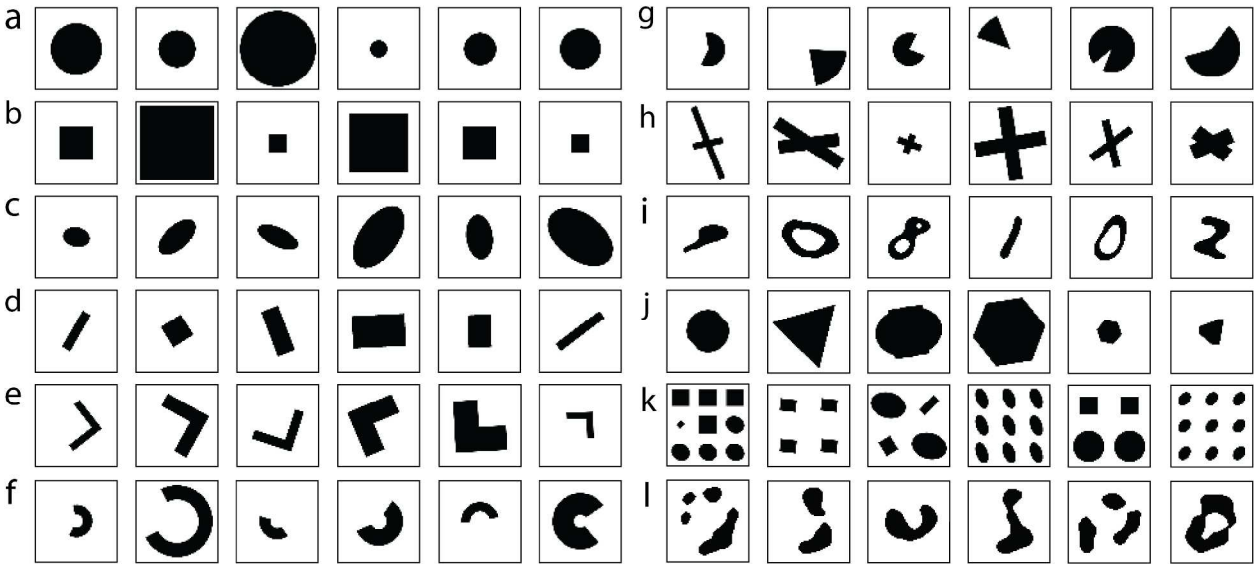

Figure S9 | Samples of data used in the training of the simulator and the generator. (a – h) Samples of the classes of circle, square, ellipse, rectangle, L-shape, arc, sector, and cross. (i) Sample data of the modified MNIST handwritten digital dataset. (j) Polygons and patterns generated by Boolean operations (union, intersection, complement) of the geometries from the class (a) to (h). (k) Arrayed patterns, in which the contained elements are randomly selected from the class (a) to (d). (l) Failed samples generated from the network while training the generator.

8. Mathematical definitions of the geometric, average and minimum accuracy

To evaluate the performance of our model, we defined three accuracies in the main article: the geometric accuracy, the average accuracy and the minimum accuracy. The detailed mathematical definitions are as follows:

Geometric accuracy:

$$a_{goem} = \frac{n_g}{n}, \tag{6}$$

where n is the total number of spectra input to the model and the n_g is the number of searched patterns that can be recognized in the same class of the geometric data fed into the critic.

Average accuracy:

$$a_{ave} = 1 - \frac{1}{f_{\max} - f_{\min}} \int_{f_{\min}}^{f_{\max}} (T - T')df, \tag{7}$$

where T is the input spectra, T' is the FEM simulated spectra of the generated patterns, and $f_{\min}$ / $f_{\max}$ is the maximum/minimum frequency of the input spectra.

Minimum accuracy:

$$a_{\min} = 1 - \max_f \left(T(f) - T'(f) \right). \tag{8}$$

The geometric accuracy is used to evaluate the similarity of the generated patterns compared with the geometric data. In practice, this accuracy is not of interest if the aim of the design is to find a pattern with a spectrum the same as the input one. For fabrication and other practical purposes, this accuracy is essential if we desire to eliminate unnecessary patterns such

as sharp corners and tiny dots. The average and the minimum accuracy defines the discrepancy of the input spectra T and the spectra of the generated patterns T'. Note that these two definitions do not imply the physical agreement of the input and generated spectra such as the consistency of the peak frequency and the bandwidth, thus the patterns cannot be regarded as valid only based on these two accuracies.

References

1. Radford, A.; Metz, L.; Chintala, S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. 2015, arXiv:1511.06434v2. arXiv.org e-Print archive. https://arxiv.org/abs/1511.06434 (accessed 19 Nov 2015).

2 Arjovsky, M.; Chintala, S.; Bottou, L. In *Wasserstein Generative Adversarial Networks*. International Conference on Machine Learning, pp. 214-223, Sydney, AU, Aug 6-11, 2017.

3 Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; Courville, A. C. In *Improved Training of Wasserstein GANs*, Advances in Neural Information Processing Systems, pp. 5769-5779, Long Beach, US, Dec 4-9, 2017.