Example-based feature painting on textures

SUPPLEMENTARY MATERIAL

ANDREI-TIMOTEI ARDELEAN, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
TIM WEYRICH, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Table 2. Overview of each texture category used in our experiments. It includes the number of anomaly/feature types (K), the number of images without anomalies (Normal images), the number of images that contain prominent features, and the total. The first 5 textures are from MVTecAD, the other 10 are acquired by us.

Texture name	K	Normal images	Feature images	Total
Tile	5	33	84	117
Grid	5	21	57	78
Carpet	5	28	89	117
Wood	5	19	60	79
Leather	5	32	92	124
Pavement	2	0	9	9
Taboret	2	0	1	1
Shirt	1	0	1	1
Puzzle	1	0	1	1
Wall	2	0	5	5
Grass	2	0	15	20
Chair	1	0	3	3
Blueberries	2	3	12	15
Dots	3	1	6	7
SVBRDF	2	0	1	1

S1 Dataset examples

It is difficult to gauge the quality of the generated textures and the painted features without having a fair understanding of how the training data looks like. Therefore, we include in Fig. 13 and Fig. 14 a set of images from each texture class, presenting all existing features. Table 2 describes how many images and anomalies, or prominent feature types, are contained in each dataset.

S2 Additional experiment on latent noise uniformization

While our noise-uniformization technique is designed for textures, the concept readily extends to more generic image types, such as panoramas. We demonstrate this by incorporating our improvements in MultiDiffusion [Bar-Tal et al. 2023] to generate large-scale images that are more realistic in terms of internal consistency. The results of this experiment are presented in Fig. 15, comparing our results with a vanilla MultiDiffusion based on Stable Diffusion 2.0. To highlight the benefits of the introduced improvements, we use a stride of 32 for the sliding window. It can be easily seen that our results are more spatially consistent while still exhibiting a realistic amount of variation. This has the added advantage that seams are less noticeable, despite using a large stride. The seams are almost completely eliminated when we additionally use our randomized sliding windows scheme. That is, instead of using a fixed stride, we offset each window by a random amount both horizontally and vertically. We limit the offset to be less than half of the stride to ensure full coverage of the latent map. The seams seen in the first row of each group could also be resolved by running MultiDiffusion with a stride of 1; however, this would incur a significantly longer running time (18 minutes per image). Moreover, as seen in Fig. 16, the high-level inconsistencies remain even in this case.

We want to stress that our latent noise uniformization is complementary to the strategy used to generate large textures. For example, a competitive approach to the noise averaging employed by MultiDiffusion is the noise rolling algorithm introduced in ControlMat [Vecchio et al. 2024]. In Fig. 16, we combine our noise uniformization with the noise rolling method for generating arbitrarily large images and show that the results are significantly improved. Our uniformization technique is most useful when there is a high diversity of different images that would fit a given prompt, as it makes it difficult to reconcile adjacent patches (e.g. beach landscape, Fig 16). The noise rolling algorithm has the advantage that each pixel is used just once for each timestep in the diffusion process (equivalent to using a stride of 64). The algorithm is thus significantly faster compared to the full MultiDiffusion (shown in the first row of each group in Fig. 16), with virtually no loss in quality. Wang et al. [2024] also proposed a similar way to improve upon MultiDiffusion in regards to the sampling of patches to be denoised. We, however, do not combine our noise uniformization with this method as their mechanism for sampling the random patches is not described with sufficient detail.

S3 Additional details on the tileable texture synthesis of arbitrary size.

Section 3.5 of the main paper presents our approach to generating textures of arbitrary size using constant GPU memory. Crucially, our noise-uniformization preprocessing ensures that the generated textures are stationary. To avoid seams caused by the patch-based denoising, we set the pixel overlap range to 32 (*i.e.*, half the window size). This means the number of evaluations of the diffusion model is 4 times larger compared to an independent generation of the same number of pixels. This constant factor aside, the complexity of texture synthesis is linear with respect to the images resolution, meaning that generating large textures in this manner is time consuming (*e.g.*, 170 seconds for a 4096×4096 image). We approach this issue by making our textures tileable; this is done by wrapping around the sliding windows circularly during denoising. This strategy directly ensures tileability in the latent space in which the diffusion model operates. In order to obtain tileable texture in RGB space, we first circularly pad the output of the multi-diffusion. Then, the enlarged map is decoded using the Stable Diffusion VAE [Rombach et al. 2022], followed by a crop to the original (unpadded size). The decoded output can then be easily tiled. This allows trading quality

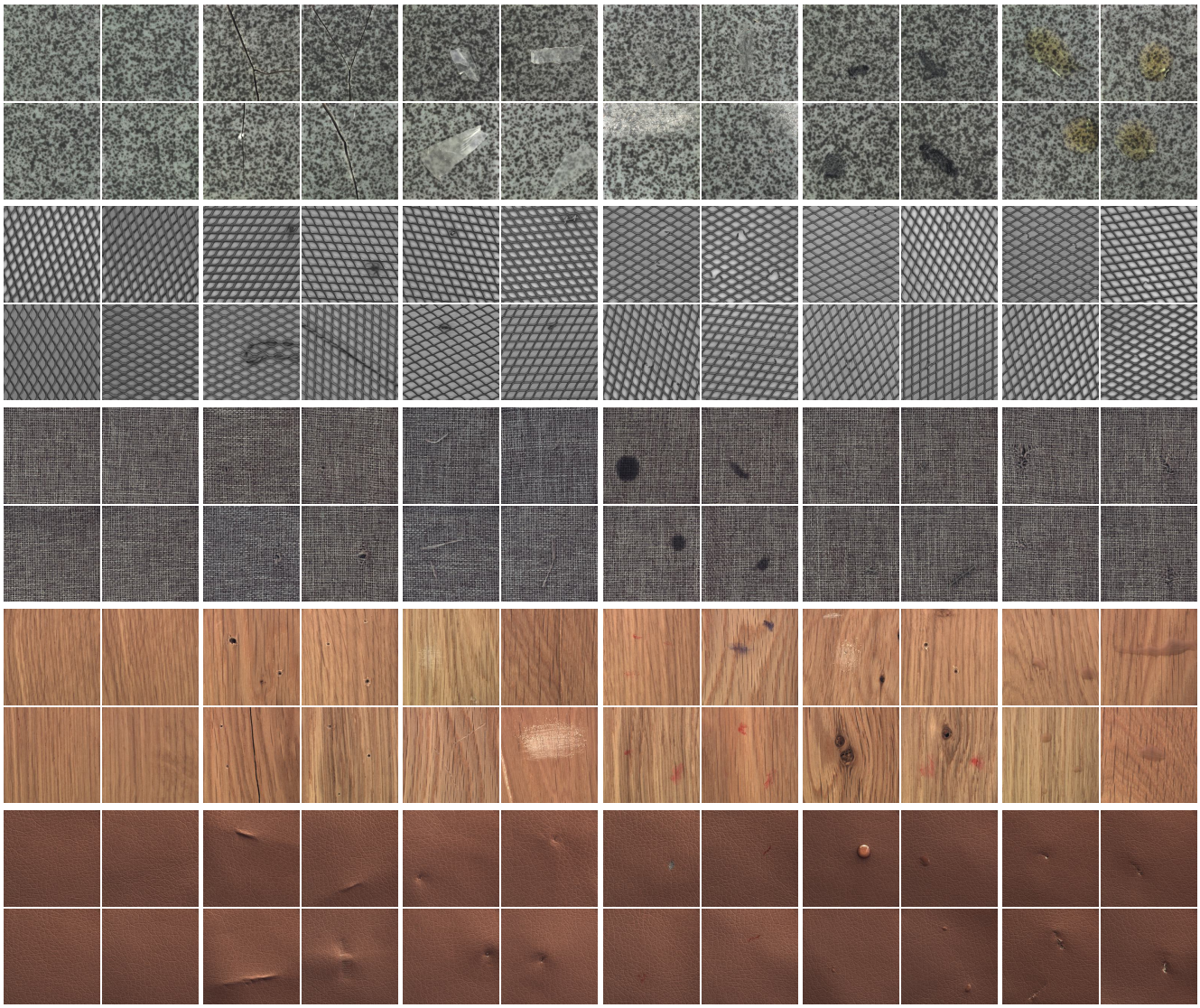


Fig. 13. Overview of the image data in the MVTec AD textures.

for efficiency by synthesizing textures up to a certain resolution, after which tiling is used to achieve the final size.

In Fig. 17 we present 6 such textures, that were generated at a 2048×2048 resolution and then tiled into 2×2 blocks, resulting in seamless 4096×4096 images. Our noise-uniformization technique reduces the impression of repeatability by making each tile more stationary. This improvement especially visible for the grid texture (second row of Fig. 17).

One can also note a difference in overall color and contrast between the images in the two columns. This appears as a consequence of the noise averaging, specific to MultiDiffusion [Bar-Tal et al. 2023]. Averaging different diffusion paths reduces variance and changes the final result for a specific window compared to what would have been obtained using the noise in that window in isolation. Thanks to our noise uniformization, the first moment of the noise varies much less over different windows, which keeps the denoising trajectory from diverting too much from the original path.

S4 Conditional generation on MVTec Textures

In Fig. 18 we show additional results generated with our method. The results on MVTecAD textures shows that our method generalizes to diverse shapes and multiple anomaly types per image.

S5 Uncurated set of generated images

In Fig. 19 we present a large set of images generated by our model, showing each texture-class feature-type combination for 35 different monochrome SVG icons from the internet (Flaticon.com).

Fig. 14. Overview of our 9 textures, captured using a handheld phone camera.

S6 Additional high-resolution results

Our method lends itself to high-resolution generation and editing. In Figures 20, 21, we add results generated at high-resolution, presenting various painted features. Fig. 22 includes additional high-resolution editing results.

S7 Comparison Addition

We include in Fig. 23 a representative failure case of Diffusion Texture Painting [Hu et al. 2024], as referenced in the main paper.

S8 Comparison of different thresholding functions

In Section 3.1, we introduce our thresholding function, together with the theoretical justification for combining a local and global threshold. In Fig. 24 we present some representative results for alternative thresholding schemes that we considered. That is, we compare our approach to using local (per-image) quantiles, dataset-level quantiles, and Otsu's method.

S9 Timings

As presented in the main paper, our pipeline is trained in 3 stages: anomaly detection, feature clustering, and diffusion-based synthesis. The first stage resembles the detection part of BlindLCA [Ardelean and Weyrich 2024b], which takes about 5 minutes. The second stage performs the binarization, mines the positive and negative pairs of connected components, and performs the contrastive learning; the time is dominated by the last step, taking around 15 minutes. The third stage is by far the most time-consuming: training the diffusion model to convergence takes between 6 and 12 hours on an A5000 NVIDIA GPU. This time, however, could be greatly reduced by using a more fitting pre-training, with more textures compared to the DTD dataset used in our experiments.

The training time is a drawback of our approach compared to single-exemplar texture synthesis methods (*e.g.*, Image Analogies [Hertzmann et al. 2001], Guided Correspondence [Zhou et al. 2023]); however, after training, our method can generate new images quickly, enabling interactive texture authoring. During inference we perform 18 steps with the Heun solver, taking around 1 second for a 512×512 image, as shown in Table 3. This Table, however, shows the timing for generating a new texture with a given mask. Editing an existing image requires an additional inversion step. Since we use an Euler solver with 4 fixed-point iterations, the number of diffusion model calls (NFE) is significantly larger. In the experiments performed in the main paper we use 250 steps (1000 NFE), taking about 15.5 seconds per image. After inversion, our noise-mixing editing does another 250 NFE, taking around 4.5 seconds. The time spent on encoding and decoding using the Stable Diffusion VAE is negligible, which means the total time for editing is 20 seconds. That being said, our interactive editing framework benefits from the ability to dynamically choose the numbers of steps performed by the diffusion model. Namely, the user can use a small number of steps at the beginning of the editing sessions and only use the full number of steps to synthesize the final result. Importantly, our noise-inversion on real images performs well even with significantly fewer steps, as seen in Fig. 25. Note that after 40 steps the returns in quality are minimal and the inverted image converges to the VAE representation. The mean absolute error (MAE) is dominated by the loss of information during encoding-decoding. For the interactive editing (as presented in the supplementary video) we only use 42 steps.

Table 3. Comparison of inference time.

Method	Time (s) ↓
Image Analogies [Hertzmann et al. 2001]	1200
Guided Correspondence [Zhou et al. 2023]	224
Neural Style Transfer [Gatys et al. 2016]	75
Texture Reformer [Wang et al. 2022b]	0.34
Ours	0.98

S10 Detailed numerical results

We include in Table 4 the detailed quantitative results of our method, and compare them with the metrics obtained using BlindLCA [Ardelean and Weyrich 2024b]. We use macro averaging for the F1-score to emphasize the importance of detecting all feature types. Note that

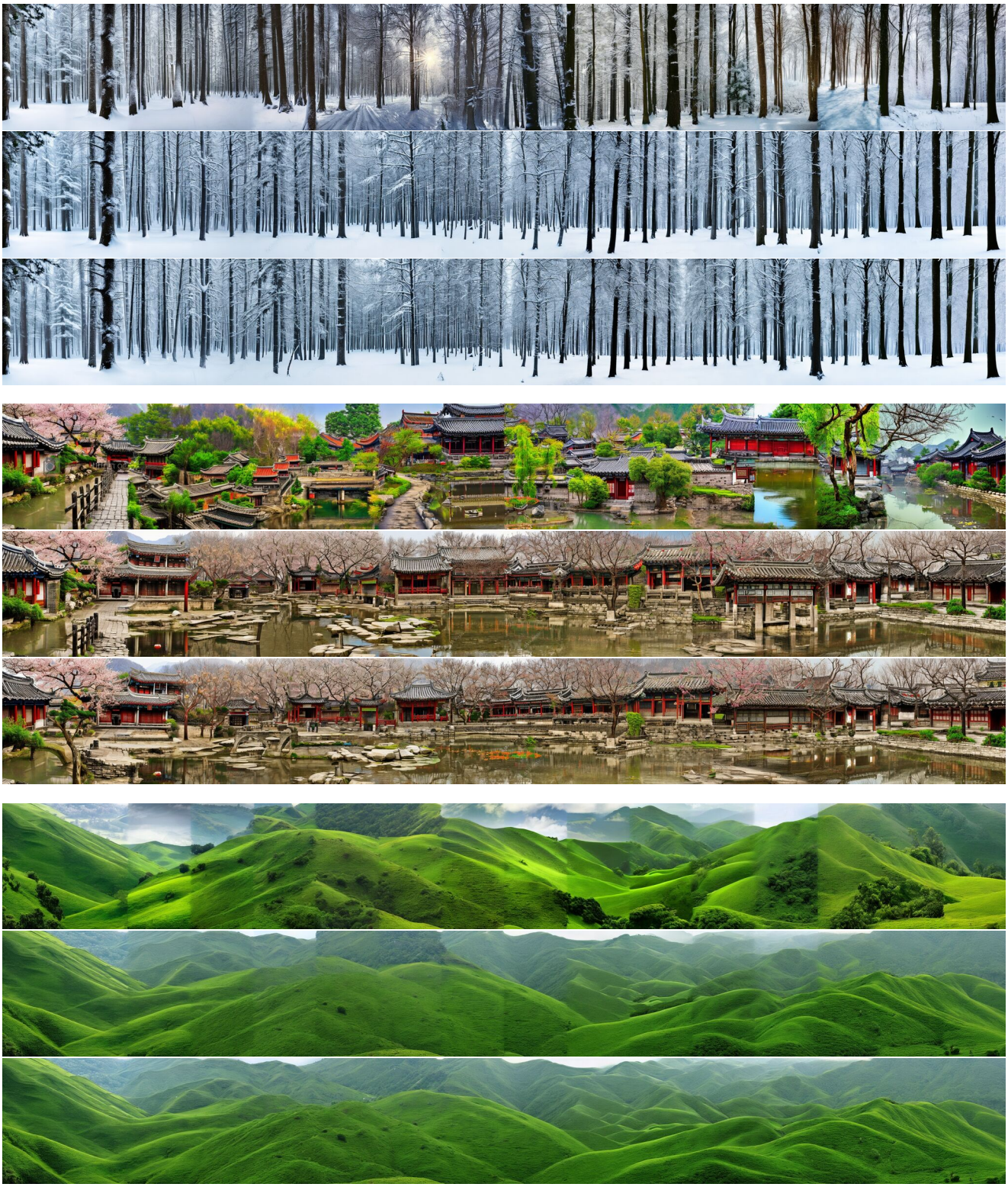


Fig. 15. Application of our noise uniformization for generic images synthesized using MultiDiffusion [Bar-Tal et al. 2023]. For each prompt the top image is obtained using pure white noise, the second image uses our harmonized noise, the third image additionally uses randomized sliding windows.

Fig. 16. Application of our noise uniformization in conjunction with noise rolling [Vecchio et al. 2024]. For each image triplet, the first row is obtained from **white noise using MultiDiffusion** with stride 1, the second row represents image generation using **noise rolling from white noise**, the third row uses **noise rolling on top of our noise uniformization**.

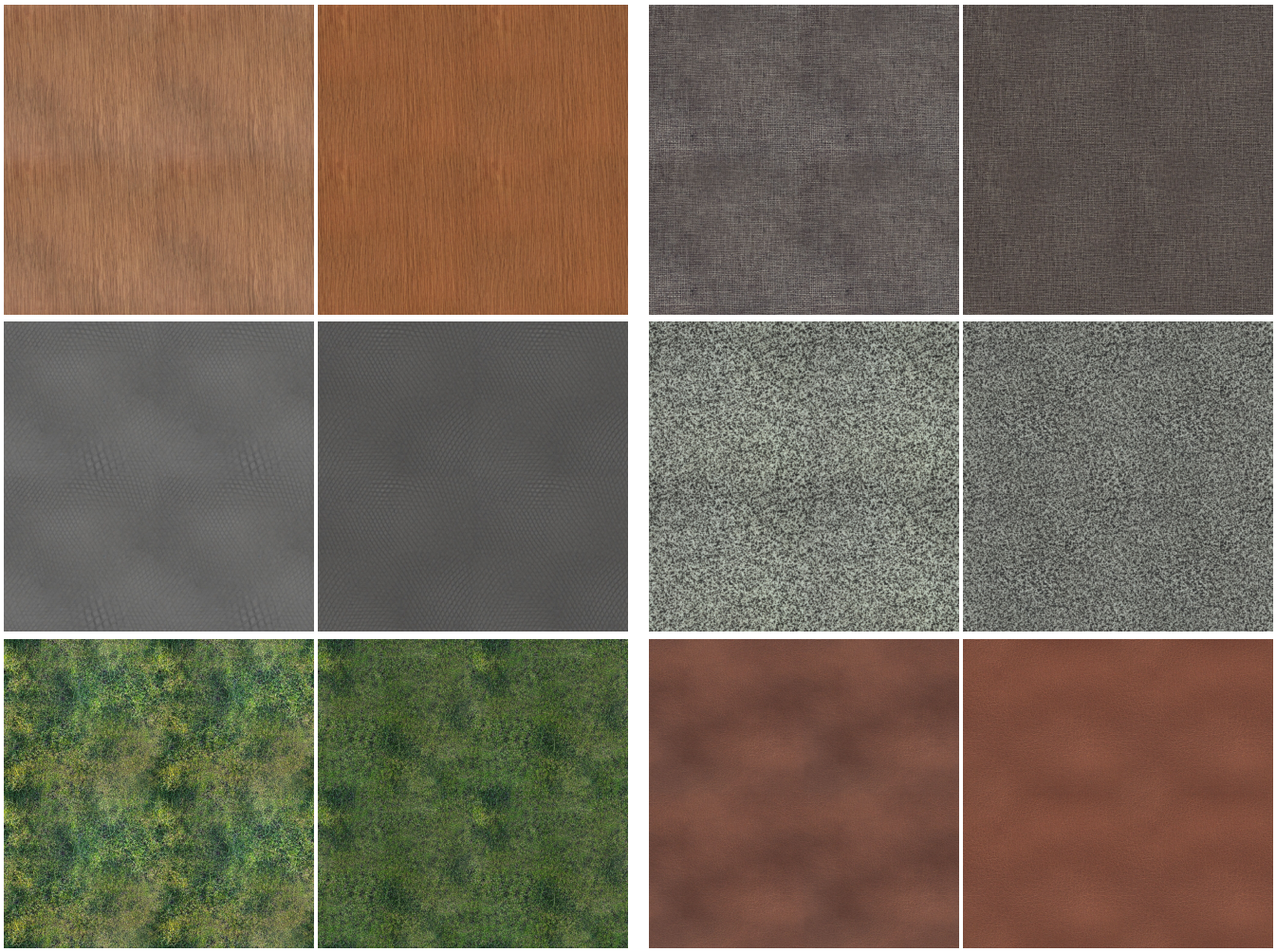


Fig. 17. Visualization of the generated tiled textures starting from white noise (left column) and our noise-uniformization method (right column). The multi-diffusion synthesizes tileable textures of 2048×2048, which are then tiled to form 4096×4096 images.

Table 4. Detailed quantitative results and comparison.

Texture	**Ours**			**BlindLCA**		
	Acc. ↑	IoU ↑	F1 ↑	Acc. ↑	IoU ↑	F1 ↑
Tile	**0.97**	**0.70**	**0.80**	0.96	0.54	0.67
Wood	0.96	**0.48**	**0.56**	**0.97**	0.41	0.50
Leather	**0.99**	**0.47**	**0.57**	0.99	0.39	0.50
Carpet	**0.99**	**0.42**	**0.53**	0.99	0.39	0.49
Grid	**0.98**	**0.30**	**0.38**	0.53	0.17	0.25

Table 5. Quantitative comparison of texture synthesis with different diffusion models. "+ DTD" denotes that the model was pretrained using the DTD [Cimpoi et al. 2014] textures dataset.

Model (FID ↓)	Tile	Carpet	Grid	Wood	Leather
SD + ControlNet	95.76	166.52	162.45	81.14	102.86
EDM2	41.60	132.12	125.70	38.67	126.05
EDM2 + DTD	46.23	88.31	92.69	34.24	103.33
EDM + DTD	50.04	57.14	81.82	68.29	105.88

this experiment uses a setting favorable to BlindLCA; that is, we excluded images that contain more than a single anomaly type per image. Nonetheless, our approach consistently yields better metrics.

S11 Discussion on the choice of generative model

Our feature painting framework is composed of several components that work together to enable the authoring and editing of textures with prominent features, which are learned from a small number of images. The generative model is the component that links the anomaly segmentation to the various desired capabilities of the system (see points 2-4 in the introduction). We choose to pose the generation as an image-to-image translation task (spatial labels to texture) using a diffusion model, and combine it with our noise-mixing and noise-uniformization to facilitate editing, feature

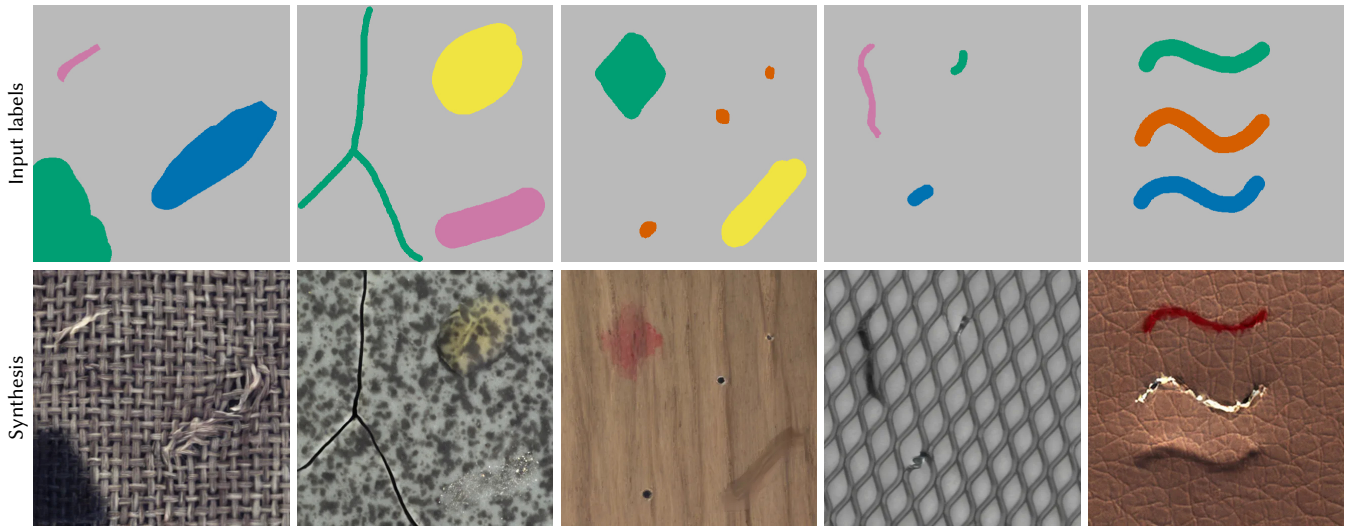


Fig. 18. Additional results on feature-conditioned texture generation by our model.

Table 6. Quantitative comparison of texture feature painting using different diffusion models; EDM and EDM2 were pretrained on DTD [Cimpoi et al. 2014]. The models are evaluated by the accuracy of a image-to-class segmentation network trained with ground-truth labels.

Model (Acc ↑)	Tile	Carpet	Grid	Wood	Leather
SD + ControlNet	92.34	48.99	41.22	95.72	**98.27**
EDM2	94.40	93.86	90.58	95.49	86.82
EDM	**96.63**	**97.94**	**91.66**	**96.17**	97.84

Table 7. Timing of different diffusion models

Method	Throughput (img/s) ↑	Latency (ms) ↓
SD + ControlNet	0.8	2720
EDM2	**1.4**	2430
EDM	1.3	**982**

transfer, and large texture generation. Alternatively, the generative task could be formulated as inpainting, to naturally support editing. While arbitrary-size texture generation could in this case be formulated as out-painting, it is not clear how the other capabilities could be obtained. For example, it is not trivial to enable feature transfer, or to ensure that the painted feature is consistent with the initial texture (see Fig. 12 and Fig. 10).

Another possible approach is to use an autoencoder or a VAE and encode the different types of features in latent space. A texture synthesis method that leverages this idea is TextureMixer [Yu et al. 2019]. This approach could potentially be used within our framework by using the pixel-level anomaly segmentation masks to extract the rare features in a format compatible with TextureMixer's training process. Nevertheless, it is unclear to what extent such method can adapt to thin structures (such as cracks) or how it can be extended to support feature transfer.

Other generative approaches, such as autoregressive models or flow-matching could be similarly considered. In general, however, we consider it out of the scope of this paper to incorporate all these methods in our framework to evaluate their performance on the various capabilities. That being said, our choice of diffusion model is made without loss of generality within that category (spatially-conditioned diffusion models). Our system uses EDM [Karras et al. 2022] as the backbone for texture synthesis and editing; nonetheless, the proposed noise-mixing and noise-uniformization algorithms can be applied using virtually any diffusion model. In our experiments, we use EDM because we find it to strike a good balance between image quality and speed. In the following, we provide a quantitative comparison between EDM and two alternative diffusion models, namely Stable Diffusion (SD) [Rombach et al. 2022] and EDM2 [Karras et al. 2024]. In order to evaluate the image quality for a certain texture, we generate 100 images of the normal class and take 64 patches of size 128×128 from every image. In the same manner, we also extract patches from the real images without anomalies, which were not seen during training. Finally, the distribution of the generated and real patches are compared using the Fréchet Inception Distance (FID) [Heusel et al. 2017]. We evaluate the three different models and in Table 5. It can be seen that pretraining the models on the DTD [Cimpoi et al. 2014] dataset improves synthesis quality; however, rather surprisingly, Stable Diffusion performs worse despite its large scale pretraining. Note that we have experimented with both full-model fine-tuning and ControlNet [Zhang et al. 2023], and we observed superior performance with the later. We further evaluate the models ability to generate realistic prominent features on the textures in Table 6. As the number of anomalous patches in MVTec is very small, and all anomalous images have been used for training the diffusion models, it is unfeasible to use FID in this case. Therefore, we evaluate the models based on the ability of a segmentation network to correctly classify the generated features. We train a CNN on the MVTec ground-truth labels and generate a

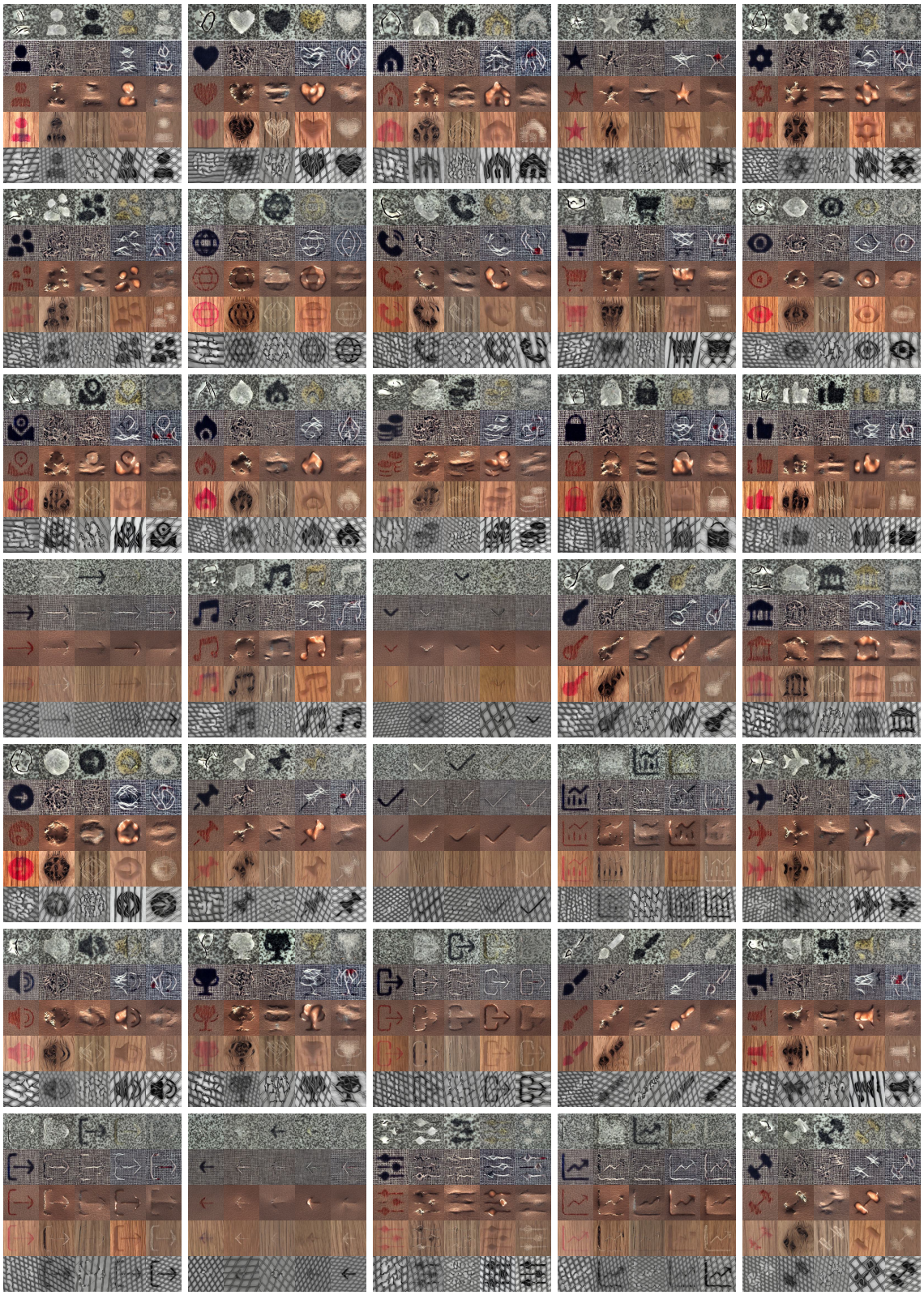


Fig. 19. Large set of uncurated images generated by our model based on icons from Flaticon.com. There are 7×5 different icons, for which we generate images with 5 anomalies for 5 different textures, for a total of 875 images.

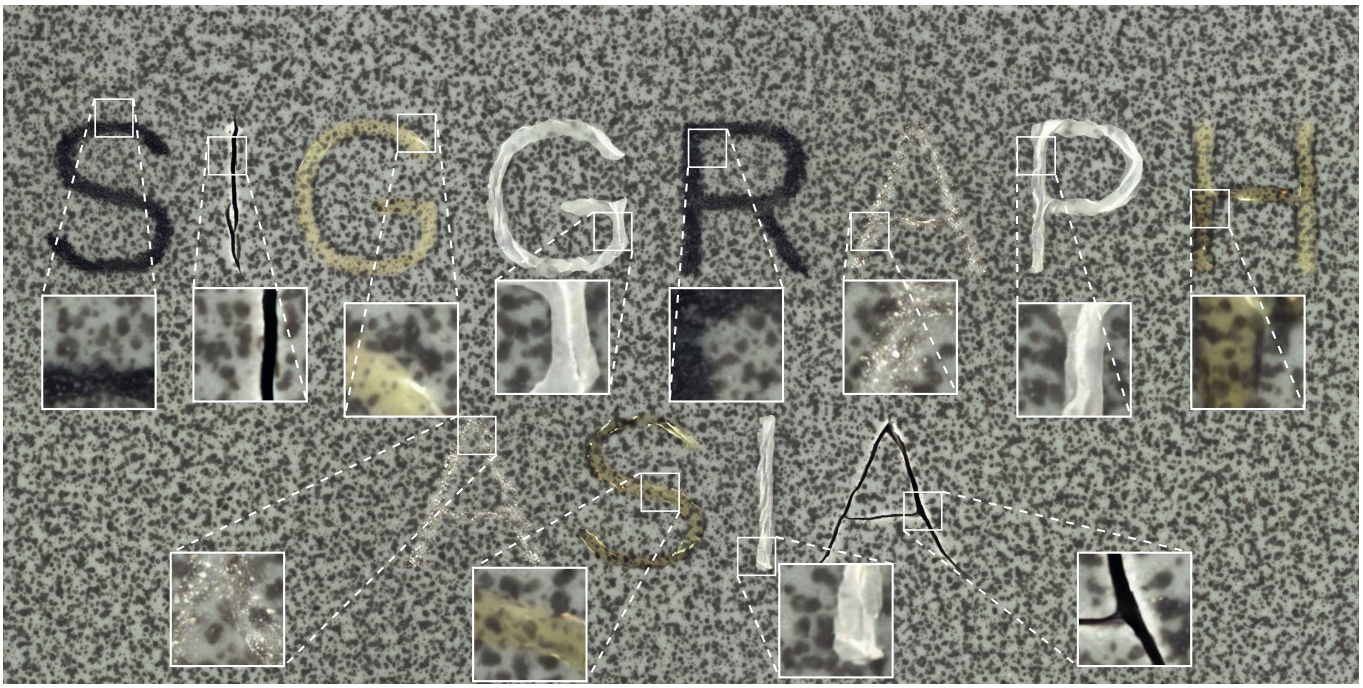


Fig. 20. High-resolution (2048×4096) conditional generation example.

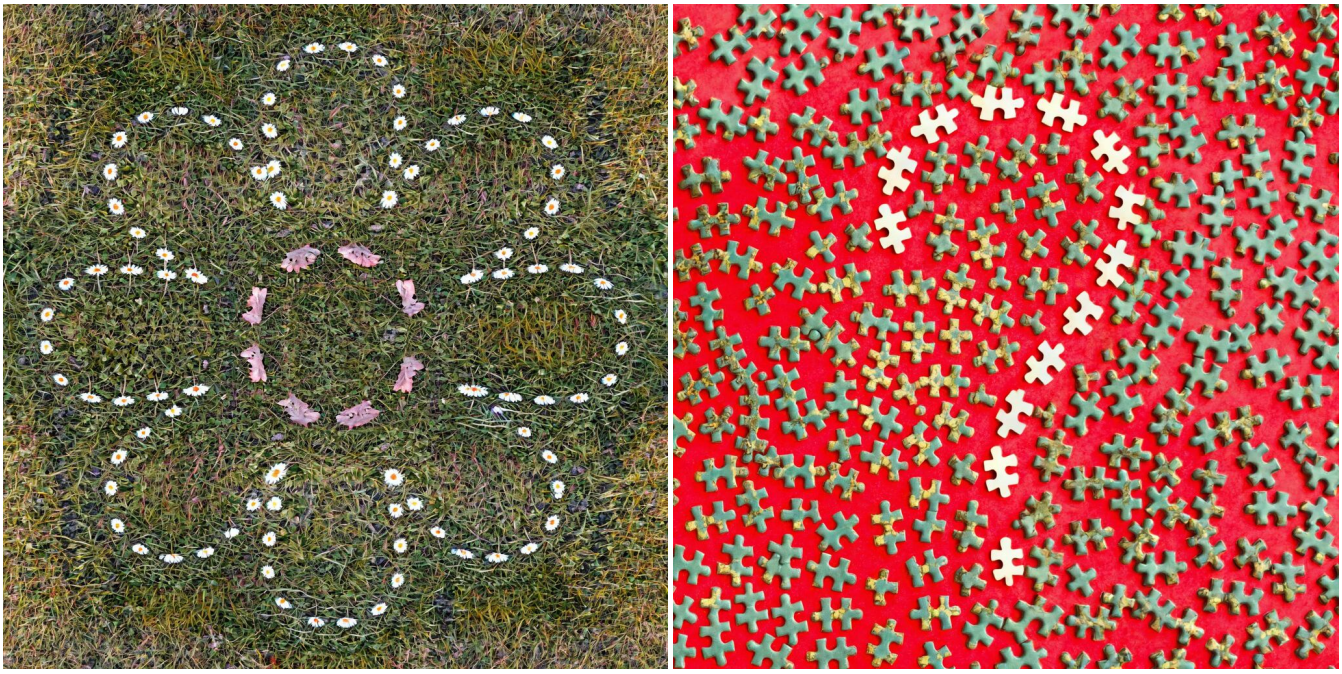


Fig. 21. High-resolution conditional generation examples. *Note that the puzzle model was trained from a single image (see Fig. 14)*

set of semantic masks as test data. The diffusion models are conditioned on these masks to generate a set of 256 images, which are then segmented with the CNN. Table 6 reports the accuracy of these segmentations; a higher accuracy indicates that the features better reflect the training data (real MVTec anomalies).

The throughput and latency of the three methods are compared in Table 7. SD has the highest computation time of the three methods.

Fig. 22. High-resolution (4096×4096) editing examples. **Please zoom in for full resolution.**

While the throughput of EDM2 is similar to EDM, the latency is significantly higher, despite using the smallest (XS) variant of EDM2. It can be reduced to 1166 milliseconds by using the same model for guidance, which allows computing the conditional and non-conditional noise directions in parallel. Overall, these experiments suggest that EDM is a good choice for our use-case, considering the high-quality synthesis with a low latency. Finally, we emphasize that even though we show that our choice of diffusion model is sound, and that good results can be obtained with a relatively small model with very little pretraining, we do not claim to have found the best possible model for this part of the pipeline, as this is not the scope of our work.

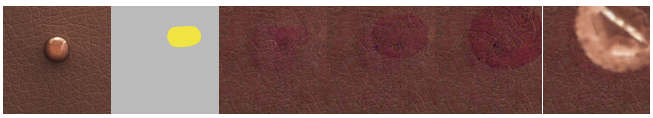


Fig. 23. Failure case for Diffusion Texture Painting [Hu et al. 2024]. The results are generated with a progressively dilated mask. Only at the largest size (dilated by 77 pixels), the feature is actually painted.

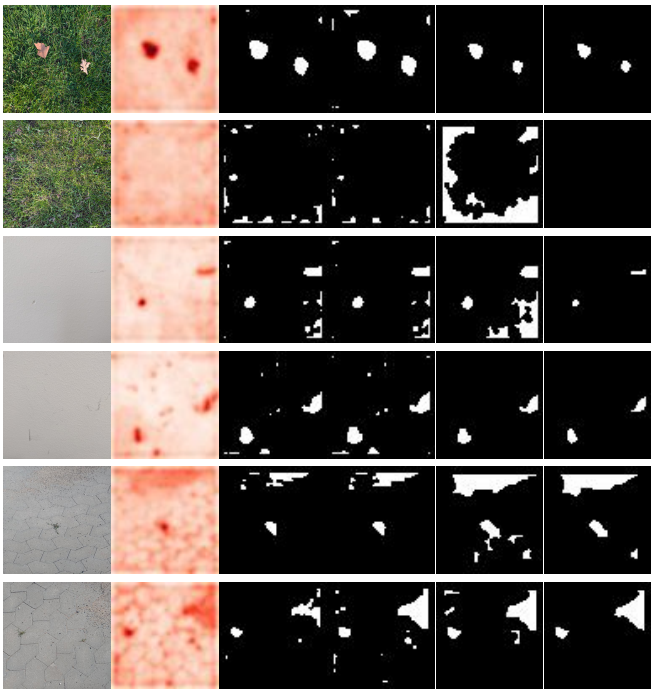
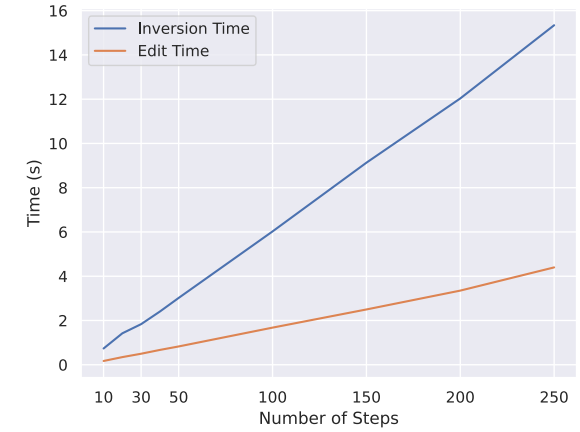


Fig. 24. Comparing the binarization of anomaly scores using different thresholding functions.



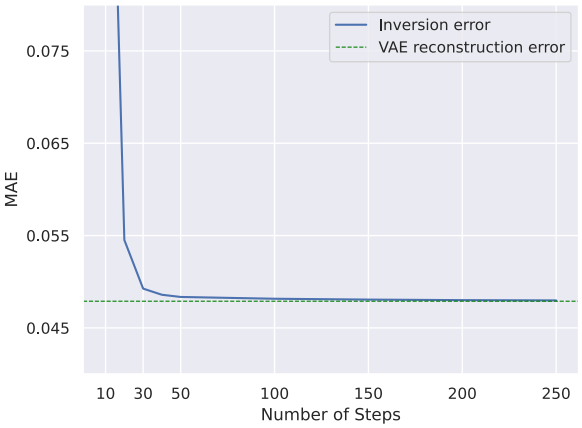


Fig. 25. Timing and errors for diffusion inversion.

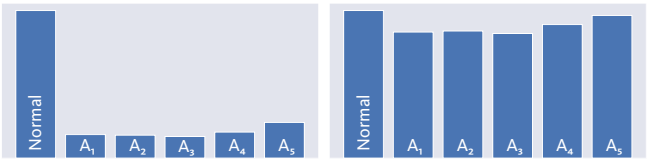


Fig. 26. Histograms of sampled negatives from each anomaly class, used for contrastive learning: uniformly sampled (*left*) and using our stratified sampling (*right*). A disproportionate number of normal samples hinders the proper separation of the anomaly classes.

S12 Implementation Details

Several implementation details have been omitted for brevity in the main text. We expand here the explanation of various steps and the hyperparameters used.

S12.1 Anomaly detection

In the first stage of the pipeline, we apply FCA to the residuals of the VAE reconstruction. We use a similar preprocessing to Ardelean and Weyrich [2024b]: resize the images to 512×512, use a Wide ResNet-50 [Zagoruyko and Komodakis 2016], and train the VAE for 10k iterations. After subtracting the original features from the reconstruction we use FCA with a patch size of 7×7, $\sigma_p = 3$, and $\sigma_s = 1$.

S12.2 Semantic feature segmentation

In order to enable an efficient training of the segmentation network through contrastive learning, we first build a database of positive and negative pairs. As described in section 3.1, we first binarize the anomaly maps from the previous step using an adaptive threshold. Afterward, we form groups of prominent features by finding the connected components. To reduce some of the noise that arises from binarization, we perform a small erosion (2×2) and then eliminate objects smaller than 12 pixels. For each region, we then compute a neural descriptor by averaging the ResNet features from the pixels inside the mask. We employ a weighted average using the softmax of the scores predicted by FCA, similarly to Sohn et al. [2023]. The

descriptors are used to compute pair-wise distances between all feature groups. To sample the positive pairs we simply take the closest $p = 10$ feature groups in terms of distance. Our stratified sampling of the negative pairs is more involved: the closest 50% groups are first discarded as potential positives; then, the remaining descriptors are clustered using k-means to obtain coarse group categories. We then select a number of n negatives in a stratified manner from this pool, where n is calculated as the expected number of groups. That is, the total number of groups minus the largest cluster, which contains normal features. Since n is generally smaller than 50% of the original number of groups, the negatives are distributed more uniformly across the different types of prominent features.

Our segmentation network takes as input the ResNet features and computes task-aligned descriptors through contrastive learning. The network consists of only 3 convolutional layers with kernel sizes: 3×3, 3×3, 1×1, LayerNorm normalization, and GeLU activations. During training, we use an additional 2-layer MLP head that is not used for clustering, as it is customary in self-supervised learning [Chen et al. 2021b; Grill et al. 2020]. We train the network for 10K iterations and then obtain per-pixel descriptors for all images. Finally, these descriptors are clustered independently using k-means; the number of classes (prominent feature types) is assumed to be known by the user. Generally, we believe the user would have a reasonable understanding of the features present in the dataset and choose the granularity of the clustering according to their use-case. Alternatively, the user could simply use a clustering method that automatically detects the number of classes, such as DBSCAN.

S12.3 Synthesis

To support conditional synthesis, we adapt the diffusion architecture used by EDM to incorporate spatial label maps. Firstly, we lift the noise embedding to a H×W×C tensor instead of a C-dimensional vector. Then, we compute label embeddings using two convolutional layers, and we add them to the noise embeddings. Finally, we use another 1×1 convolution before propagating this spatial embedding to all the U-Net blocks. This minimal modification of the architecture effectively enables the control of the model through the label mask.

We pretrain the model on the DTD dataset for 750K iterations, taking around 12 hours. Then, we separately fine-tune the model for each texture for another 750K iterations. The diffusion is performed in the latent space of SD, making it more efficient to generate high-resolution images. The only exception is the SVBRDF synthesis experiment, which performs the diffusion directly in the space of material maps. All models are trained at a resolution of 64×64. We use Pytorch's RandomResizedCrop augmentation, along with random horizontal and vertical flips, and a slight color jitter.

For interactive editing, we developed a Blender script that leverages the native tool for painting masks on textures. We process the masked image to extract the desired edit and take a bounding patch of at least 442×442 around the masked region. Our noise-mixing algorithm is then applied with the Euler solver for 42 steps. The average edit latency is 1.5 seconds, which enables interactive asset modifications (as seen in the video attached to this supplementary material).

S12.3.1 Noise uniformization. Our noise uniformization algorithm specifies a way to make a noise map $\boldsymbol{w}$ more consistent with a different noise tensor $\boldsymbol{z}$, which dictates the *style* of the instance (overall color, contrast, pattern density, etc). We achieve this by making the noise map follow the same low frequency distribution, as described in the main paper:
$\boldsymbol{w}' := \boldsymbol{w} - \texttt{blur}(\boldsymbol{w}) + \texttt{upscale}(\texttt{shuffle}(\texttt{downscale}(\texttt{blur}(\boldsymbol{z}))))$.
For blurring, we use a Lanczos filter with a cutoff frequency $f_c = 0.1$, and we downsample the filtered noise to a resolution of 32 × 32. For the supplementary experiments, based on StableDiffusion (Fig. 15), we use the same parameters, except that we only perform the downsampling and shuffling along the columns. This is because the generated images resemble panoramas, which only have a stationary nature along the width of the image. To create a large uniform noise tensor, we first generate white noise and then divide the map into equally-sized non-overlapping patches. The first patch conveys the style ($\boldsymbol{z}$). All other patches ($\boldsymbol{w}$) are modified using the algorithm described above to follow this prototype; finally, the patches are rearranged to form the large noise map, which is the input for the diffusion model.

S12.3.2 SVBRDF. The main difference between the synthesis of material maps compared to RGB images is that we do not use a latent model for SVBRDFs. We train the model with 64×64 patches from the input material and do not use color jitter for this experiment. Since there is only one SVBRDF as input, we did not use contrastive learning for the semantic segmentation. Instead, as the irregularities are easily noticeable in either the albedo or the roughness maps, we computed embeddings using a random of set of 5×5 filters applied on the pixels' features. The resulting descriptors were directly clustered using k-means.

S12.4 Video

The video attached to this supplementary material contains an interactive editing session in Blender. Here, we make use of the brush tool from Blender, which allows the user to paint on a 3D mesh, while enabling programmatic access to the updated underlying UV-mapped 2D texture. We use the mask-painted texture to extract the semantic conditioning and then apply our trained method on the texture patch which is being edited. To improve the latency, we preload into GPU memory the weights of the model and the diffusion trajectory needed for our noise-mixing. This is done in a background thread when a certain object is selected for editing, so that a single set of weights must be stored in memory at a specific time. All textures in the scene have a resolution of 2048×2048, and they have been generated using our method except for the teapot, for which we use an arbitrary image to showcase our feature transfer capabilities. The blemishes are transferred from the MVTec tile texture. Note that we only apply our method to the base color of the materials, leaving the other material maps unchanged.

S12.5 Code Release

The code is available at: github.com/TArdelean/FeaturePainting.