Noise function	Sampled parameters	Range	Noise function	Sampled parameters	Range
cells 4	scale	$i[5, 50]$	grunge rust fine	base grunge contrast	$f[-0.3, 0.3]$
cells 1	scale	$i[10, 50]$		base warp intensity	$f[0.0, 256.0]$
voronoi	scale	$f[5.0, 15.0]$	grunge damas	distortion	$f[0.0, 1.0]$
	distortion intensity	$f[0.0, 1.0]$		divisions	$i[4, 16]$
	distortion scale multiplier	$f[1.0, 2.0]$		waves	$i[1, 3]$
microscope view	scale	$i[25, 64]$		details	$f[0.0, 0.75]$
	warp intensity	$f[0.0, 1.0]$		rotation random	$f[0.05, 1.0]$
bnw spots 1	scale	$i[1, 3]$	grunge map 002	N/A	N/A
liquid	scale	$i[10, 45]$	grunge map 005	N/A	N/A
	warp intensity	$f[0.1, 0.8]$	messy fibers 3	scale	$i[1, 3]$
grunge galvanic small	crispness	$f[0.0, 0.75]$	perlin	scale	$i[10, 50]$
	dirt	$f[0.0, 1.0]$	gaussian	scale	$i[10, 50]$
	micro distortion	$f[0.0, 0.6]$	clouds 1	scale	$i[1, 5]$
grunge leaky paint	leak intensity	$f[0.0, 1.0]$	clouds 2	scale	$i[1, 5]$
	leak scale	$i[1, 8]$	clouds 3	scale	$i[1, 3]$
	leak crispness	$f[0.0, 0.8]$			

Table 1. We enumerate the noise functions that are sampled from Adobe Substance 3D Designer, along with their parameters and accompanying ranges – integer and real ranges are denoted by i and f respectively.

	FID↓			FID↓	
Noise	**PSGAN**	**Ours**	**Noise**	**PSGAN**	**Ours**
cells 4	218.8	33.6	rust fine	88.5	12.3
cells 1	171.3	2.4	damas	56.1	71.0
voronoi	149.3	12.5	map 002	37.3	13.7
microscope	133.1	29.7	map 005	155.4	34.5
bnw spots 1	22.4	4.4	fibers	86.4	34.6
liquid	163.7	38.0	perlin	47.8	4.8
galvanic small	79.2	24.9	gaussian	45.2	1.7
leaky paint	155.1	44.3	clouds 1	20.4	1.5
clouds 3	38.4	2.9	clouds 2	110.8	9.0
Mean	99.2	20.9	**Median**	87.5	13.1

Table 2. We compare our FID scores for each noise type alongside PSGAN. The mean and median of all values are shown as well.

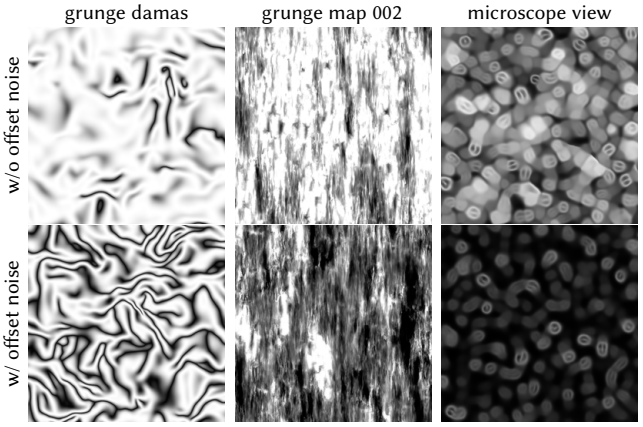

Fig. 16. Ablation of offset noise. Without offset noise, the network occasionally fails to synthesize noise images with extreme intensity distributions (i.e. intensely dark and bright images). We show representative examples above.

A EXPERIMENTAL AND IMPLEMENTATION DETAILS

A.1 Noise dataset details

We include a table enumerating all sampled noise types, the parameters that we sample, as well as the parameter ranges (see Table 1). In general, we choose to include any parameters that lead to noticeable changes in the resulting noise, with exceptions to parameters that simply act as color correction (i.e. `grunge map 005`'s *contrast* parameter). We note that, despite their names, some of these noises correspond to those that belong to graphics literature; for instance, `cells 4`, `cells 1`, and `voronoi` are variants of Worley noise [Worley 1996]. For more details about these noises, please refer to the Adobe Substance documentation [2023b].

The conditioning vector $\mathbf{f}_p$ contains an entry for each unique noise parameter – we treat identical parameter names as separate, with the exception of `scale`, which is treated as a single entry in the vector. Parameters are independently normalized to the range $[0, 1]$ before being passed to our conditioning MLP. Below we show many samples of our model's outputs with all listed noise parameters being sampled randomly (see Figs. 19 to 23).

A.2 PSGAN Baseline

We slightly modify the PSGAN [2017] architecture for the results shown in Figure 6. Since PSGAN is a purely unconditional generative model (i.e. it has no class or parameter conditioning beyond randomized latent code inputs), we endow the PSGAN model with SPADE conditioning blocks that are identical to the ones used throughout our method. We made the discriminator slightly larger to compensate for the added parameters in the generator. Finally, we use the WGAN loss [Arjovsky et al. 2017]. The remaining parts of the network and training are largely kept as-is from the open implementation. In total the generator has 30 million parameters.

A.3 Inverse Material Design Details

We expose three parameters to the MATch differentiable material graph optimizer; a soft-class vector $\mathbf{v}_c$, the parameter vector $\mathbf{f}_p$, and the diffusion Gaussian noise image $\mathbf{z}$. The soft-class vector represents a list of $[0, 1]$ values that are used to index into our class embeddings – this amounts to taking a convex combination over the class embeddings. During optimization, we apply the SoftMax function to $\mathbf{v}_c$ to ensure the values are valid. In order to encourage sparsity, we include an L1 regularization term on both $\mathbf{v}_c$ and $\mathbf{f}_p$, using a weight of 0.1 for both L1 terms in the final optimization objective. Additionally, we include a scheduled temperature parameter τ into the SoftMax operator by performing element-wise division of $\mathbf{v}_c$ by τ, $\text{softmax}(v, t) = \frac{e^{v_i/t}}{\sum_{j=1}^{K} e^{v_j/t}}$. The temperature is initialized to 0.25 and is updated at every optimization step via the update rule $\tau' \leftarrow \tau \cdot 0.97$. Finally we clamp τ to be at minimum 0.01. This scheduled temperature forces the optimization to "hone in" on a single class. We use a learning rate of 0.01 for all graphs. After $\tau = 0.01$ we perform a warm-restart of the optimizer and add noise to the exposed parameters to avoid local minima.

B TRAINING WITH ADDITIONAL NOISE TYPES

We train our model on two additional noise types that are commonly used in graphics literature: Phasor noise [2019] and Gabor noise [2009]. We utilize the same data sampling method as mentioned in Section 4. The parameters that we sampled for both noise types, as well as their ranges, are enumerated in Table 3. We use the released implementations for both noise types to accrue our training data. Examples of spatially-varying images using parameter interpolation and class interpolation for both Phasor and Gabor noise are shown in Figs. 17 and 18.

We note that our model exhibits an FID of 93.6 and 129.3 on Phasor and Gabor noise respectively, which is notably higher than the FID scores for other noise types. Upon inspection of our model's outputs, we see that some parameter configurations are not well captured by our training. For example, Gabor noise exhibits a significantly different visual appearance when its principal frequency and kernel width are very small; however, since this appearance is only captured by a narrow subset of the parameter space, the dataset thus contains much fewer of such samples. One drawback of diffusion models is that they model low-density regions of the data distribution less accurately (compared to higher-density regions) [Song 2021; Um et al. 2024], and hence our performance is worse in such situations. More careful data sampling methods may be needed when the desired noise distribution exhibits low-density modes; e.g. by sampling some regions of the parameter space more frequently.

C MODEL ARCHITECTURES

The U-Net model detailed in Section 4 has just ∼5.1 million parameters and is constructed using two downsampling blocks, a bottleneck block, and two upsampling blocks. Each block contains two ResNet sub-blocks. An additional ResNet sub-block is placed at the end of the network. We use channel dimensions of 32 and 64 for the outer

Noise function	Sampled parameters	Range
phasor noise	principal frequency, F	$f[8.0, 32.0]$
	num cells	$i[1, 12]$
	phasor density	$f[0.3, 0.5]$
	factor angle spread, θ	$f[0.0, 1.0]$
gabor noise	principal frequency, F	$f[0.02, 0.08]$
	kernel width, α	$f[0.01, 0.35]$
	kernel orientation, ω	$f[0.0, 2\pi]$

Table 3. Parameters and sampling ranges for additional noise types.

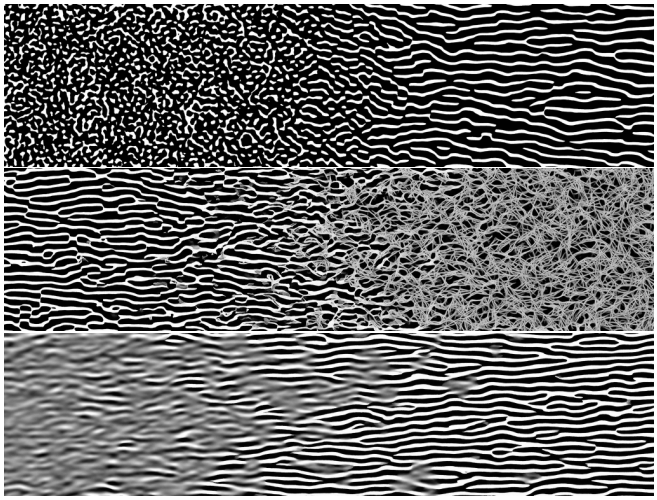

Fig. 17. Examples of our model's Phasor noise. From top to bottom: isotropic Phasor to anisotropic Phasor (parameter interpolation), anisotropic Phasor to Messy Fibers, and anisotropic Gabor to anisotropic Phasor (class interpolation).

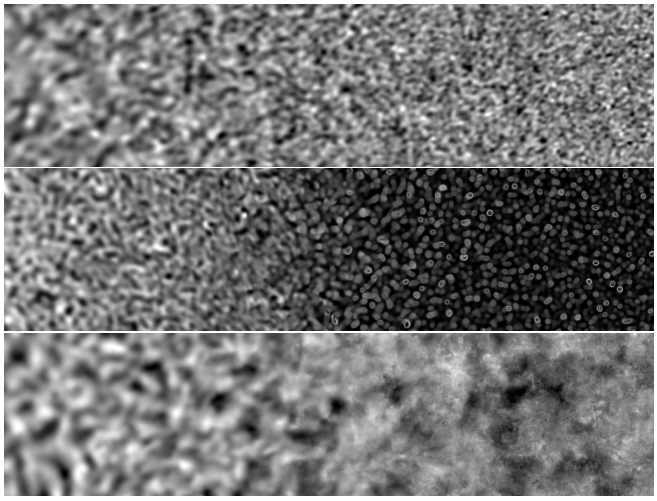

Fig. 18. Examples of our model's Gabor noise. From top to bottom: Gabor to Gabor (parameter interpolation), Gabor to Microscope View, and Gabor to Grunge Rust Fine (class interpolation).

Architecture	Block dimensions	Attention	Num. Parameters	**Mean FID↓**	**Median FID↓**	**Steps per second↑**
`Model-XS`	(32, 64, 128)	✗	5.1M	20.9	13.1	**79.5/s**
`Model-S`	(32, 64, 64, 128)	`heads` = 4, `dim` = 32	6.5M	14.0	10.8	28.6/s
`Model-M`	(64, 128, 128, 256)	`heads` = 4, `dim` = 32	22.5M	**10.3**	**10.4**	20.1/s

Table 4. Three model architectures with varying parameter counts and design choices are shown. The *Block dimensions* column shows the dimension counts for the downsampling blocks followed by the bottleneck block (the networks are symmetric; i.e. the upsampling blocks follow the same dimensions in reverse order). Our primary model (first row) is significantly faster than the other models, but compromises slightly on FID.

blocks, and 128 for the bottleneck. The ResNet sub-blocks are globally conditioned on the diffusion time t, and spatially conditioned (via SPADE) by 128-dim noise embeddings. We will refer to this model as `Model-XS` (extra small) below.

We train two additional model architectures to evaluate the potential performance of larger and more complex models. `Model-S` is identical to `Model-XS`, with the exception of added *linear attention* layers inside of each block [Shen et al. 2021], and the addition of an extra set of downsampling/upsampling blocks. Similarly, `Model-M` is identical to `Model-S`, with the exception of larger channel dimensions. In Table 4 we summarize these architectures as well as their FID scores and inference performance. The models were trained for the same number of optimization steps following the details in Section 4.

For our inverse material design application, we found that using `Model-XS` was most suitable due to the added memory cost of attention layers. For consistency, we used this model for all figures in the main text; however, in applications that do not require such light-weight networks, the larger models may be suitable.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009

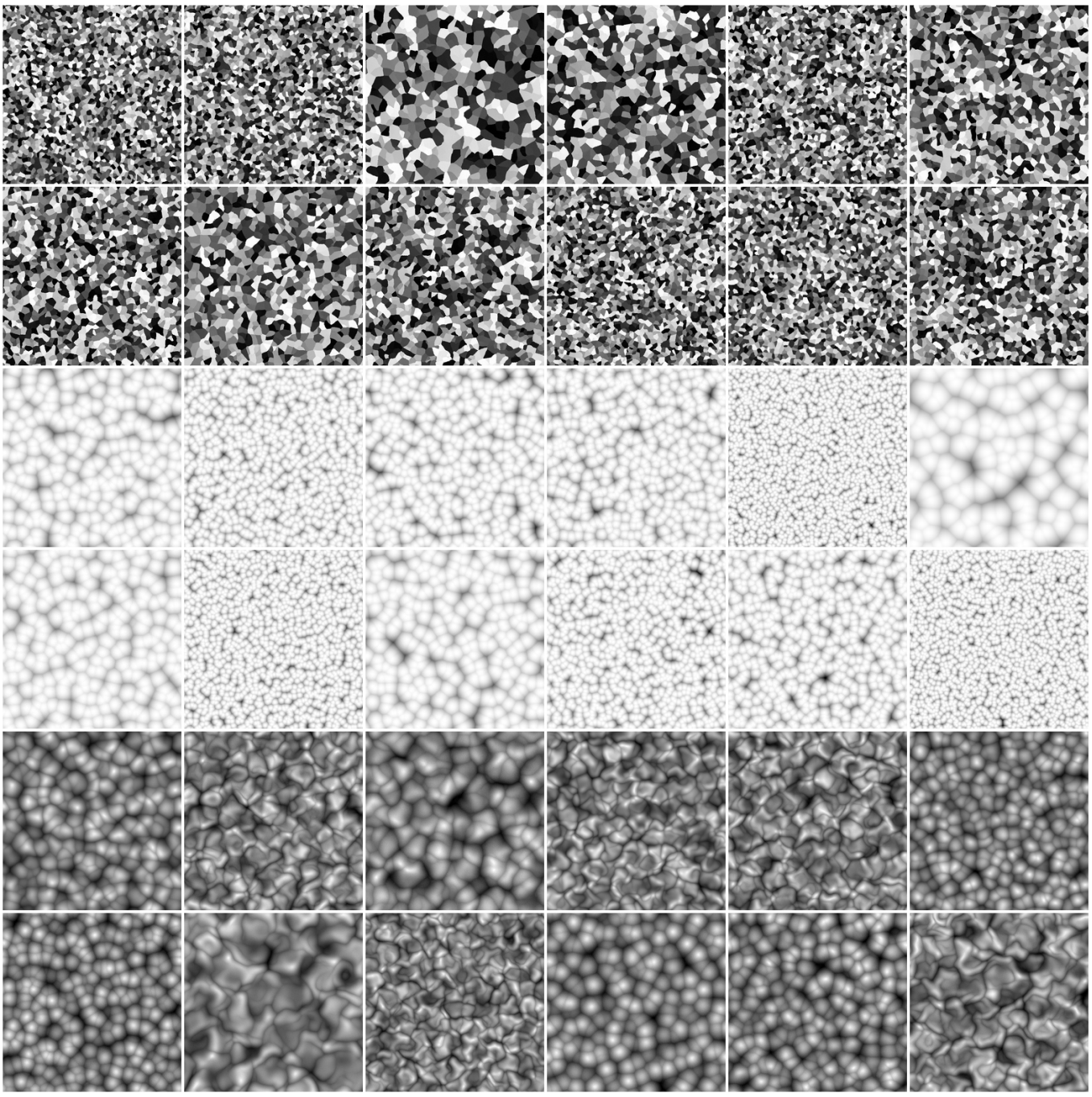

Fig. 19. Random samples of our model's `cells 4`, `cells 1`, and `voronoi` noises at 256×256 resolution.

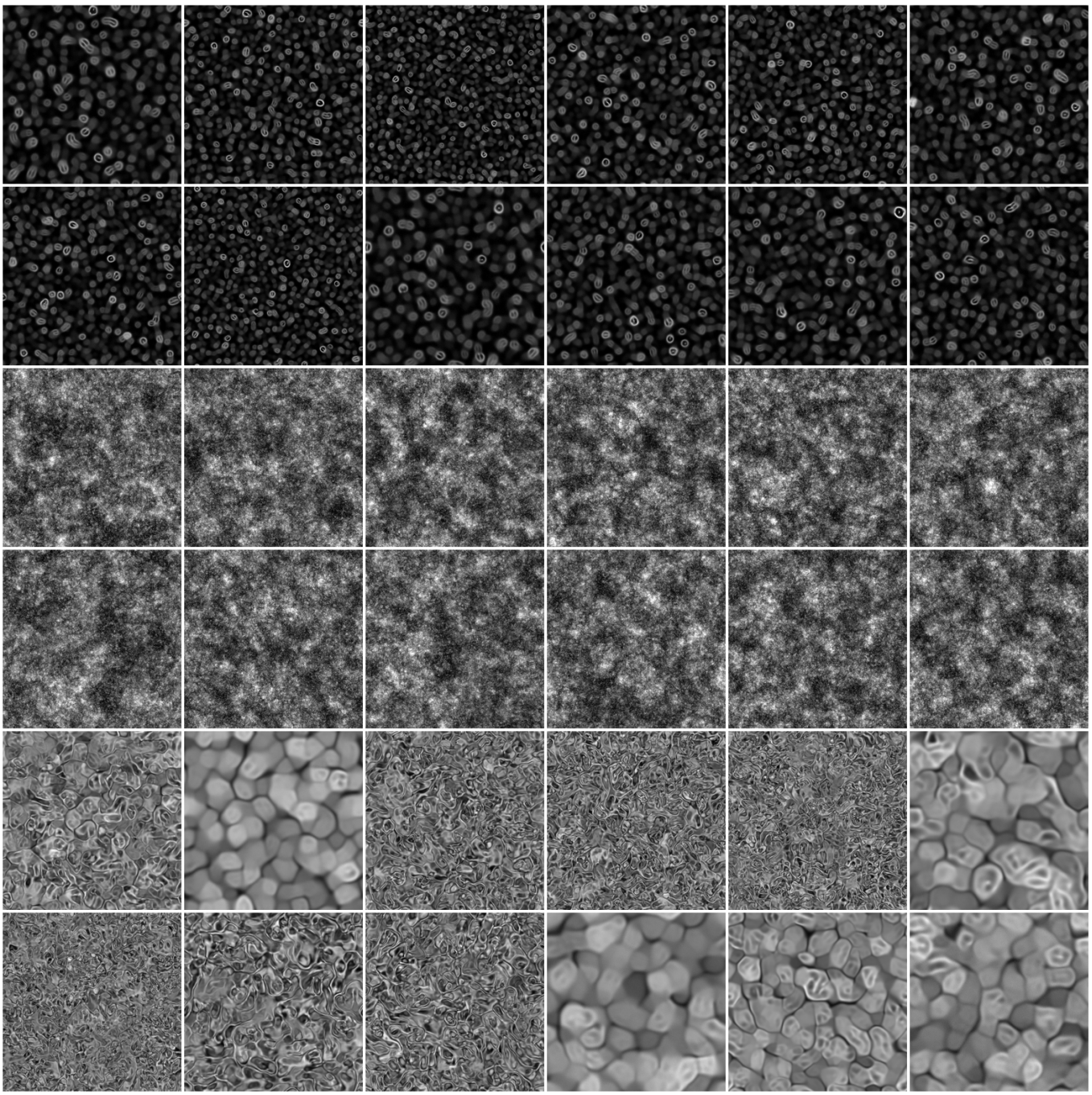

Fig. 20. Random samples of our model's `microscope view`, `bnw spots1`, and `liquid` noises at 256×256 resolution.

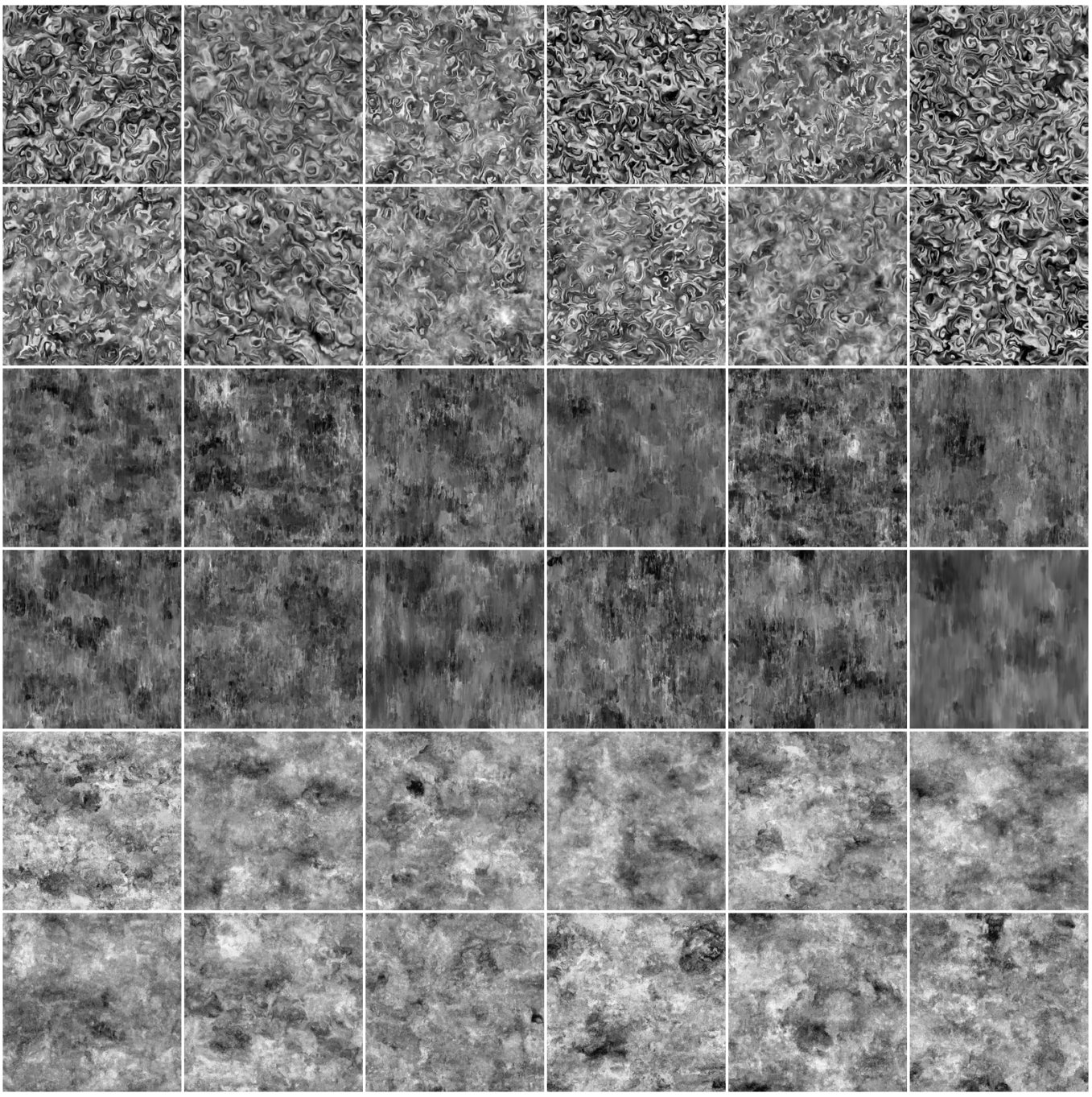

Fig. 21. Random samples of our model's grunge `galvanic small`, grunge `leaky paint`, and grunge `rust fine` noises at 256×256 resolution.

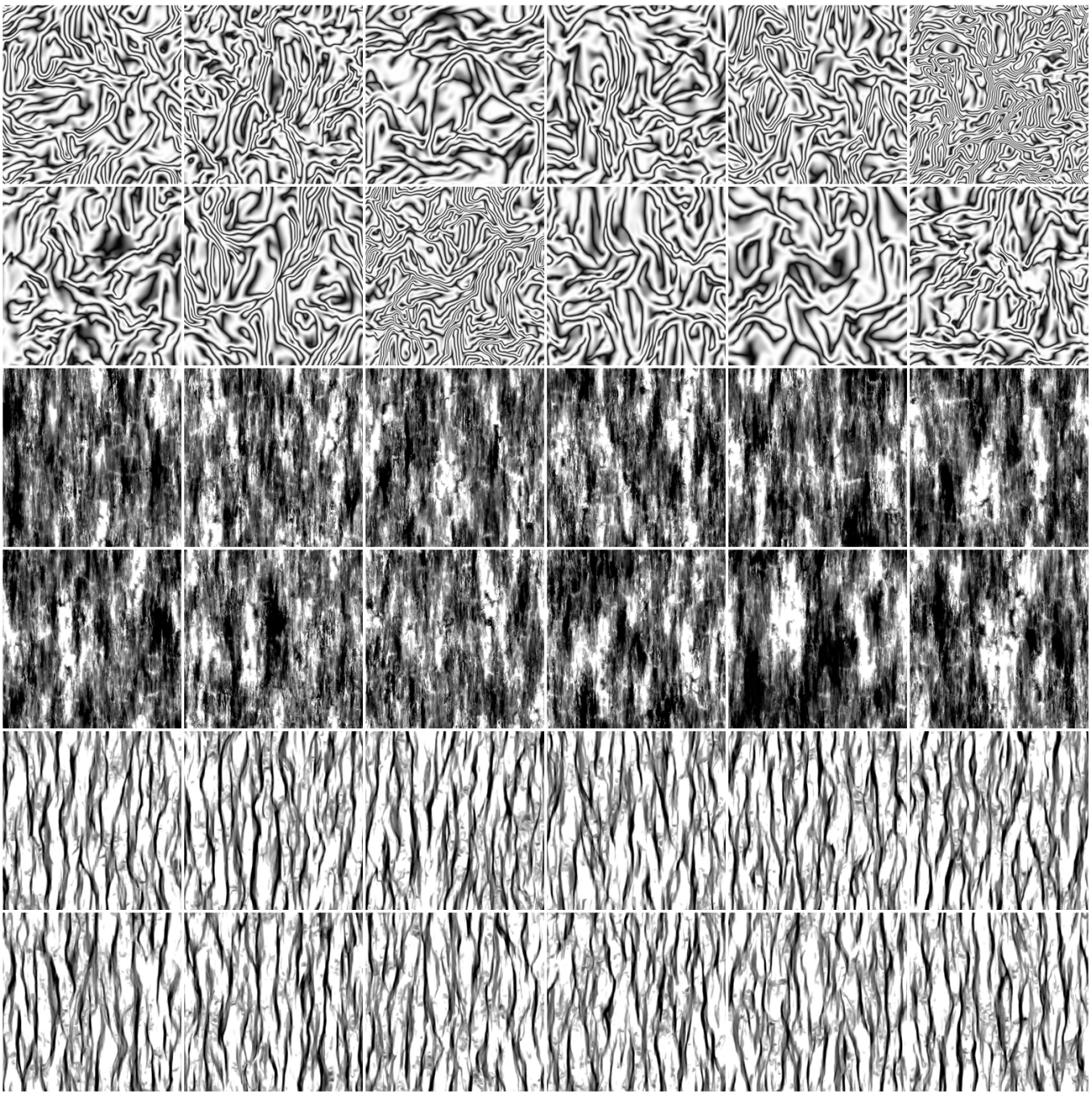

Fig. 22. Random samples of our model's `grunge damas`, `grunge map 002`, and `grunge map 005` noises at 256×256 resolution.

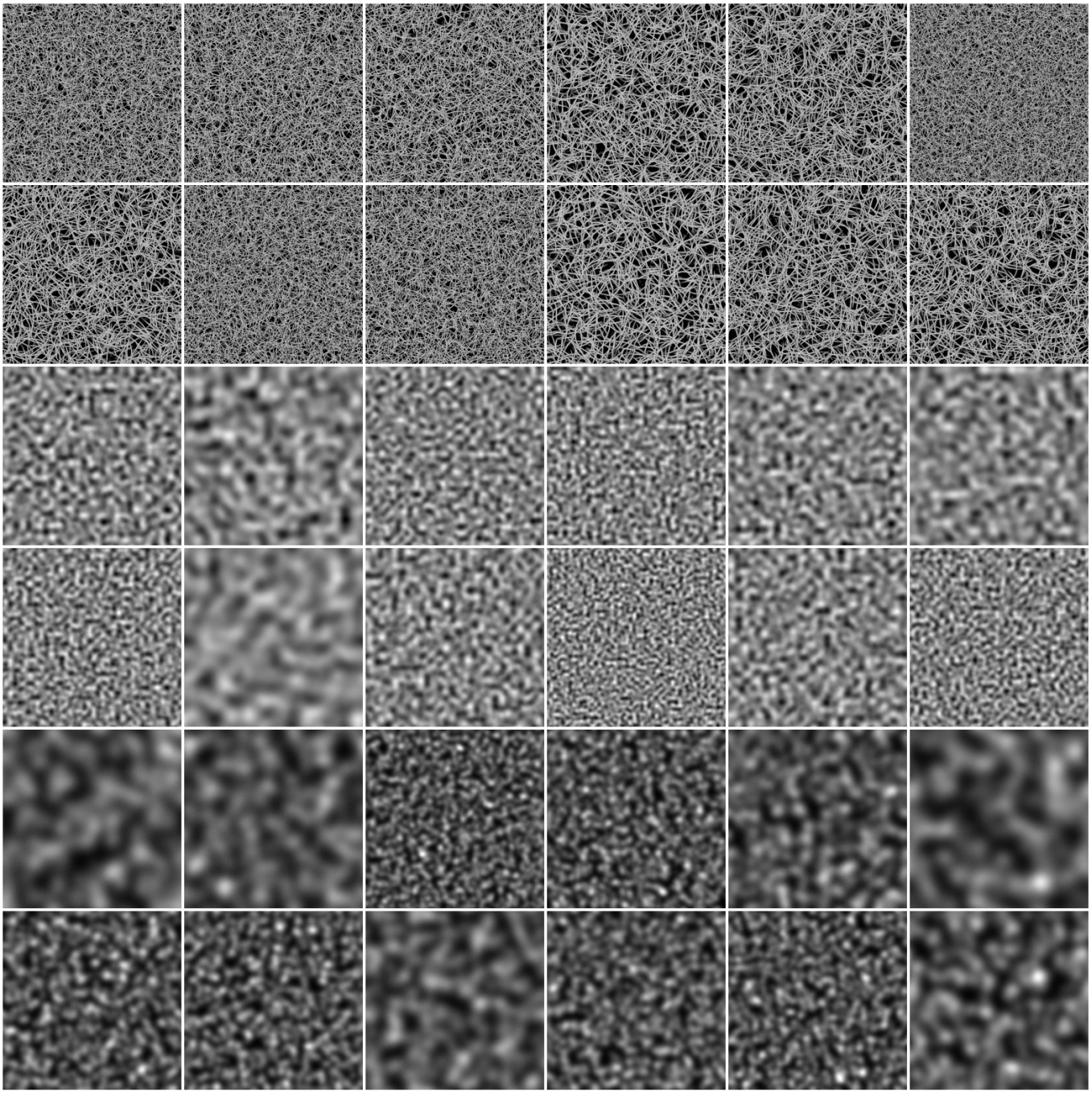

Fig. 23. Random samples of our model's `messy fibers 3`, `perlin`, and `gaussian` noises at 256×256 resolution.

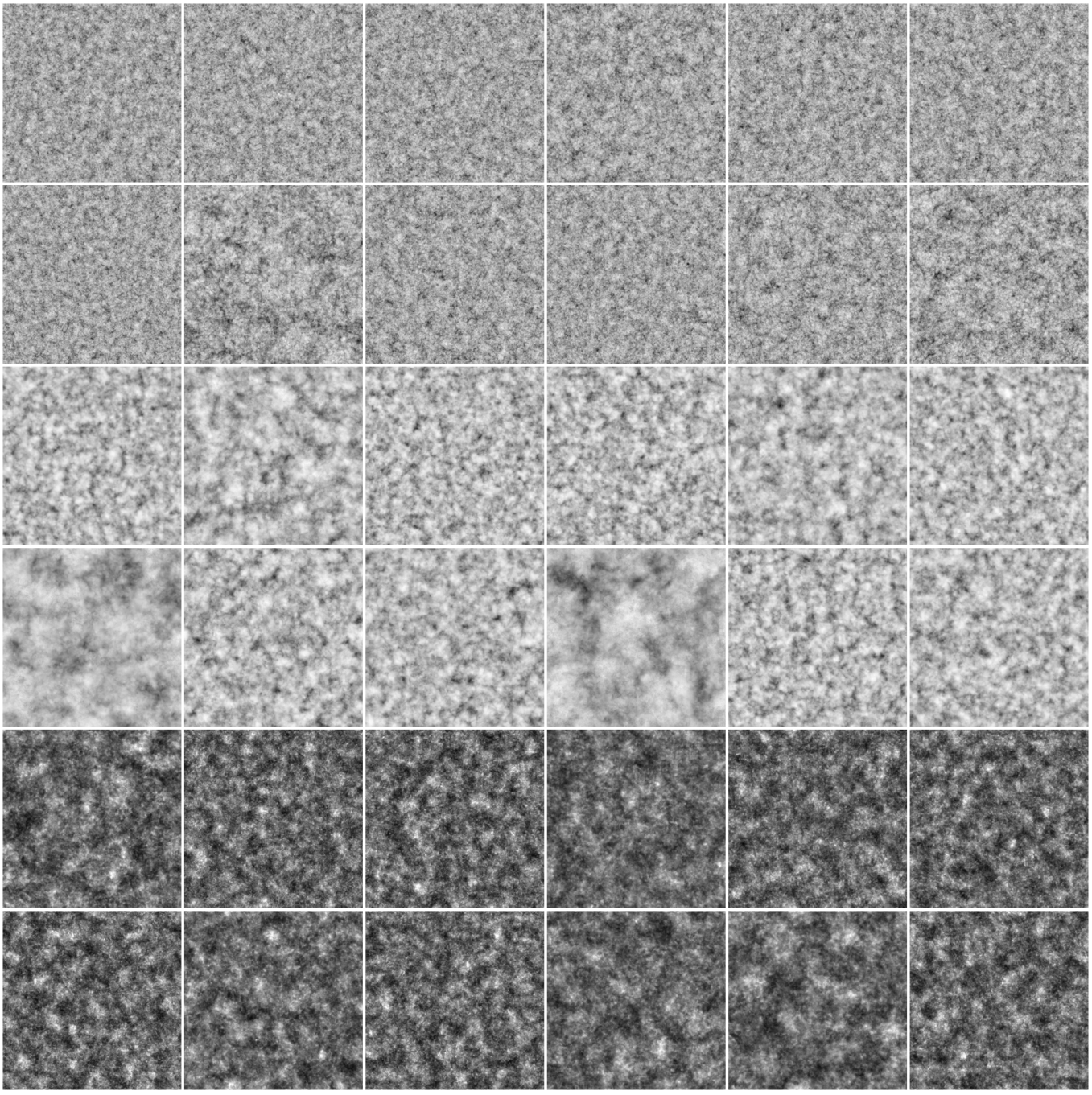

Fig. 24. Random samples of our model's `clouds 1`, `clouds 2`, and `clouds 3` noises at 256×256 resolution.