Supplementary – Table of Contents

In this supplementary material, we provide following additional details regarding the proposed model and data:

- In Section A we describe **how traing and eval data was obtained**. Visualizations of real and fake image pairs for all evaluation datasets in the proposed **SynRIS benchmark** can be found in the end of the document in Fig. F.1-F.7.
- In Section B we provide a **derivation** of the relationship (8) from the main paper - connecting likelihood with inversion maps, DDIM discretization and reconstruction errors.
- In Section C we provide technical details of how our model was **trained**: architecture, inversion and captioning models we used.
- In Section D we provide **extended evaluation** results: Average Precision (AP) and overall accuracy for all evaluated methods, ROC, PR and DET curves, evaluation of robustness to prompt shift and fine-tuning, and the effect of text conditioning.
- In Section E we discuss how baselines were trained and issues we encountered when evaluating DIRE.
- In Section F, we include sample visualizations from the various datasets of our **SynRIS benchmark**.

Ethics and Limitations

The ultimate goal of this work is to prevent abuse and the spread of misinformation, an inherently ethical task. When generating our datasets, we sourced our prompts from an existing database. As such, some of the generated images may inadvertently contain inappropriate content. We explicitly address misalignment between fake and real images in our training and evaluation to ensure that the detector is not favouring any styles or themes to avoid marginalization of any groups.

While our method performs well at detecting images from existing diffusion models, this same performance may not transfer well to text-to-image models that do *not* make use of diffusion, such as text-to-image GANs (GigaGAN [28]) and transformers (Muse [15]). Training our model also requires significantly more compute than similar methods (CNNDetect [54], DMDetect [17]) since we must first pass all training images through our inversion pipeline.

Acknowledgements

We would like to thank J. P. Lewis, David Marwood, Shumeet Baluja, Sergey Ioffe and Arkanath Pathak for their feedback and technical advise.

A. Data

In this section we discuss how training and evaluation datasets were generated. Our new training set along with all our RIS-based evaluation benchmark can be found at our project page: [will be released with camera ready].

A.1. Training Data

We train our method and other baselines on two different training sets: ProGAN + LSUN and DiffusionDB + LAION (DDB-L).

A.1.1 ProGAN+LSUN

Authors of CNNDetect [54] introduced this dataset along with their GAN detection method. This dataset consists of 360k real images from LSUN [59] and 360k fake images generated by ProGAN [30], each composed of 20 different classes. All real and fake images are 256x256 resolution.

A.1.2 Stable Diffusion+LAION

For this training set, we took random 300k fake images from DiffusionDB [56] and random 300k images from LAION [49] with predicted aesthetic scores of 6.25 or higher. Note that DiffusionDB consists of images from Stable Diffusion v1.

A.2. Evaluation Data - Fakes

This sections provides detail how each evaluation dataset's *fake* images were obtained.

A.2.1 Imagen

We obtained Imagen images from authors of Imagen – they generated them using an internal closed API using the same prompt distribution that was used to train the Imagen model.

A.2.2 Midjourney

For our Midjourney images, we use this dataset on Hugging Face (link). These images were scraped from the Midjourney Discord server. This dataset includes a tag indicating whether or not the image was "upscaled" by the user. We choose images that had been upscaled since they are presumably of higher quality (since the user spent additional credits to upscale them).

A.2.3 DALL·E 3

For our DALL·E 3 images, we use this dataset on Hugging Face (link). These images were generated by users and shared on the LAION Discord server.

A.2.4 Kandinsky 2

For our Kandinsky 2 images, we use the Kandinsky 2.2 [51] model from Hugging Face (link), using the default parameters given in their usage example:

- `prior_guidance_scale=1.0`
- `height=768`
- `width=768`
- `negative_prompt=''low quality, bad quality''`

A.2.5 Kandinsky 3

For our Kandinsky 3 images, we use the Kandinsky 3 [10] model from Hugging Face (link), using the default parameters given in their usage example:
- `num_inference_steps=50`

A.2.6 PixArt-α

For our PixArt-α [16] images, we use the 1024 resolution model from Hugging Face (link). All parameters are left as their defaults.

A.2.7 Playground 2.5

For our Playground 2.5 images, we use the Playground 2.5 [31] model from Hugging Face (link), using the default parameters given in their usage example:
- `num_inference_steps`
- `guidance_scale=3`

A.2.8 SDXL Direct Preference Optimization

For our SDXL-DPO images, we use the SDXL-DPO [53] model from Hugging Face (link), with the default parameters given in their usage example:
- `guidance_scale=5`

A.2.9 Stable Diffusion XL

For our SDXL images, we use the Stable Diffusion XL [41] model from Hugging Face (link), using both the base and refiner models with the default parameters given in their usage example:
- `num_inference_steps=40`
- `denoising_end=0.8`
- `denoising_start=0.8`

A.2.10 Segmind Mixture of Experts

For our Seg-MoE images, we use the SegMoE-4x2-v0 [58] model from Hugging Face (link), with the default parameters given in their usage example:
- `negative_prompt=''nsfw, bad quality, worse quality''`
- `height=1024`
- `width=1024`
- `num_inference_steps=25`
- `guidance_scale=7.5`

A.2.11 Segmind Stable Diffusion 1B

For our SSD-1B [21] images, we use the SSD-1B model from Hugging Face (link), using the default parameters given in their usage example:
- `negative_prompt="ugly, blurry, poor quality"`

A.2.12 Stable Cascade

For our Stable Cascade [39] images, we use the Stable Cascade model from Hugging Face (link), using the default parameters given in their usage example for the prior model:
- `height=1024`
- `width=1024`
- `guidance_scale=4.0`
- `num_inference_steps=20`

and decoder model:
- `guidance_scale=0.0`
- `num_inference_steps=10`

A.2.13 Segmind Vega

For our Segmind Vega [21] images, we use the Segmind Vega model from Hugging Face (link), using the default parameters given in their usage example:
- `negative_prompt="(worst quality, low quality, illustration, 3d, 2d, painting, cartoons, sketch)"`

A.2.14 Würstchen 2

For our Wüerstchen [39] images, we use the Wüerstchen v2 model from Hugging Face (link), using the default parameters given in their usage example:
- `height=1024`
- `width=1024`
- `prior_guidance_scale=4.0`
- `decoder_guidance_scale=0.0`

We will also include images and results from Wüerstchen v3, which is currently in beta development.

A.3. Evaluation Data - Reals

The corresponding *real* images for all of our evaluation sets were found via a reverse image search API provided by one of the major image search engines.

B. Derivations

In this section we show the relationship between likelihood, DDIM inversion and DDIM reconstruction error. Notably, in the first approximation, it can be expressed using these terms without explicit dependency on model parameters θ.

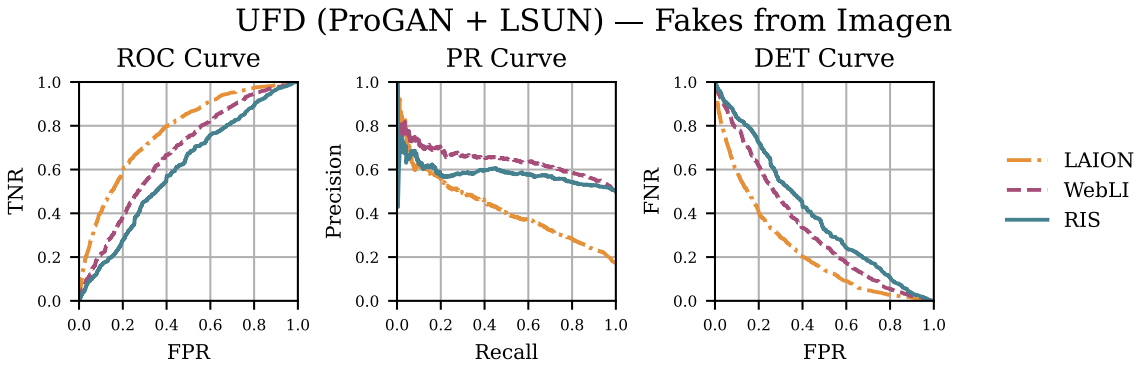


Figure A.1. Receiver Operating Characteristic (**left**) Precision-Recall (**middle**) and Detection Error Tradeoff **(right)** curves for detecting Imagen versus real images from its training set WebLI [60] (red), Reverse Image Search (green) and LAION (orange). These curves show that Imagen versus RIS is indeed a significantly harder task than Imagen versus LAION [49] and matches Imagen versus WebLI.

B.1. Derivation of Eq. (8)

Given an appropriate change of variable $\mathbf{f}$ we know:

$$\log p(\mathbf{x}) = \log \mathbf{p_z}(\mathbf{f^{-1}}(\mathbf{x})) + \log\det \mathbf{J}[\mathbf{f^{-1}}](\mathbf{x})$$

Rewriting the negative log Jacobain determinant as:

$$\begin{aligned}
&-\log\det \mathbf{J}[\mathbf{f^{-1}}](\mathbf{x}) \\
&= \log\det \mathbf{J}[\mathbf{f}](\mathbf{f^{-1}}(\mathbf{x})) \\
&= \mathbf{Tr}\left(\log \mathbf{J}[\mathbf{f}](\mathbf{f^{-1}}(\mathbf{x}))\right) \\
&= \mathbf{Tr}\left(\sum_{k=1}^{\infty}(-1)^{k+1}\frac{\left(\mathbf{J}[\mathbf{f}](\mathbf{f^{-1}}(\mathbf{x})) - \mathbf{I}\right)^k}{k}\right) \\
&\quad \text{(take the first term)} \\
&\approx \mathbf{Tr}\left(\mathbf{J}[\mathbf{f}](\mathbf{f^{-1}}(\mathbf{x})) - \mathbf{I}\right) \\
&\propto \mathbf{Tr}\left(\mathbf{J}[\mathbf{f}](\mathbf{f^{-1}}(\mathbf{x}))\right) \\
&\quad \text{(use Hutchinson estimator, assuming } \mu_v = 0, \Sigma_v = I) \\
&= \mathbb{E}_{\mathbf{v}}\langle \mathbf{v}, \mathbf{J}[\mathbf{f}](\mathbf{f^{-1}}(\mathbf{x}))\ \mathbf{v}\rangle \\
&\quad \text{(reparametrize with } \delta \text{ such that } \mu_\delta = 0, \Sigma_\delta = \sigma_\delta I) \\
&= \mathbb{E}_{\delta}\langle \delta, \mathbf{J}[\mathbf{f}](\mathbf{f^{-1}}(\mathbf{x}))\ \delta\rangle/\sigma_\delta^{\mathbf{2}} \\
&\quad \text{(take a single sample estimate)} \\
&\approx \langle \delta, \mathbf{J}[\mathbf{f}](\mathbf{f^{-1}}(\mathbf{x}))\ \delta\rangle/\|\delta\|^{\mathbf{2}} \\
&\quad \text{(Taylor expansion of } \mathbf{f} \text{ around } \mathbf{f^{-1}}(\mathbf{x})) \\
&\approx \langle \delta, \mathbf{f}(\mathbf{f^{-1}}(\mathbf{x}) + \delta) - \mathbf{f}(\mathbf{f^{-1}}(\mathbf{x}))\rangle/\|\delta\|^{\mathbf{2}} \\
&= \langle \delta, \mathbf{f}(\mathbf{f^{-1}}(\mathbf{x}) + \delta) - \mathbf{x}\rangle/\|\delta\|^{\mathbf{2}}
\end{aligned}$$

and substituting

$$\begin{aligned}
\mathbf{x} &= \mathbf{x_0} \\
\mathbf{f^{-1}}(\mathbf{x}) &= \mathbf{x_T} \\
\mathbf{f^{-1}}(\mathbf{x}) + \delta &= \hat{\mathbf{x}}_\mathbf{T} \\
\mathbf{f}(\mathbf{f^{-1}}(\mathbf{x}) + \delta) &= \hat{\mathbf{x}}_\mathbf{0}
\end{aligned}$$

and assuming a small enough isotropic δ, in the first approximation, we get:

$$\log p(\mathbf{x_0}) \propto \log \mathbf{p_z}(\mathbf{x_T}) - \langle \delta, \hat{\mathbf{x}}_\mathbf{0} - \mathbf{x_0}\rangle/\|\delta\|^{\mathbf{2}}$$

C. Training Details

In this section, we detail the various components of the pipeline used to train our detector.

C.1. Captioning

We use the BLIP-2, OPT-2.7b [32] model from Hugging Face (link) to caption our images *before* inversion and *after* and augmentations. Captioning is done *after* any augmentation since the augmentations could significantly change the caption (*e.g.*, an RGB image converted to grayscale).

C.2. Inversion

We base our inversion process on Pix-to-Pix Zero [38], making use of the Hugging Face implementation (link) without the additional attention map guidance and other regularizers.

We first resize to 512×512 and then invert *all* images (training and evaluation) using the same Stable Diffusion 1.5 [45] checkpoint from Hugging Face (link).

C.3. Training

The original images, inverted noise maps, and denoised reconstructions are then concatenated along the channel dimension and used as input to a ResNet-50 [22]. We train our detector for 25 Epochs, but most of the performance is gained in the first few. We otherwise use the same hyperparameters as CNNDet [54].

D. Extended Results

D.1. Extended metrics

In Table D.1 and Table D.2 we show AP and Acc@EER metrics for all experiments in addition to AUCROC. Figures

Train Data	ProGAN + LSUN			Stable Diffusion + LAION		
Eval Set | Model	CNNDet	UFD	Ours	CNNDet†	UFD†	Ours
DALL·E 2 [44]	0.499	0.694	**0.867**	0.653	0.750	**0.751**
DALL·E 3 [12]	0.473	0.384	**0.625**	0.703	0.474	**0.756**
Midjourney v5/6 [2]	0.498	0.419	**0.736**	0.602	0.555	**0.643**
Imagen [46]	0.474	0.582	**0.759**	0.705	0.553	**0.791**
Kandinsky 2 [51]	0.493	0.479	**0.764**	0.592	0.547	**0.695**
Kandinsky 3 [10]	0.491	0.470	**0.860**	0.654	0.605	**0.755**
PixArt-α [16]	0.490	0.502	**0.871**	0.623	0.625	**0.744**
Playground 2.5 [31]	0.510	0.462	**0.778**	0.556	0.571	**0.617**
SDXL-DPO [53]	0.495	0.475	**0.849**	0.829	0.683	**0.874**
SDXL [41]	0.506	0.470	**0.777**	**0.799**	0.651	0.792
Seg-MOE [58]	0.490	0.428	**0.800**	0.644	0.611	**0.704**
SSD-1B [21]	0.544	0.509	**0.840**	0.714	0.613	**0.787**
Stable-Cascade [39]	0.508	0.399	**0.892**	0.712	0.656	**0.766**
Segmind Vega [21]	0.524	0.475	**0.834**	0.723	0.612	**0.796**
Würstchen 2 [39]	0.508	0.592	**0.803**	0.600	0.670	**0.712**
DALL·E 2 [44] (A)	0.160	0.256	**0.620**	0.202	0.216	**0.587**
Craiyon [18] (A)	0.621	**0.977**	0.893	0.737	**0.922**	0.876
LDM [45] (A)	0.598	**0.933**	0.897	0.901	0.924	**0.976**
Average	0.493	0.528	**0.804**	0.664	0.624	**0.757**

Table D.1. **Main Results – Detector Average Precision.** In the main paper, we report the AUCROC metric when evaluating our classifiers. We report AP (Average Precision) here for completeness as well. We observe similar trends: our method performs best across nearly all datasets. *Note: This DMDet classifier was trained with fakes from an LDM checkpoint rather than Stable Diffusion. †These models were re-trained by us.

Figures D.2 to D.7 show ROC, PR and DET curves for all compared classifiers.

D.2. Very Out-of-Distribution Data

The generators analyzed thus far are trained to be *general* aesthetic text-to-image models.

Here, we evaluate our model on models that have been fine-tuned generate images from *very* specific domains: Anime and Pokémon. Table D.3 shows that even when using low-quality training data (ProGAN), our model still generalizes to these very out-of-distribution domains while existing methods (UFD [37], CNNDet [54]) completely fail, even performing worse than random guessing.

D.3. Text-Conditioning

In Figure D.1, we show the effect of text-conditioning on the inversion-reconstruction process. By using text-conditioning, we recover a more faithful reconstruction of the input image, providing better signal for our model.

E. Baselines

E.1. DIRE [55]

The DIRE [55] paper reports almost perfect detection performance on all unseen test sets. However, after their code was released, several researchers noticed a fundamental issue with the training and evaluation setup present in their released code and checkpoints (link) causing the in-the-wild performance to drop to near-random levels. More specifically, all pre-processed images used for training are saved with the **same extension** as the source images. Since all input *real* images in the training set are saved as `*.JPG` files and all fake images are saved as `*.PNG` files, all real DIRE images used to train and evaluate the network were embedded with JPEG artifacts while the non of the fake images used for training and evaluation were. As such, the model seemingly learned to detect the presence of JPEG artifacts. This holds even for the robustness experiments since augmented real and fake images are also saved as `JPG` and `PNG` respectively. In all of *our* datasets, both real and fake images are saved as lossless `*.PNG` files, explaining DIRE's poor performance on all our test sets. To honor the contribution of DIRE authors, we conducted an ablation that

Train Data	ProGAN + LSUN			Stable Diffusion + LAION		
Eval Set | Model	CNNDet	UFD	Ours	CNNDet†	UFD†	Ours
DALL·E 2 [44]	0.470	0.674	**0.773**	0.624	**0.700**	0.678
DALL·E 3 [12]	0.435	0.371	**0.592**	0.659	0.473	**0.698**
Midjourney v5/6 [2]	0.490	0.413	**0.661**	0.595	0.558	**0.606**
Imagen [46]	0.470	0.580	**0.706**	0.674	0.538	**0.720**
Kandinsky 2 [51]	0.490	0.483	**0.687**	0.574	0.541	**0.652**
Kandinsky 3 [10]	0.481	0.478	**0.766**	0.609	0.600	**0.684**
PixArt-α [16]	0.485	0.504	**0.769**	0.591	0.606	**0.669**
Playground 2.5 [31]	0.508	0.477	**0.707**	0.553	0.562	**0.591**
SDXL-DPO [53]	0.486	0.473	**0.764**	0.761	0.647	**0.801**
SDXL [41]	0.497	0.473	**0.688**	0.735	0.620	**0.737**
Seg-MOE [58]	0.472	0.429	**0.725**	0.625	0.586	**0.664**
SSD-1B [21]	0.545	0.506	**0.748**	0.665	0.585	**0.724**
Stable-Cascade [39]	0.477	0.383	**0.802**	0.652	0.633	**0.694**
Segmind Vega [21]	0.522	0.476	**0.741**	0.676	0.587	**0.733**
Würstchen 2 [39]	0.504	0.580	**0.715**	0.580	0.640	**0.658**
DALL·E 2 [44] (A)	**0.656**	0.620	0.590	0.556	**0.562**	0.558
Craiyon [18] (A)	0.610	**0.917**	0.783	0.699	**0.836**	0.810
LDM [45] (A)	0.591	**0.845**	0.793	0.837	0.840	**0.933**
Average	0.510	0.538	**0.723**	0.648	0.617	**0.700**

Table D.2. **Main Results – Acc @ Equal Error Rate.** We further report Acc@EER here as an additional metric. We observe similar trends: our method performs best across nearly all datasets. *Note: This DMDet classifier was trained with fakes from an LDM checkpoint rather than Stable Diffusion. †These models were re-trained by us.

	Detection Method		
Eval Set	**Ours**	UFD	CNNDet
Anime	**0.67**	0.13	0.34
Pokémon	**0.83**	0.32	0.48

Table D.3. **AUCROC**. When trained on lower-quality data (ProGAN), existing methods (UFD [37], CNNDet [54]) *completely* fail to generalize to our extremely out-of-distribution datasets, even doing worse than randomly guessing. On the other hand, our method continues to generalize well.

used the exact same absolute residuals signals as reported in DIRE paper. It is evident from our results that the inversion signals we propose in this paper perform much better then DIRE signals across all evaluation benchmarks.

E.2. CNNDet [54]

We trained using the original CNNDet training code (link) using following default parameters.

- `--name blur_jpg_prob=0.5`
- `--blur_prob=0.5`
- `--blur_sigma=0.0,0.3`
- `--jpeg_prob=0.5`
- `--jpeg_method=cv2,pil`
- `--jpeg_qual=30,100`

All training and eval images are resized to 256×256 and saved as `.PNG` files.

E.3. UFD [37]

We trained using the original UFD training code (link) using following default parameters.

- `--name=clip_vitl14`
- `--wang2020_data_path=datasets`
- `--data_mode=wang2020`
- `--arch=CLIP:ViT-L/14`
- `--fix_backbone`

All training and eval images are resized to 256×256 and saved as `.PNG` files.

Figure D.1. Unconditional DDIM inversion and reconstruction catastrophically fails to reconstruct the original image. To remedy this, we predict a caption using BLIP 2 [32] and use this for text conditioning.

F. SynRIS Visualization

Figures F.1 to F.4 contain samples from our evaluation sets using *closed-source* models. Figures F.5 to F.15 contain samples from our evaluation sets using *open-source* models. The corresponding images in each of these open-source model figures were generates using the same prompts. The top panel consists of *fake* images either found online or generated by us, and the bottom panel are the corresponding *real* images found via reverse image search.

Figures F.16 and F.17 contain samples from our Pokémon and Anime datasets respectively. The real images were taken from a source dataset [6, 40] and the attached captions were used to generate the fake images with SD Pokémon Diffusers [4] and Animagine [7].

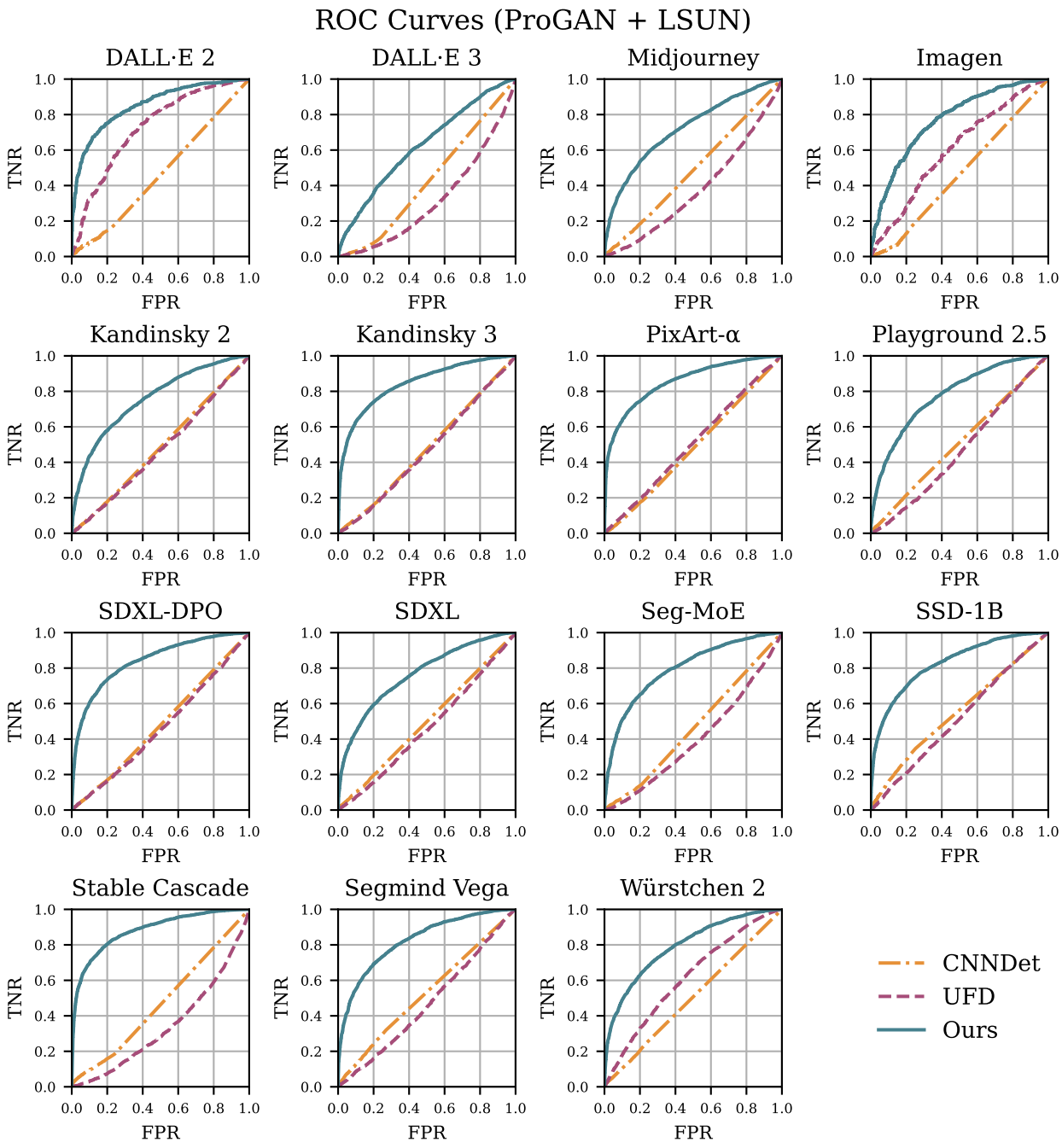


Figure D.2. ROC curves for all 15 of our SynRIS evaluation datasets. Models trained on ProGAN+LSUN.

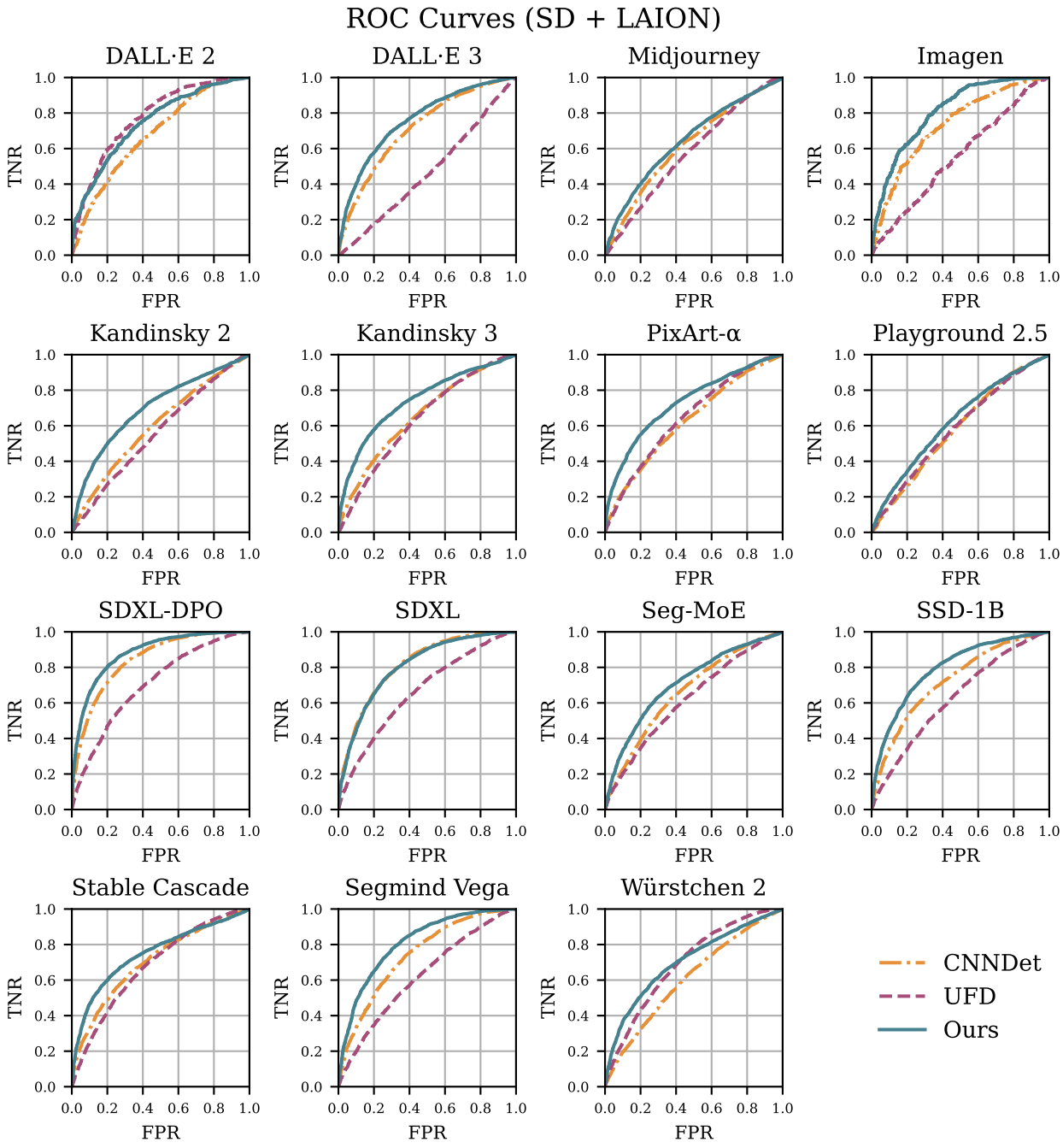


Figure D.3. ROC curves for all 15 of our SynRIS evaluation datasets. Models trained on SD+LAION.

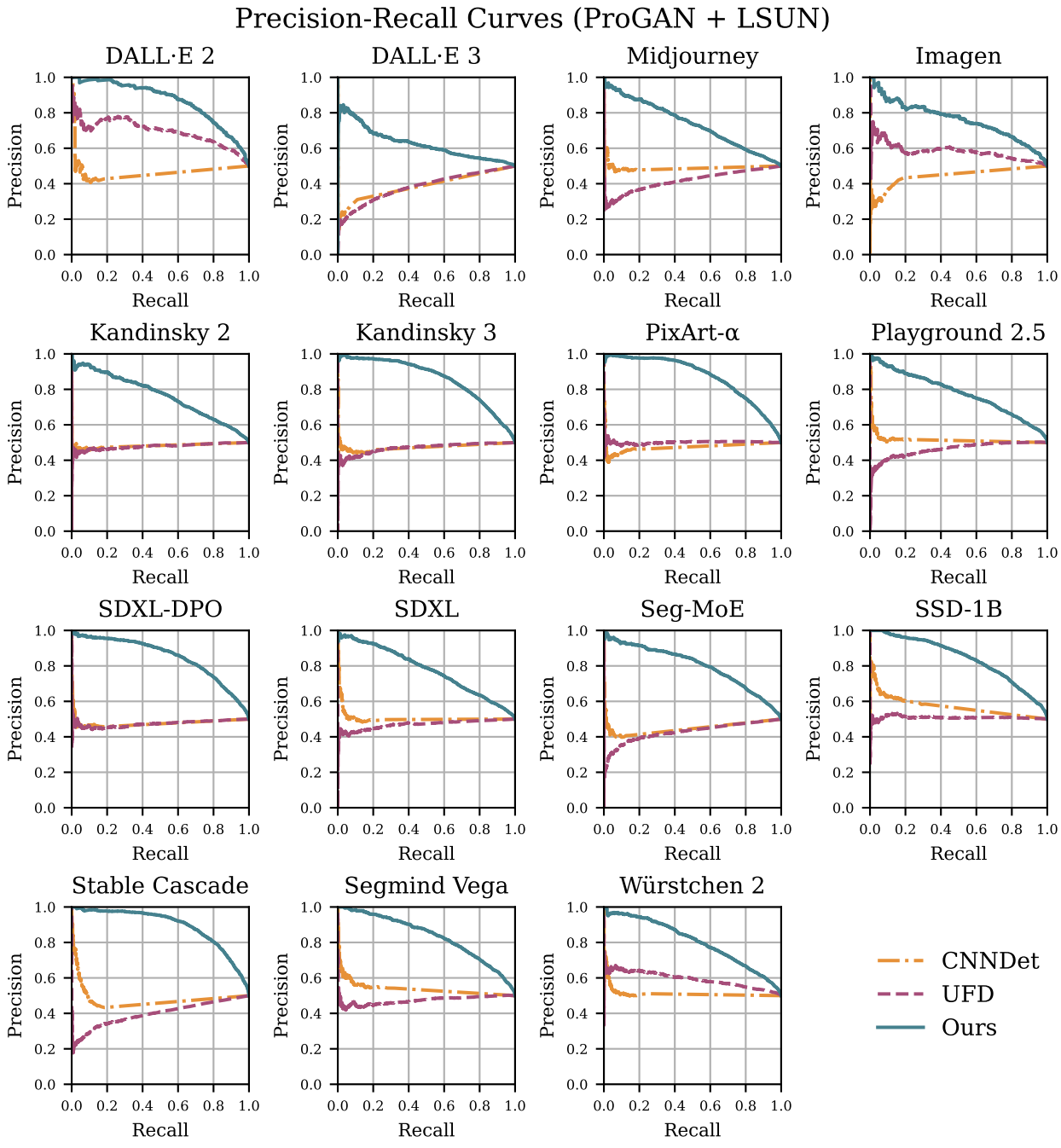


Figure D.4. Precision-Recall curves for all 15 of our SynRIS evaluation datasets. Models trained on ProGAN+LSUN.

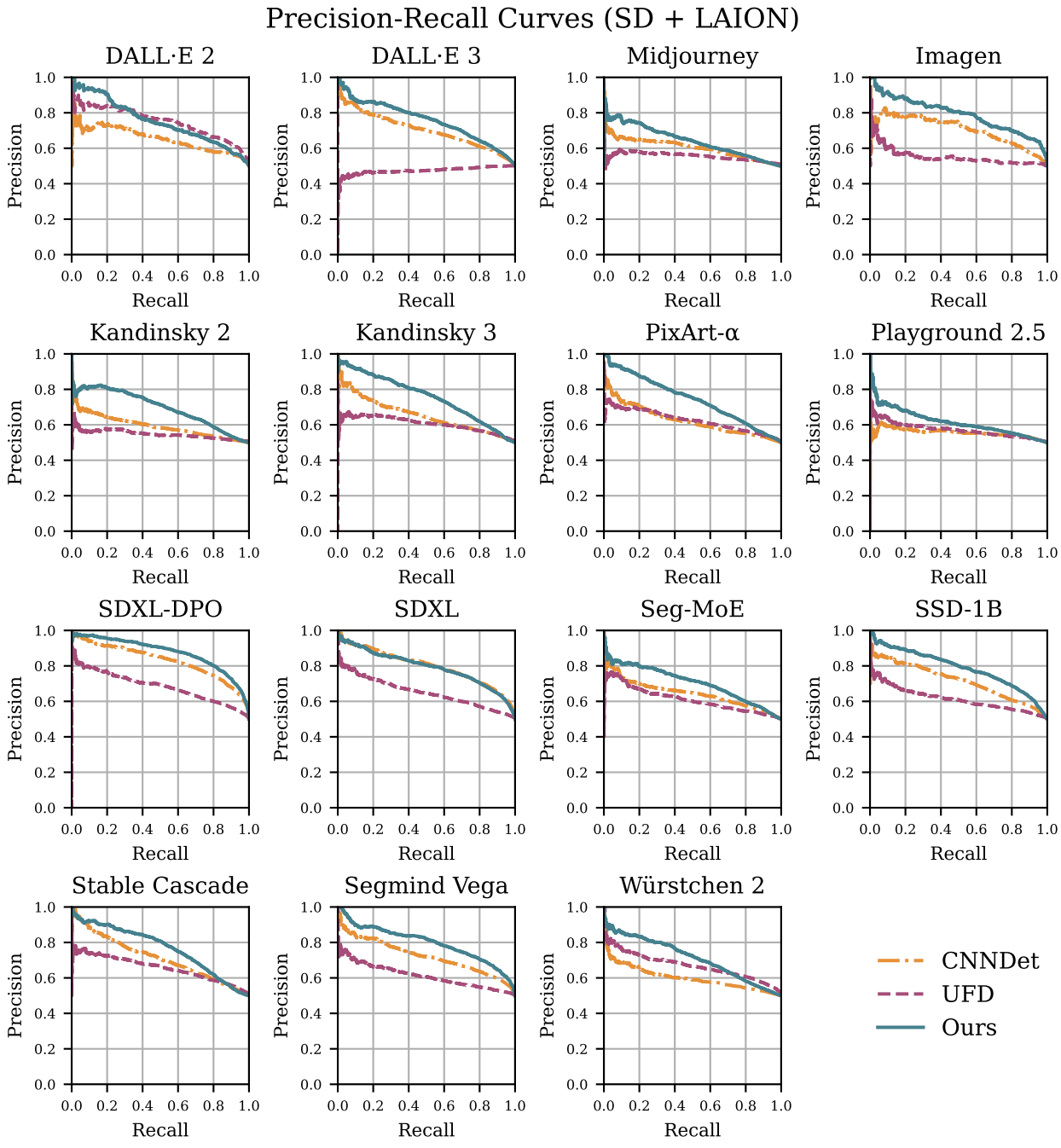


Figure D.5. Precision-Recall curves for all 15 of our SynRIS evaluation datasets. Models trained on SD+LAION.

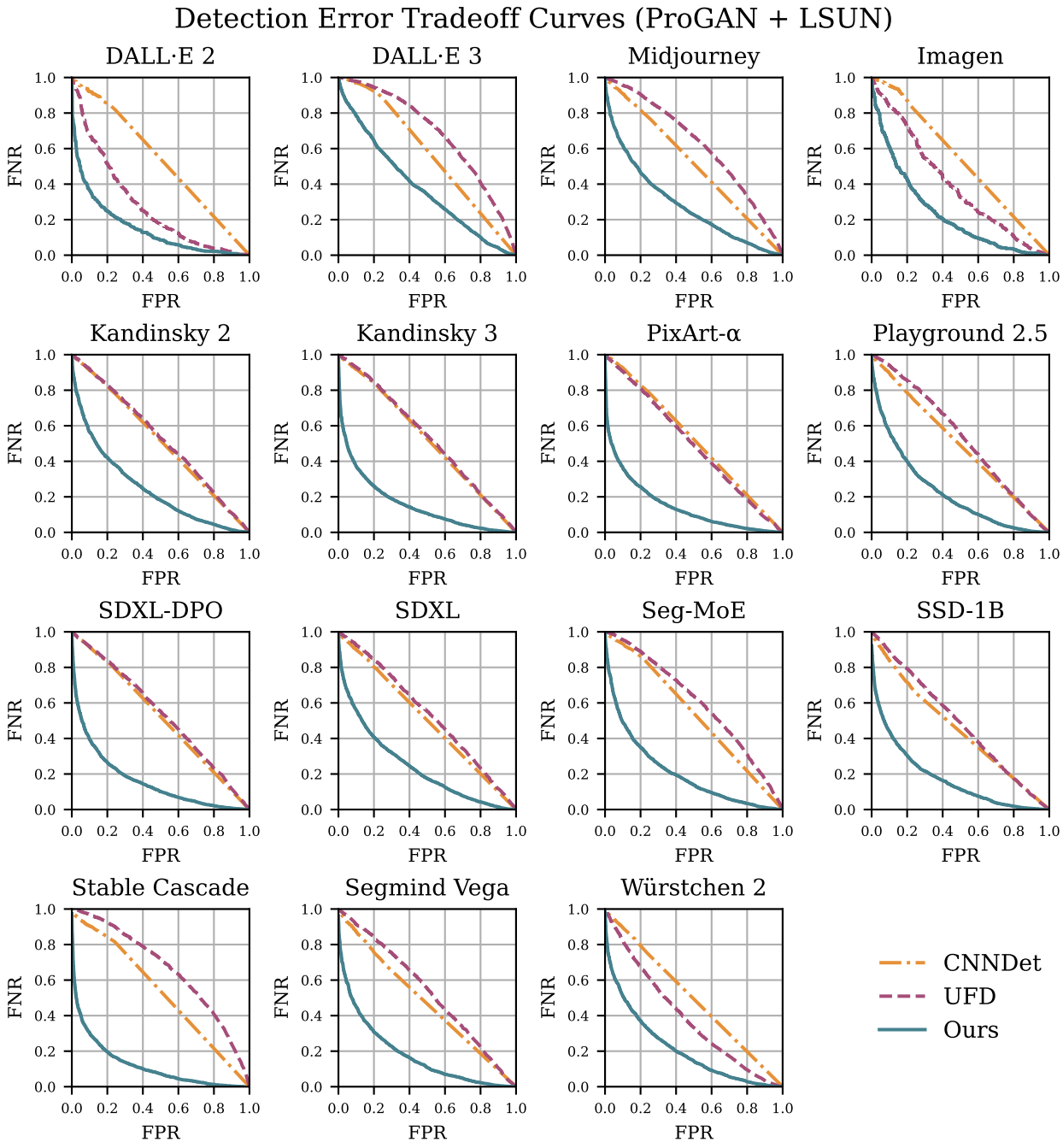


Figure D.6. Detection Error Tradeoff curves for all 15 of our SynRIS evaluation datasets. Models trained on ProGAN+LSUN.

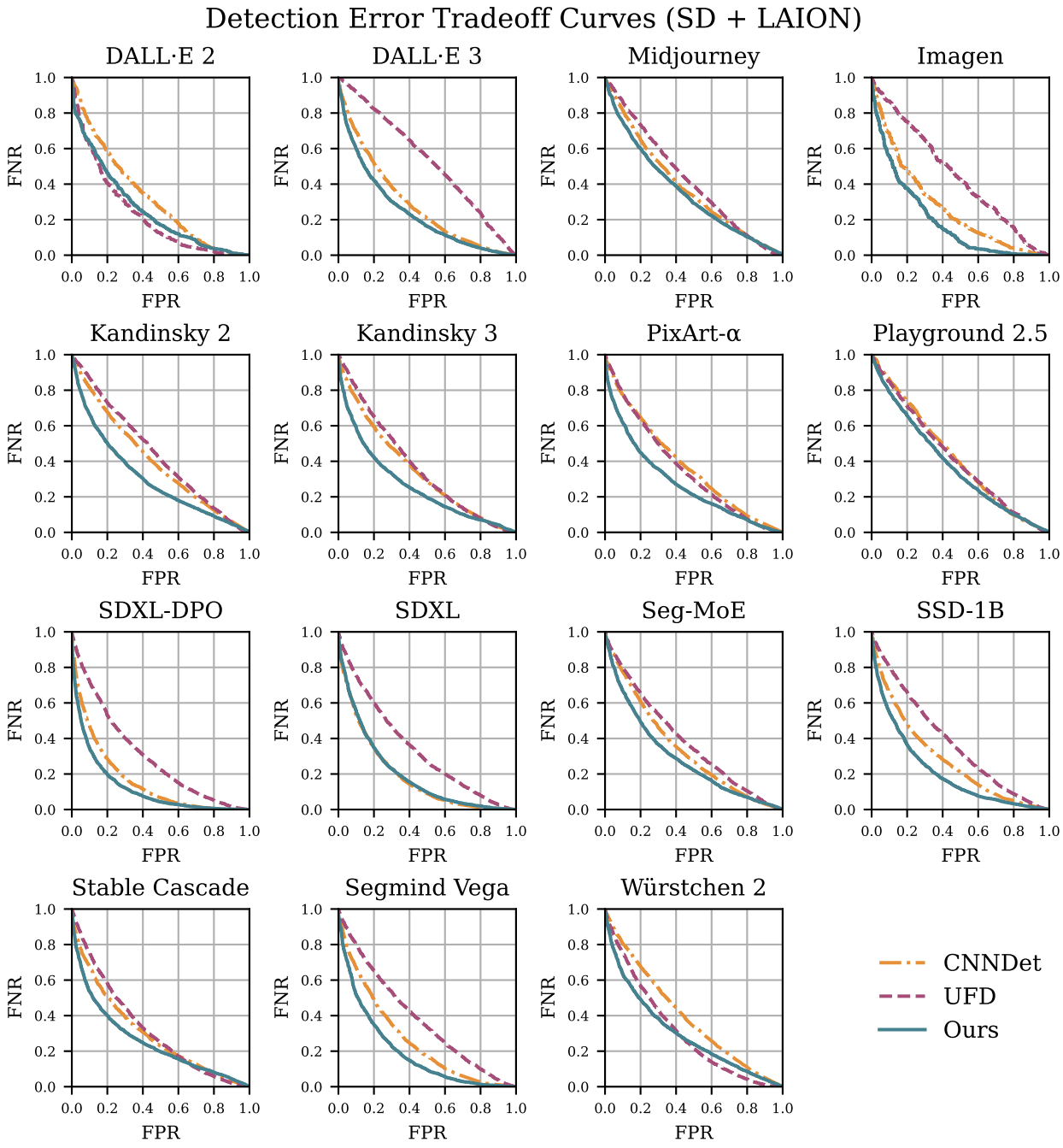
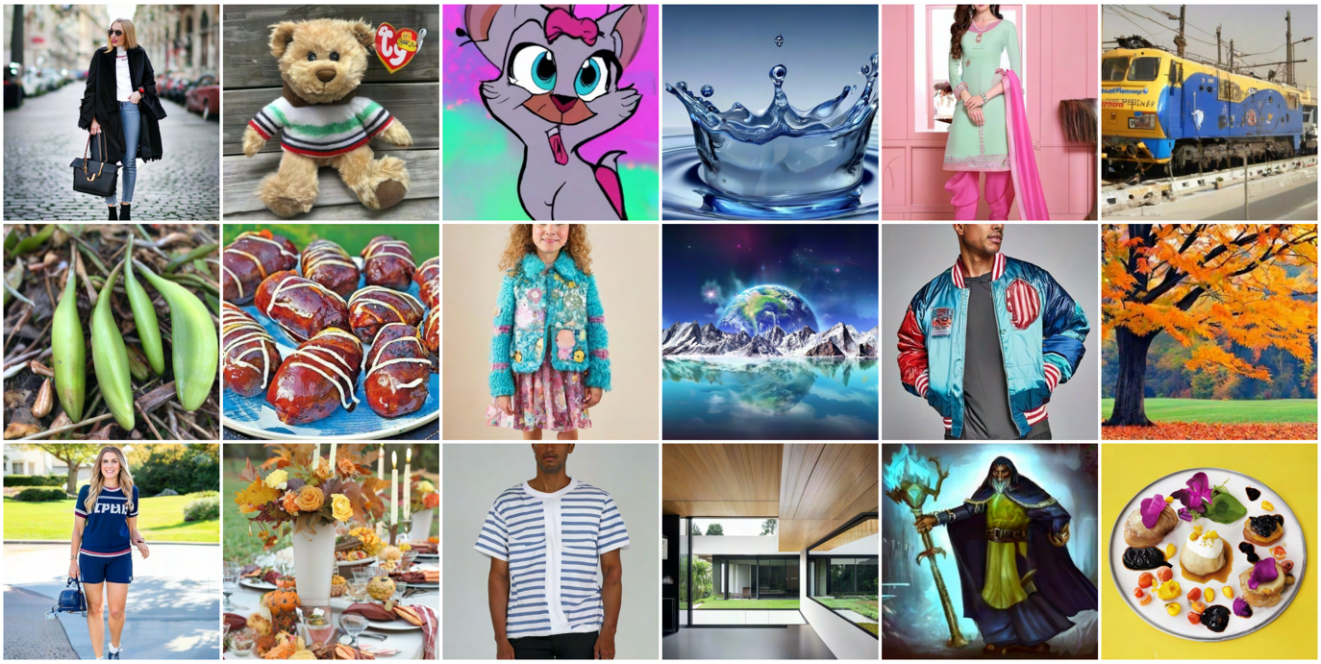


Figure D.7. Detection Error Tradeoff curves for all 15 of our SynRIS evaluation datasets. Models trained on SD+LAION.

Fake (Imagen [46])



Real (Reverse Image Search)

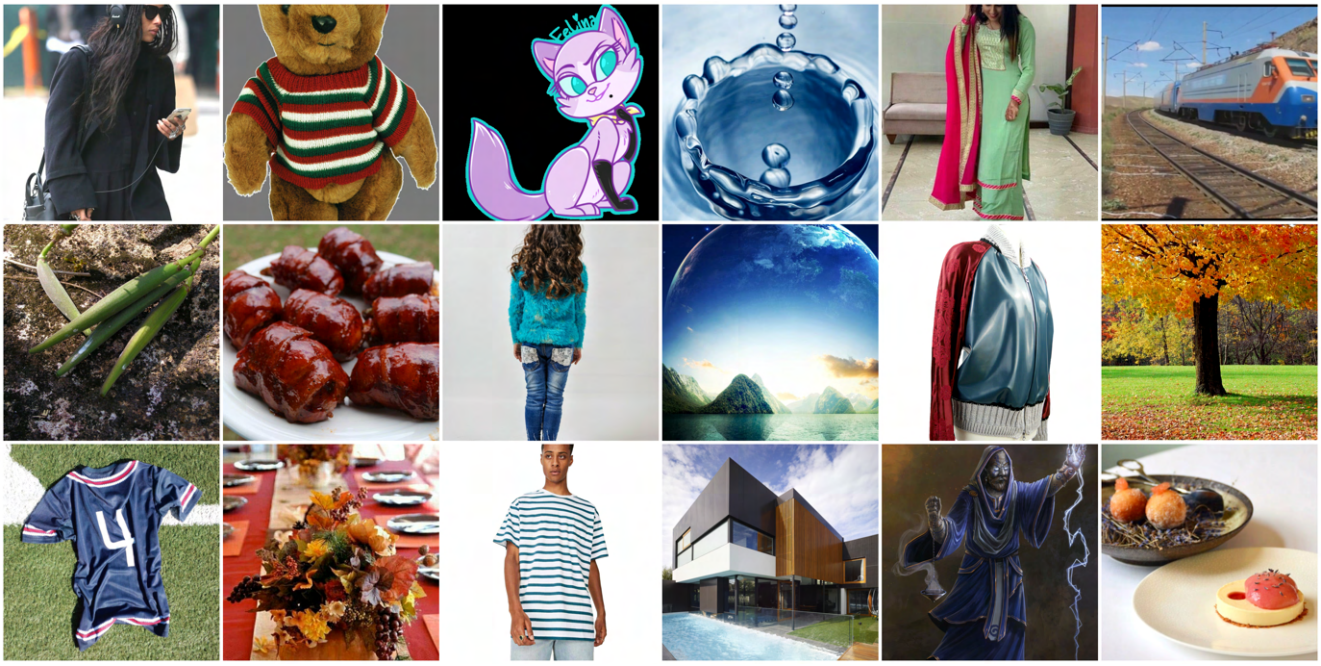
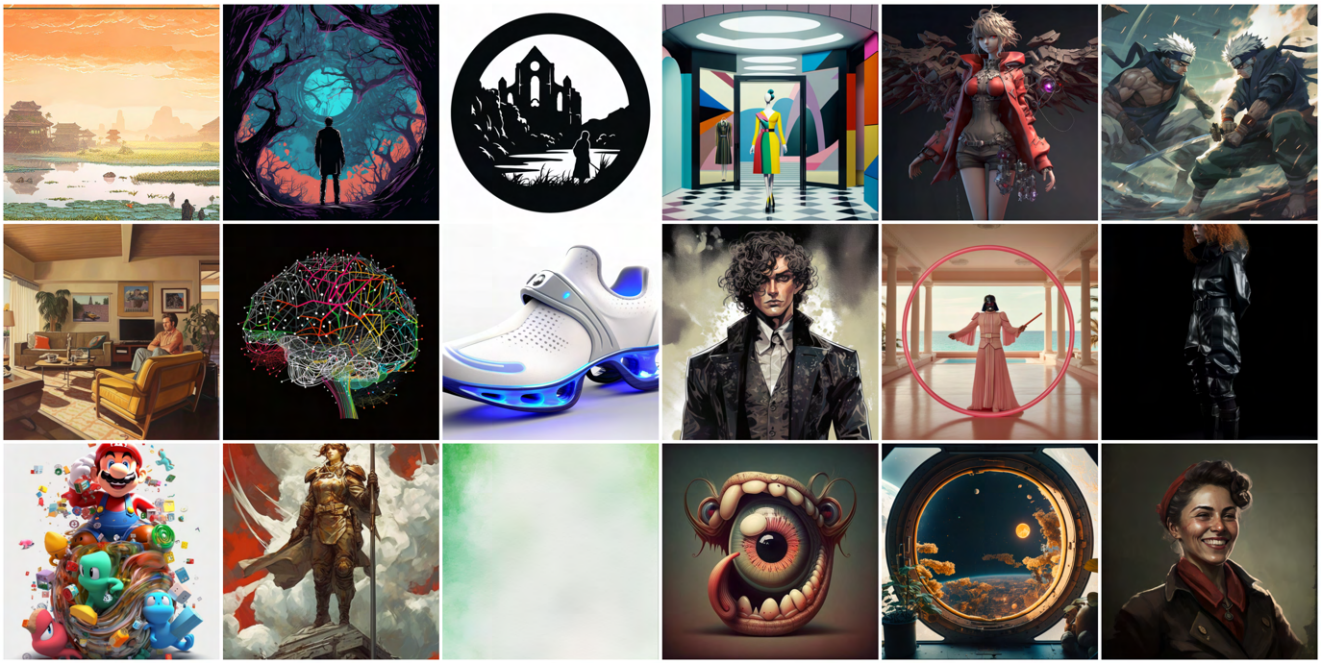


Figure F.1. **Top**: A sample of *fake* Imagen [46] images we generated for our dataset. **Bot**: Corresponding *real* images found via RIS.

Fake (Midjourney [2])



Real (Reverse Image Search)

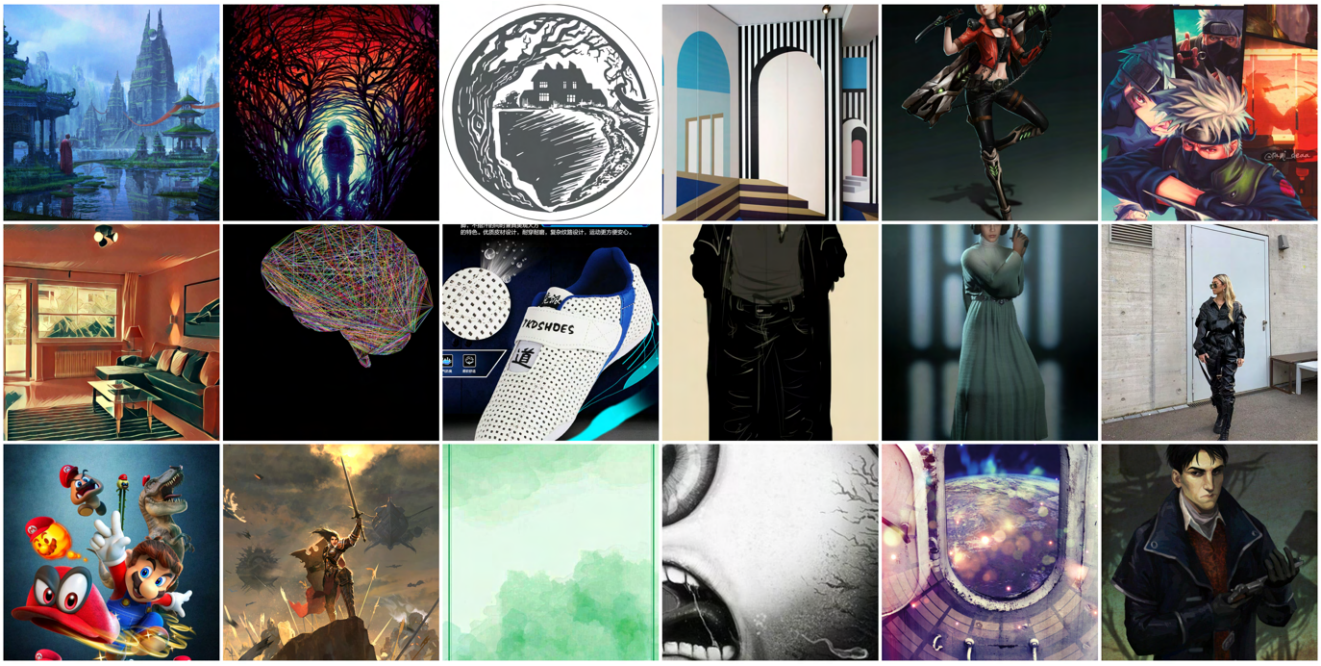


Figure F.2. **Top**: A sample of *fake* Midjourney [12] images taken from Hugging Face (link). **Bot**: Corresponding *real* images found via RIS.

Fake (DALL·E 2 [44])

Real (Reverse Image Search)

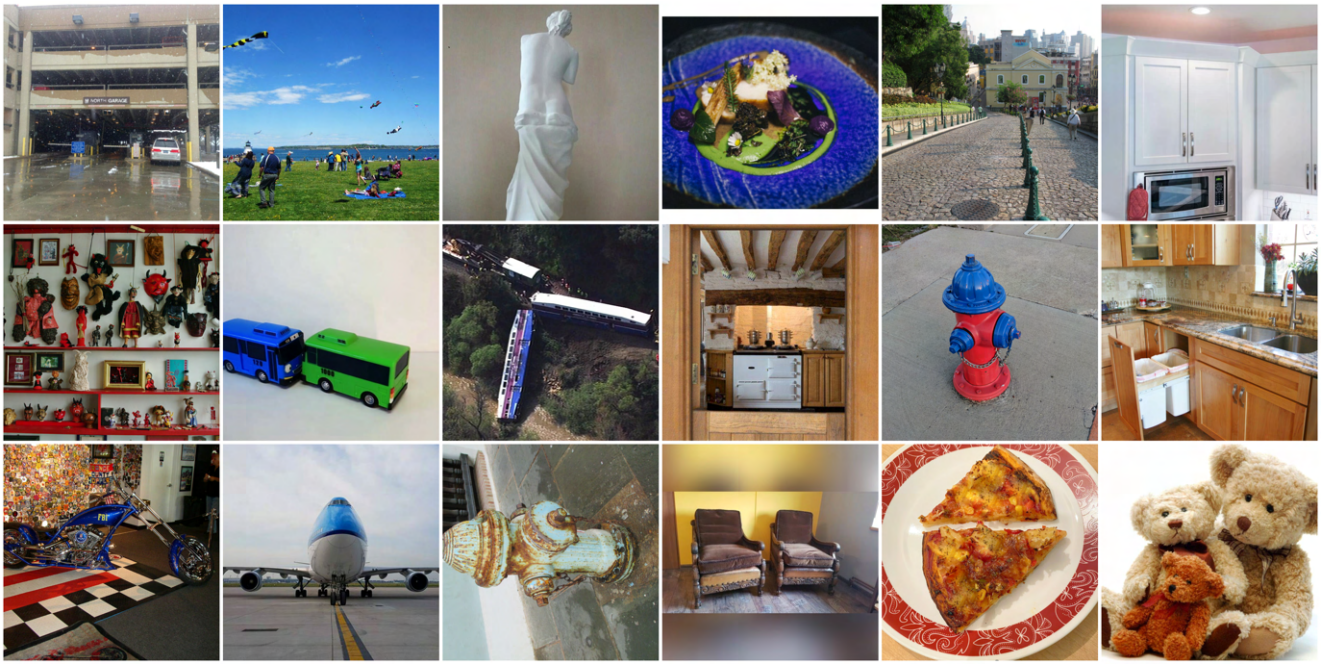
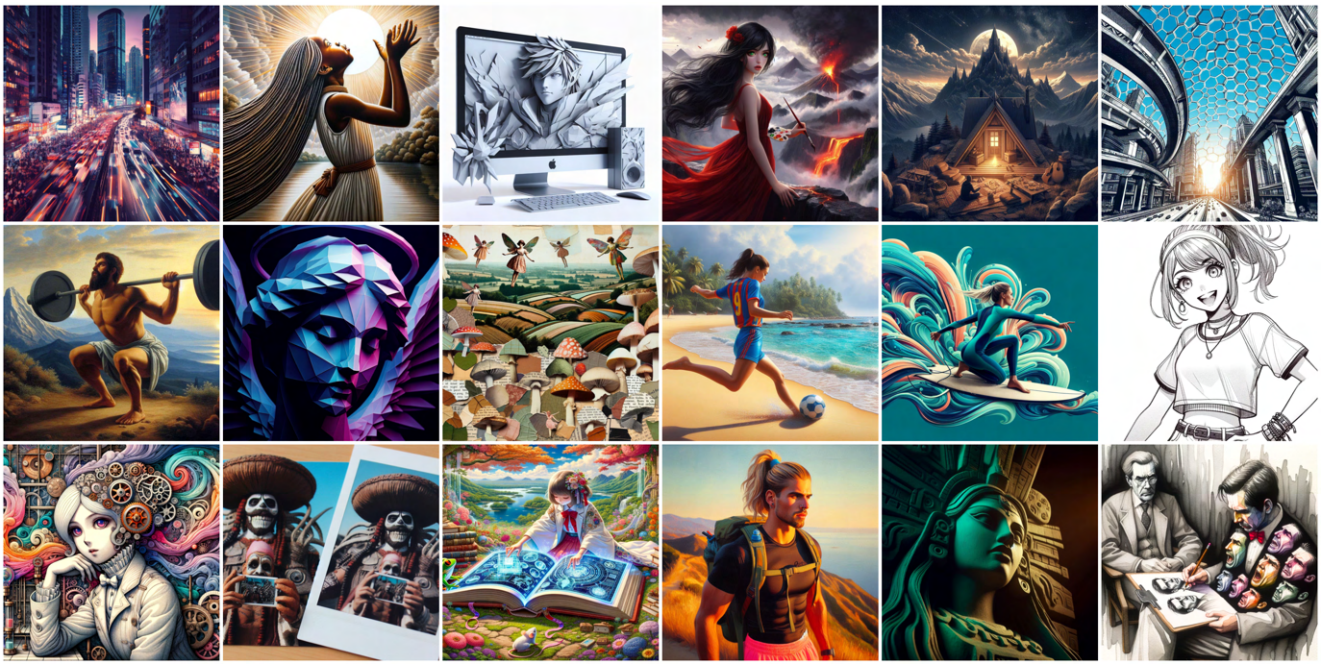


Figure F.3. **Top**: A sample of *fake* DALL·E 2 [44] images taken from the DMDetect [17] repo (link). **Bot**: Corresponding *real* images found via RIS.

Fake (DALL·E 3 [12])



Real (Reverse Image Search)

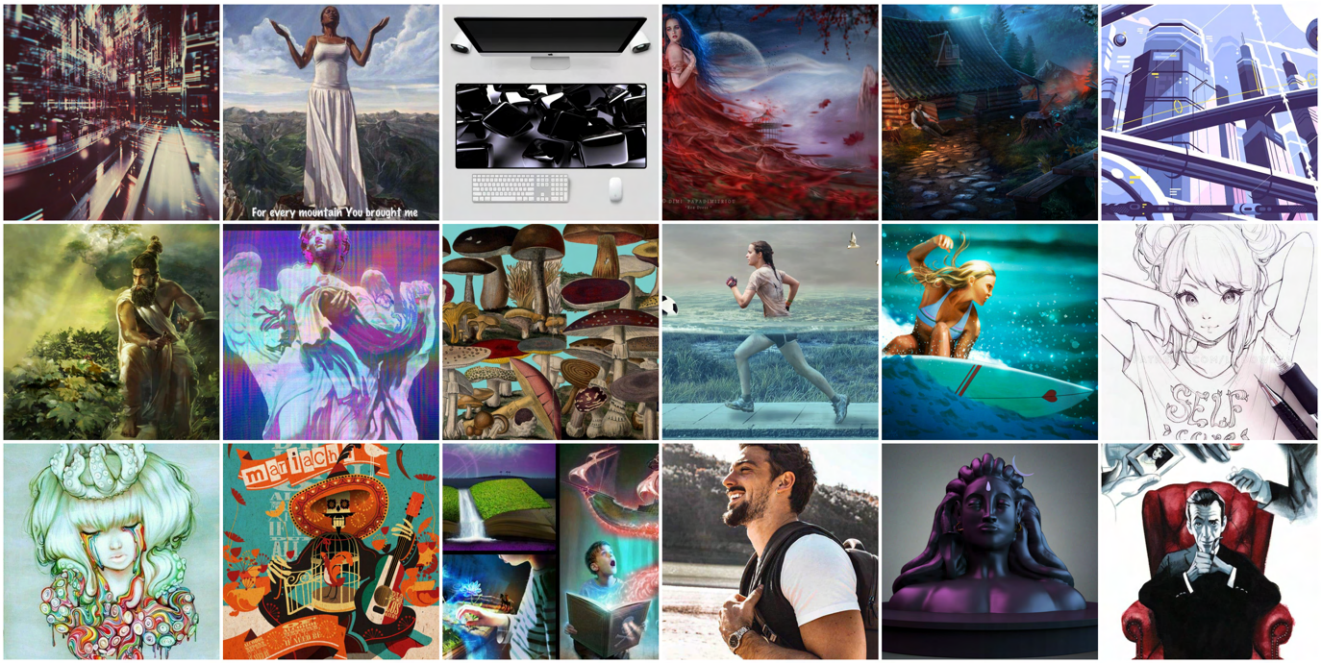


Figure F.4. **Top**: A sample of *fake* DALL·E 3 [12] images taken from Hugging Face (link). **Bot**: Corresponding *real* images found via RIS.

Fake (Kandinsky 2 [51])

Real (Reverse Image Search)

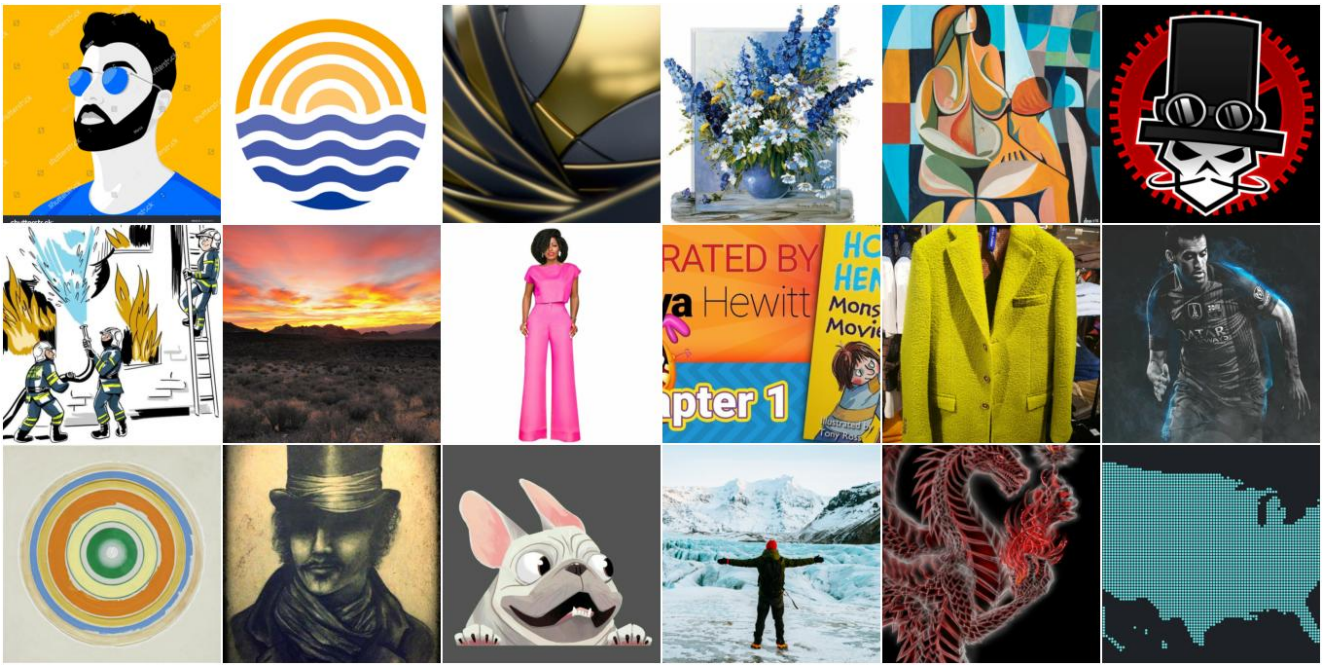
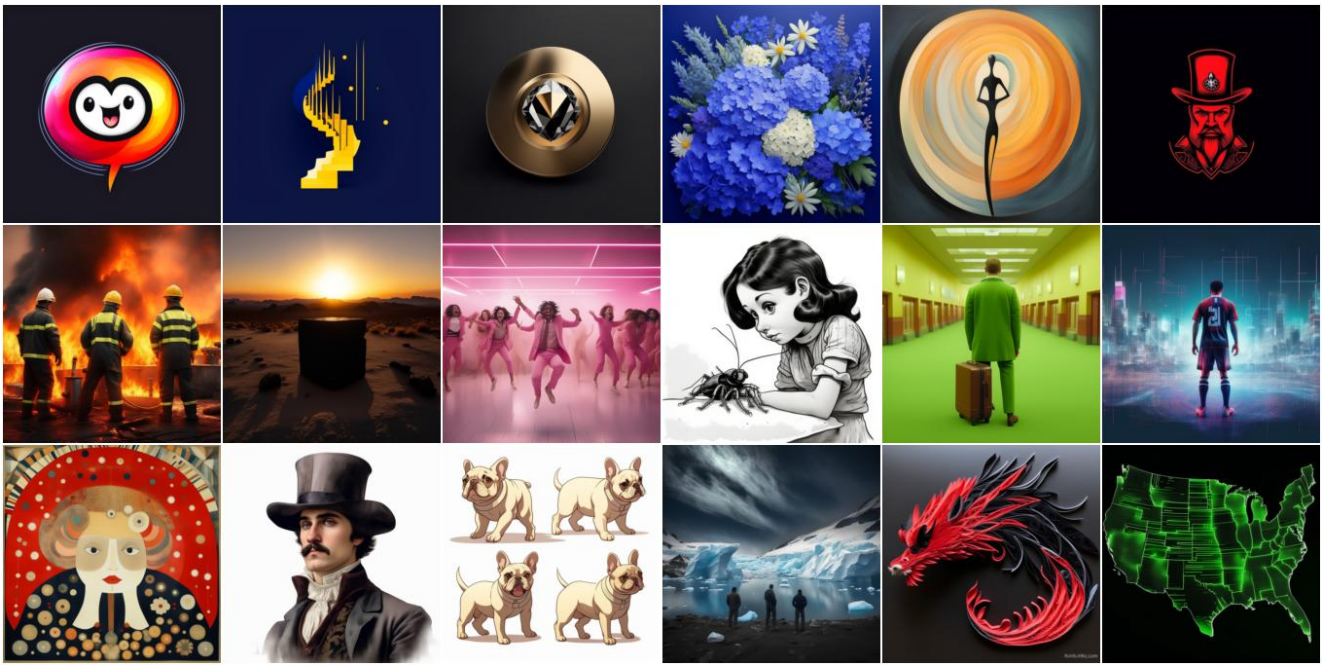


Figure F.5. **Top**: A sample of *fake* Kandinsky 2 [51] images we generated for our dataset. **Bot**: Corresponding *real* images found via RIS.

Fake (Kandinsky 3 [10])



Real (Reverse Image Search)

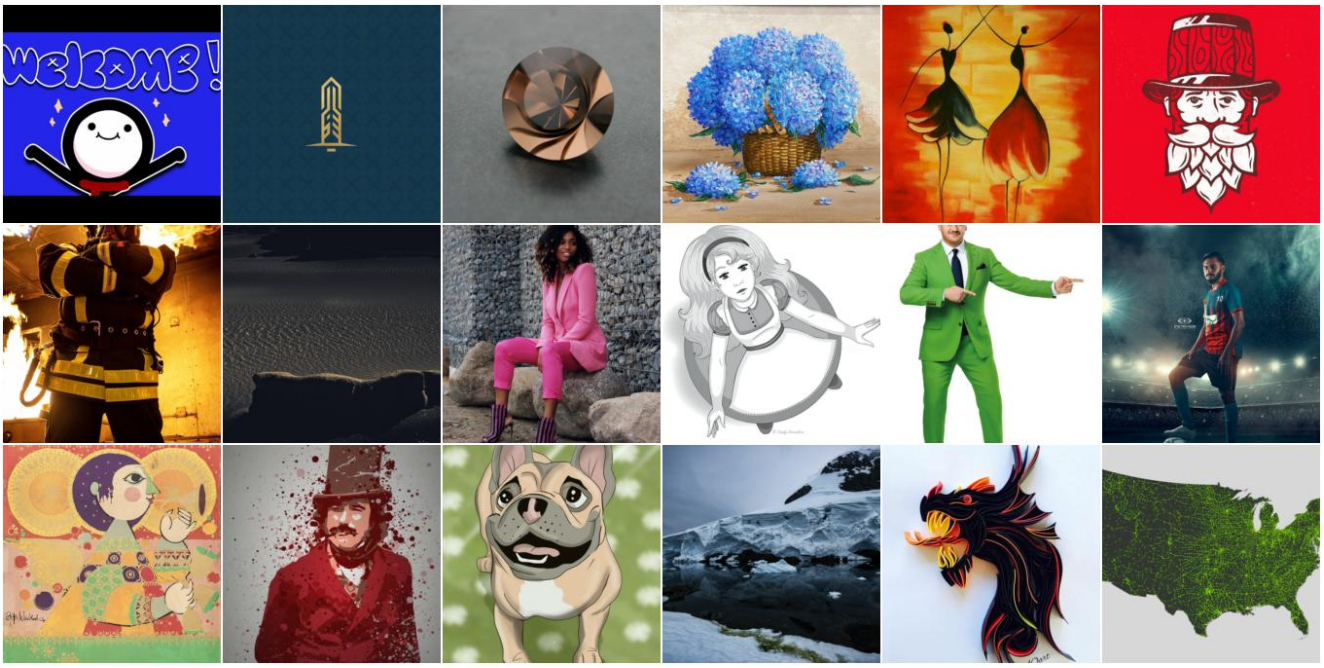


Figure F.6. **Top**: A sample of *fake* Kandinsky 3 [10] images we generated for our dataset. **Bot**: Corresponding *real* images found via RIS.

Fake (PixArt-α [16])

Real (Reverse Image Search)

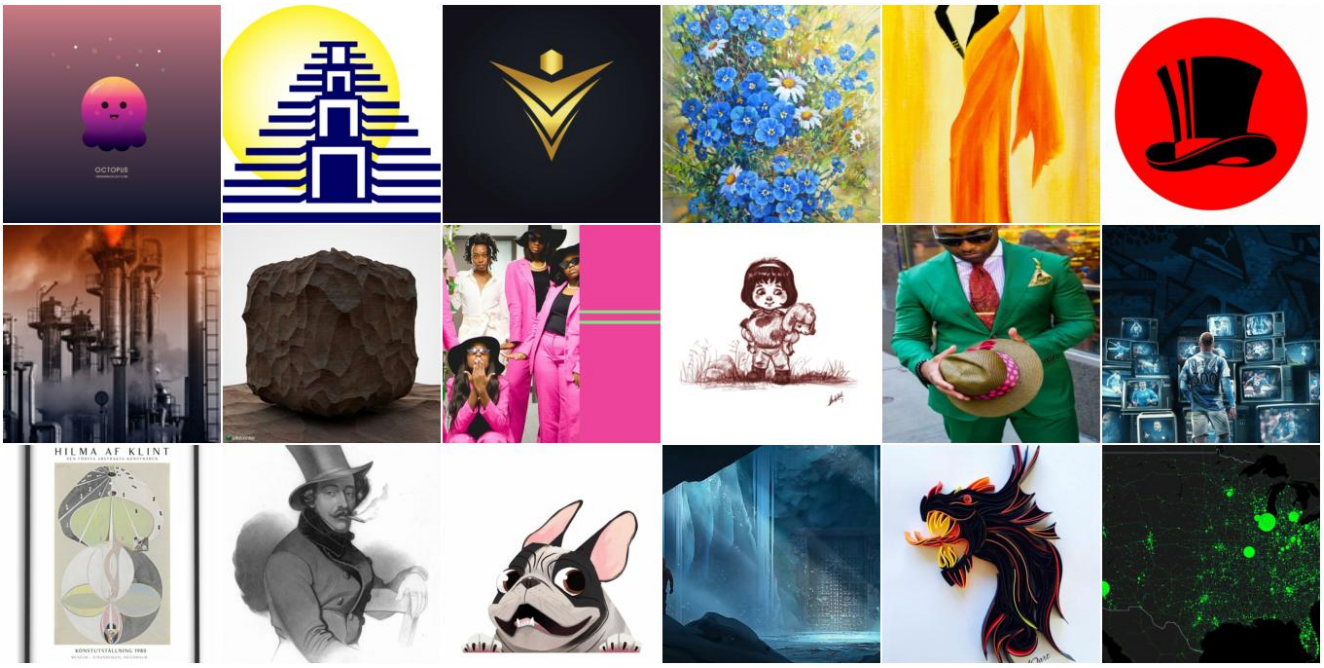
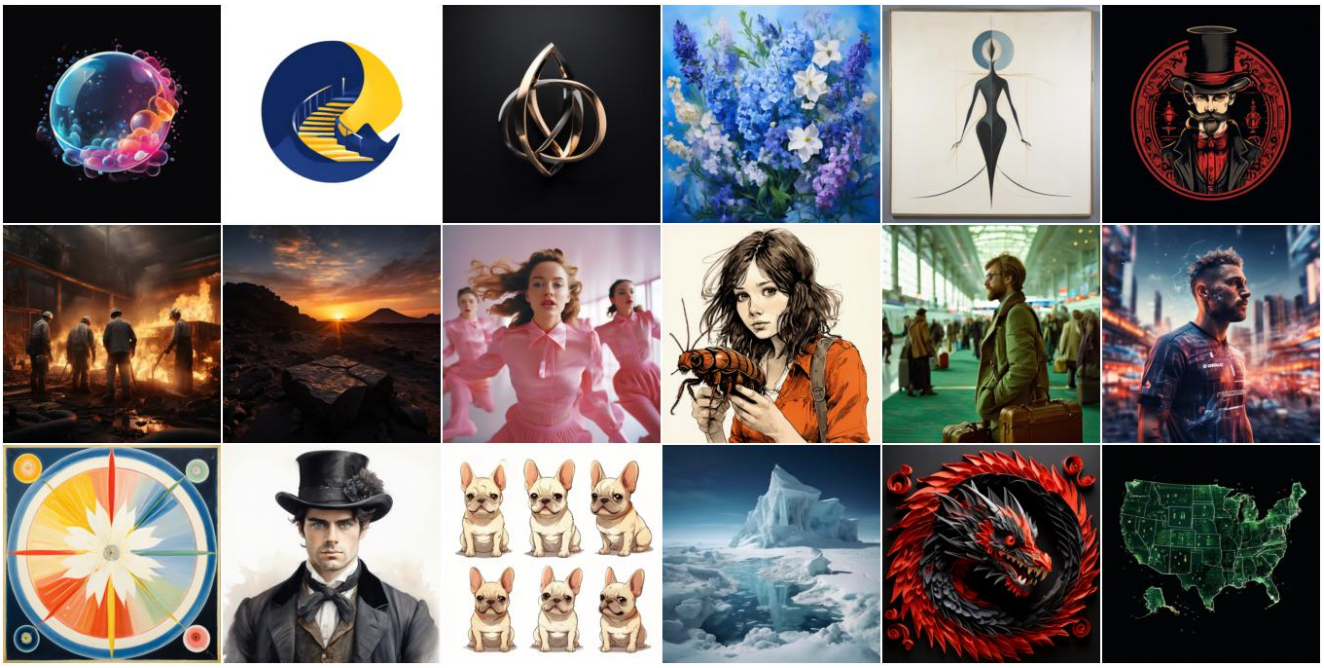


Figure F.7. **Top**: A sample of *fake* PixArt-α [16] images we generated for our dataset. **Bot**: Corresponding *real* images found via RIS.

Fake (Playground 2.5 [31])



Real (Reverse Image Search)

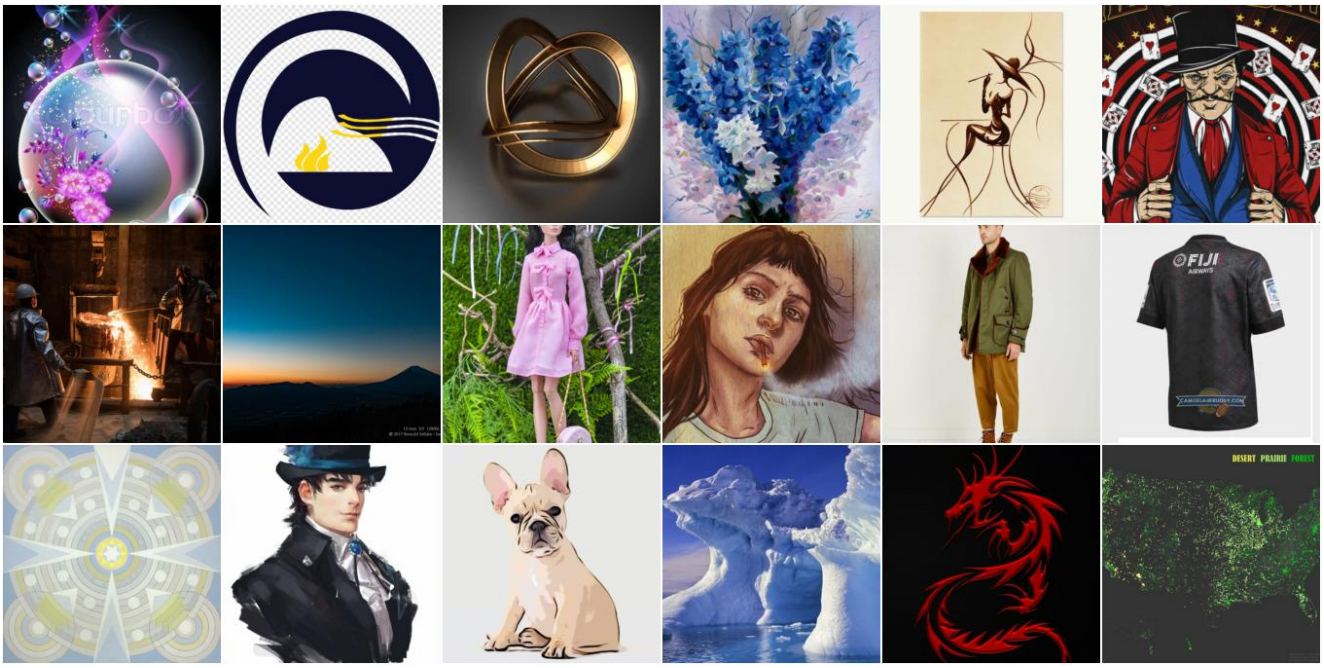
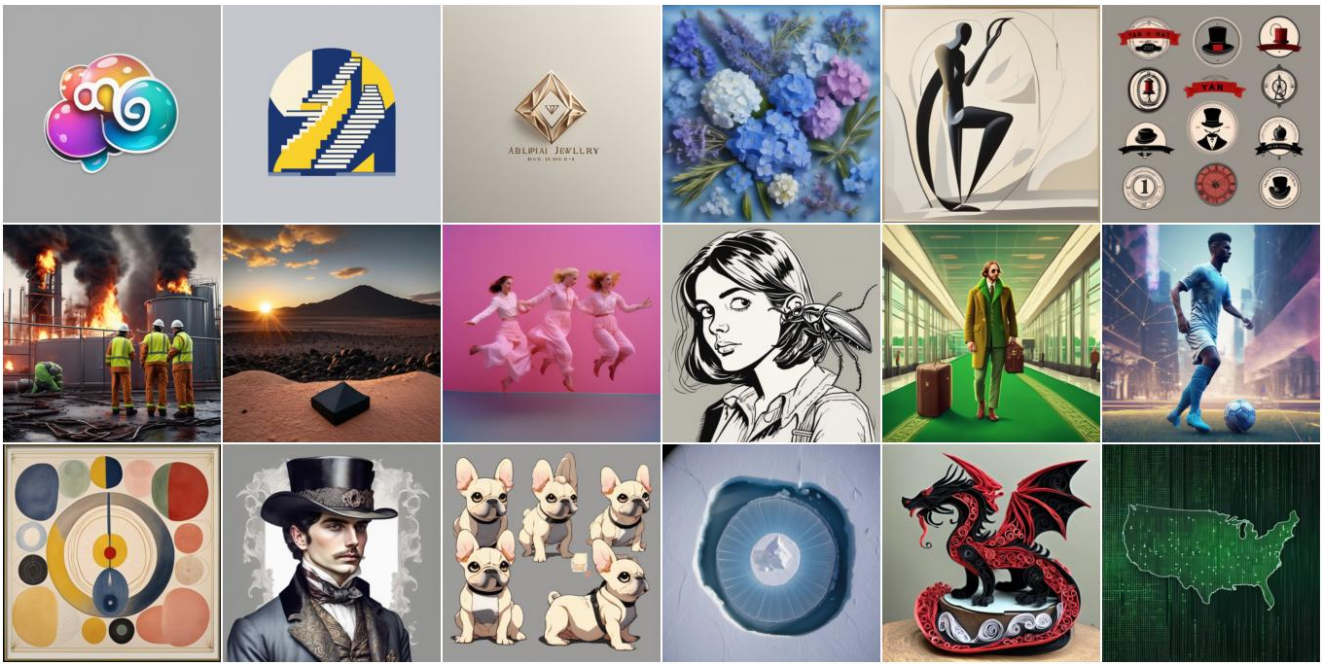


Figure F.8. **Top**: A sample of *fake* Playground [31] images we generated for our dataset. **Bot**: Corresponding *real* images found via RIS.

Fake (SDXL-DPO [53])



Real (Reverse Image Search)

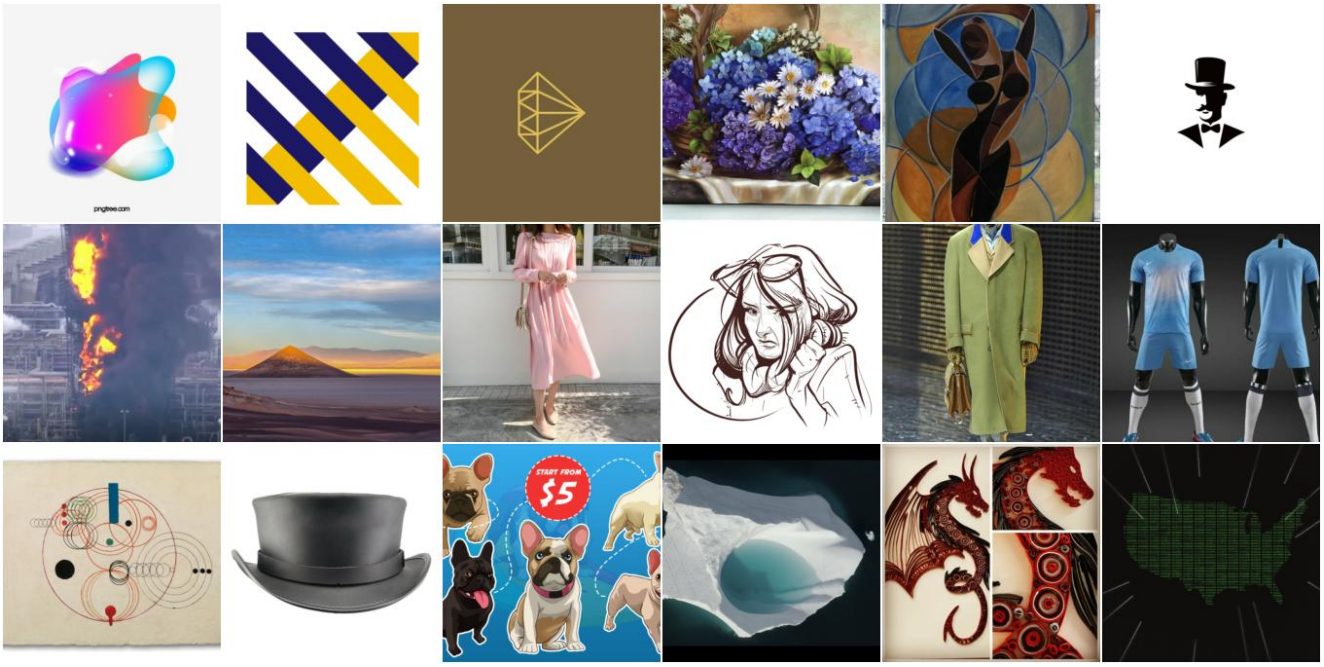
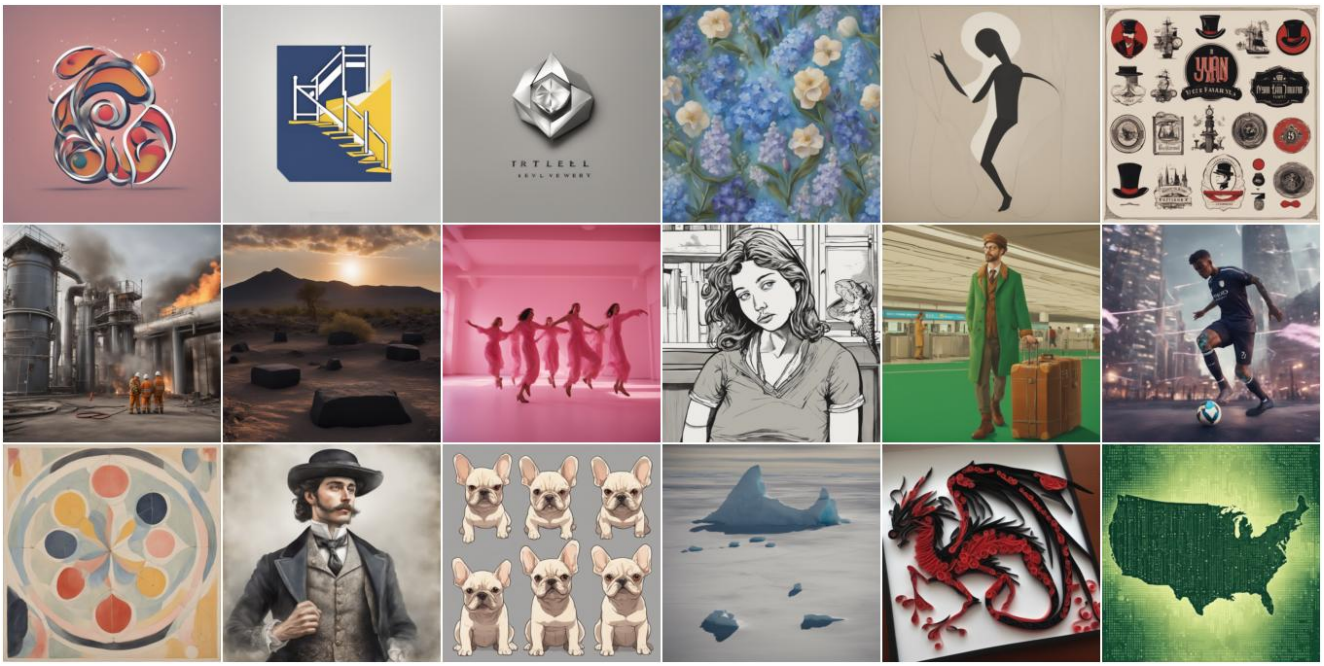


Figure F.9. **Top**: A sample of *fake* SDXL-DPO [53] images we generated for our dataset. **Bot**: Corresponding *real* images found via RIS.

Fake (SDXL [41])



Real (Reverse Image Search)

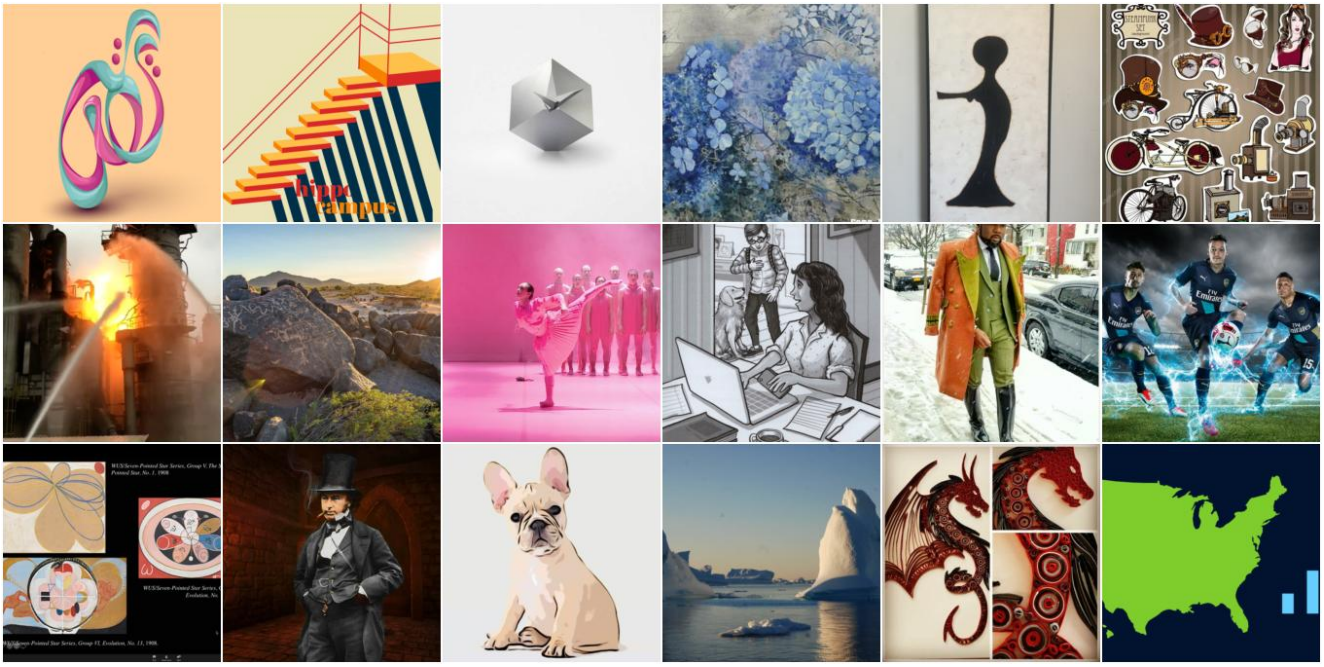
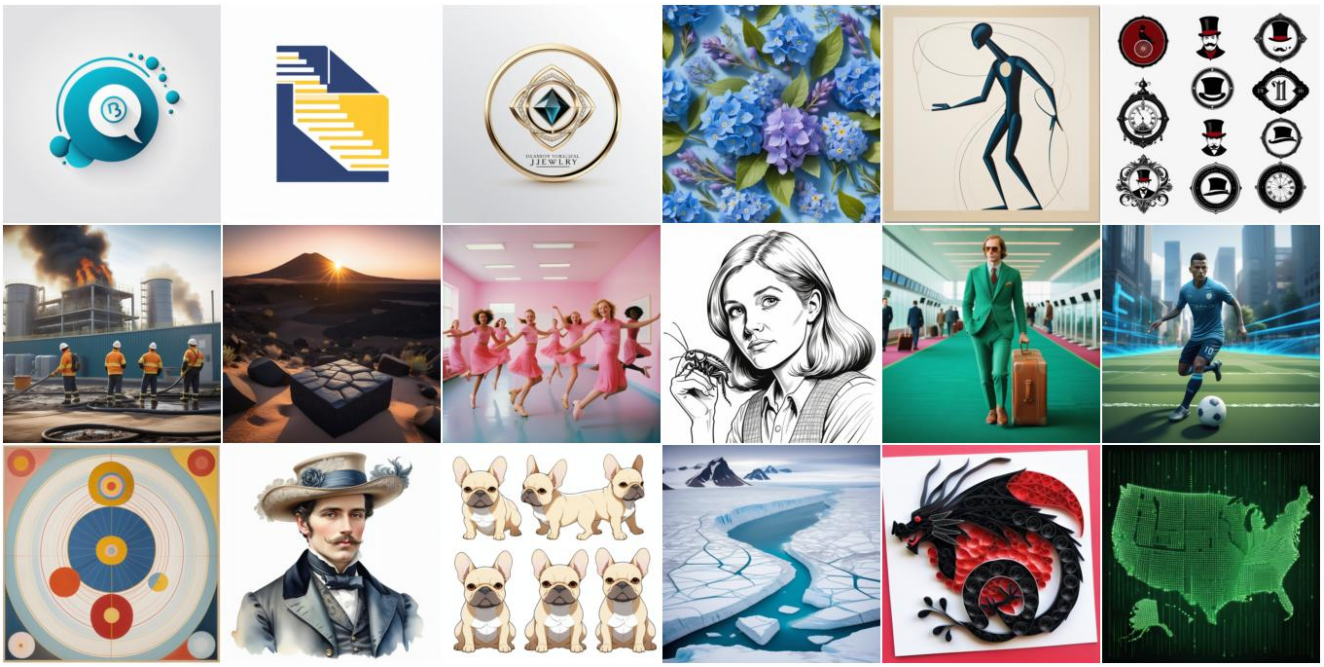


Figure F.10. **Top**: A sample of *fake* SDXL [41] images we generated for our dataset. **Bot**: Corresponding *real* images found via RIS.

Fake (Seg-MoE [58])



Real (Reverse Image Search)

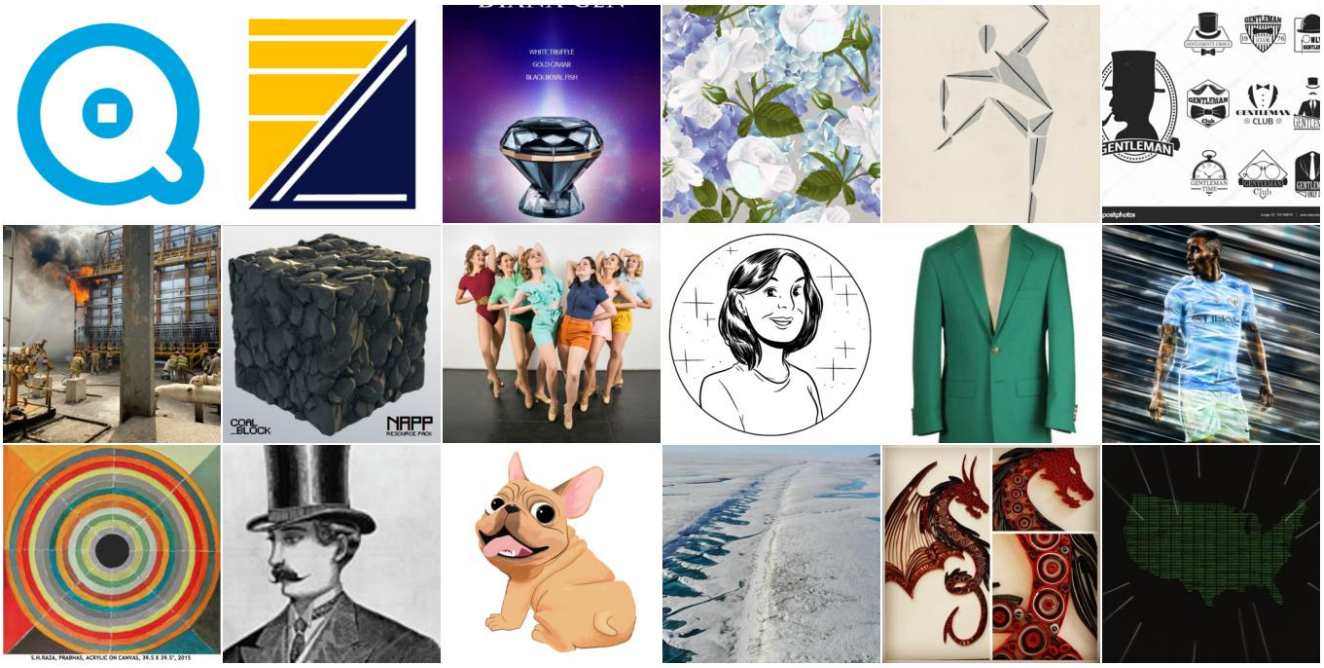
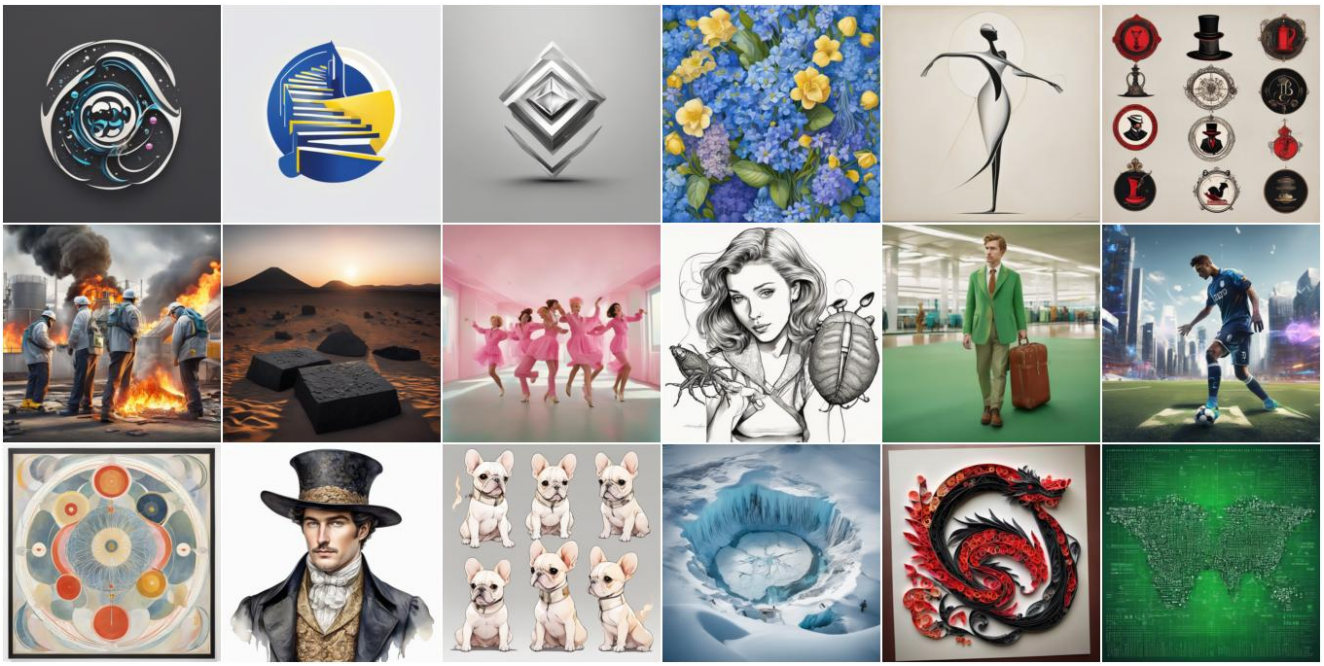


Figure F.11. **Top**: A sample of *fake* Seg-MoE [58] images we generated for our dataset. **Bot**: Corresponding *real* images found via RIS.

Fake (SSD-1B [21])



Real (Reverse Image Search)

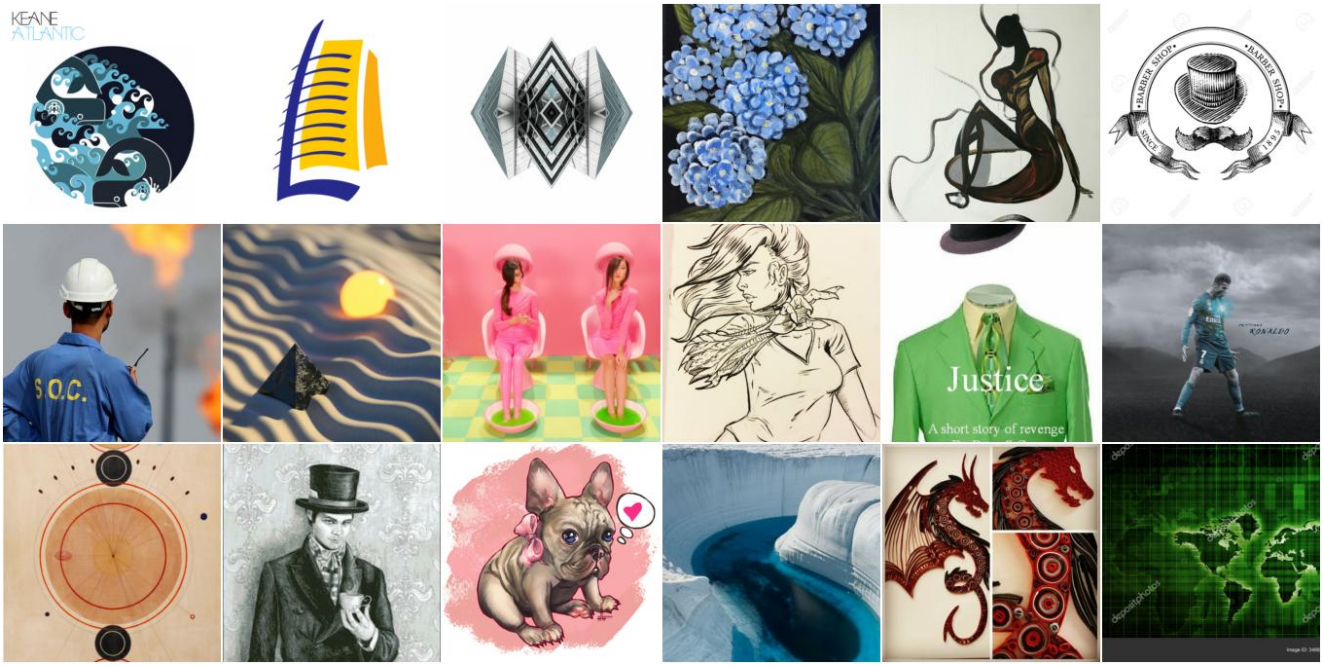
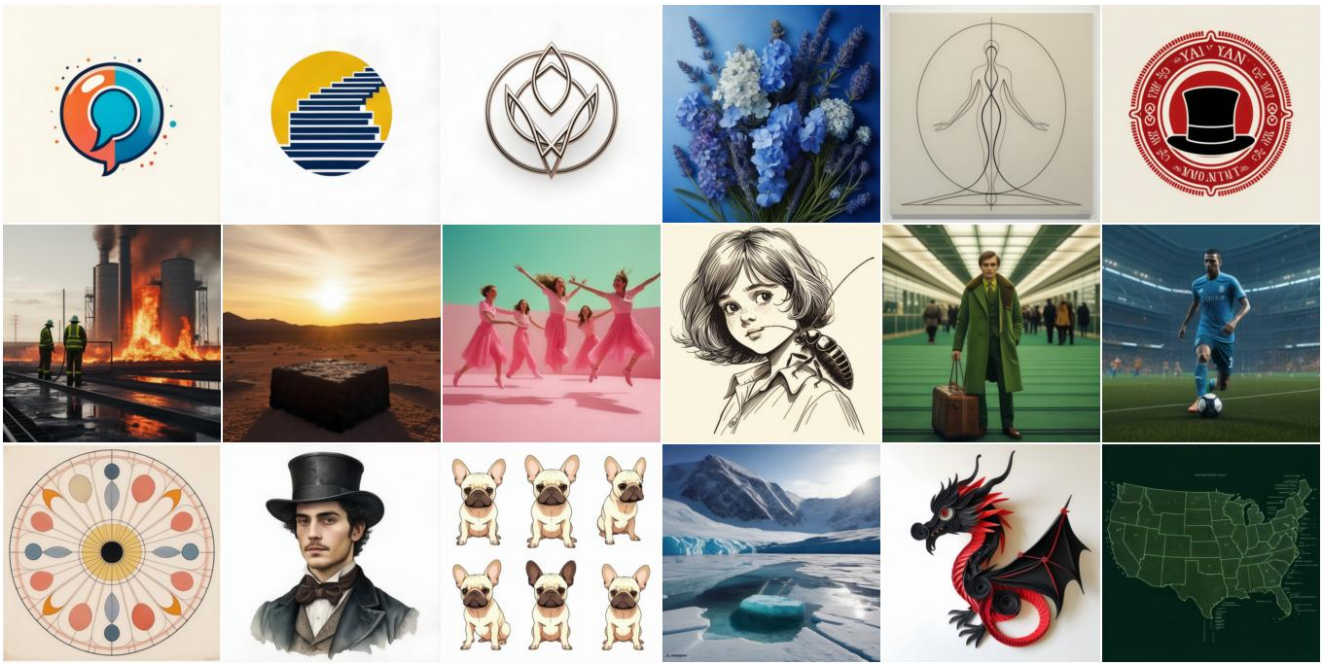


Figure F.12. **Top**: A sample of *fake* SSD-1B [21] images we generated for our dataset. **Bot**: Corresponding *real* images found via RIS.

Fake (Stable-Cascade [39])



Real (Reverse Image Search)

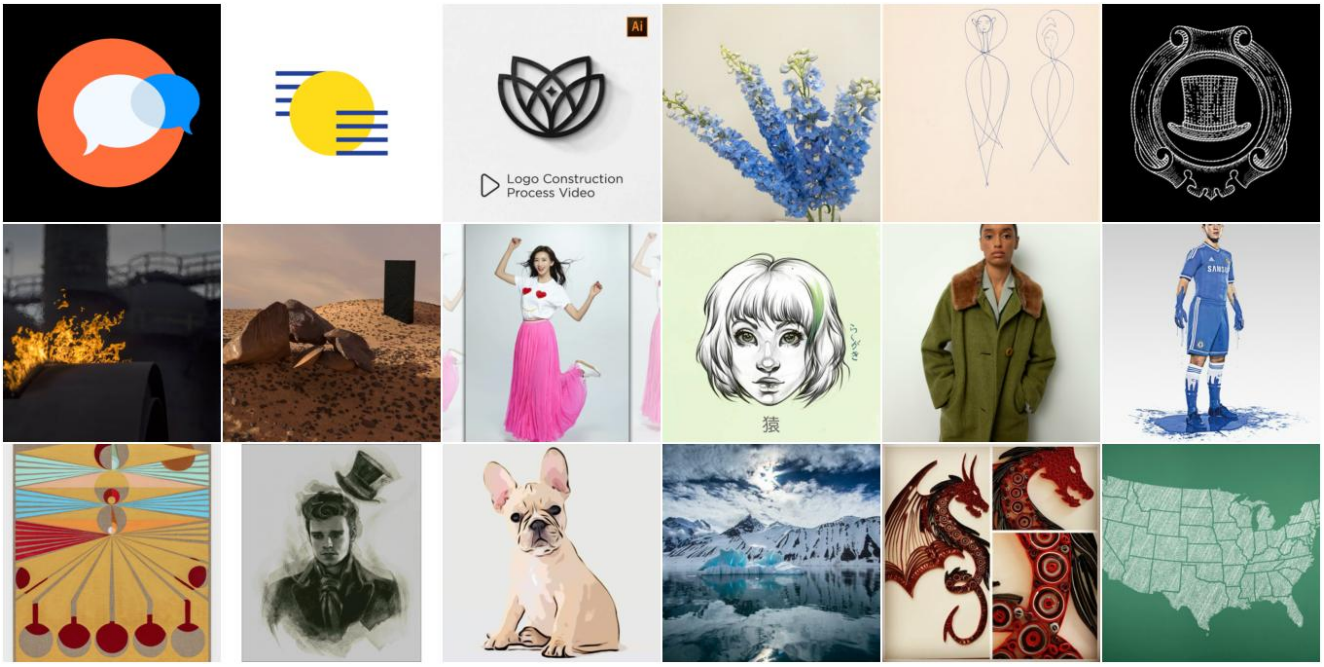
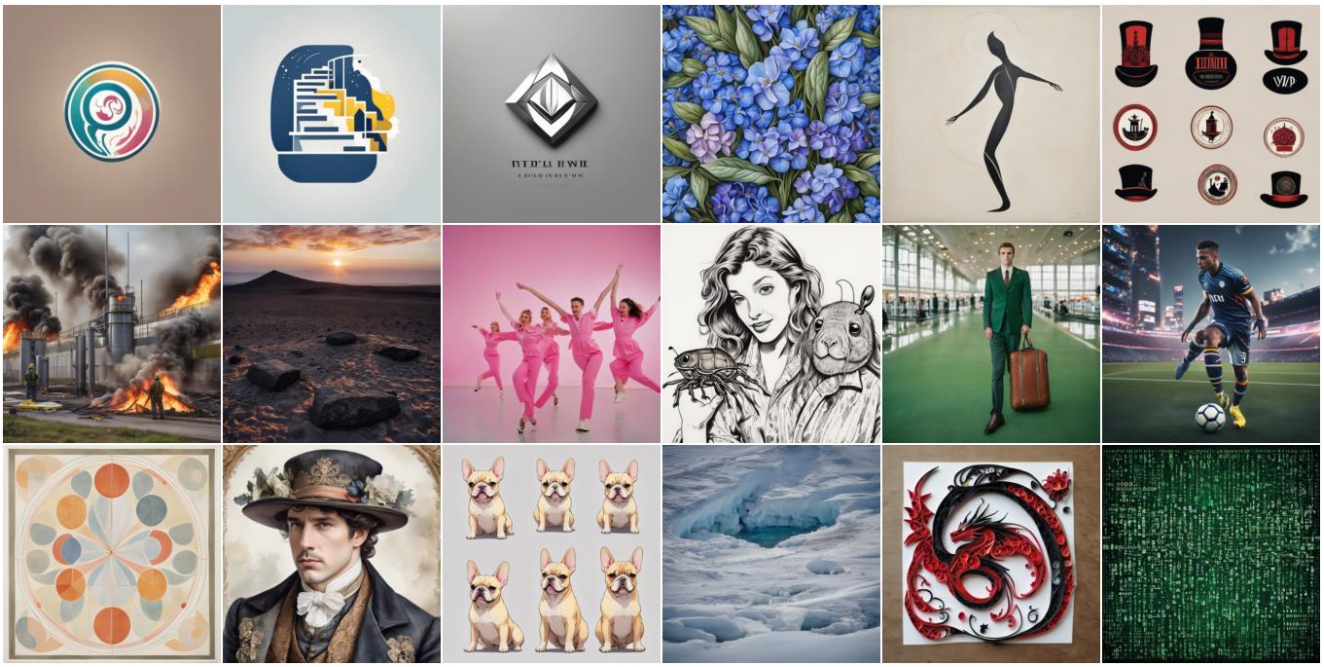


Figure F.13. **Top**: A sample of *fake* Stable-Cascade [39] images we generated for our dataset. **Bot**: Corresponding *real* images found via RIS.

Fake (Segmind Vega [21])



Real (Reverse Image Search)

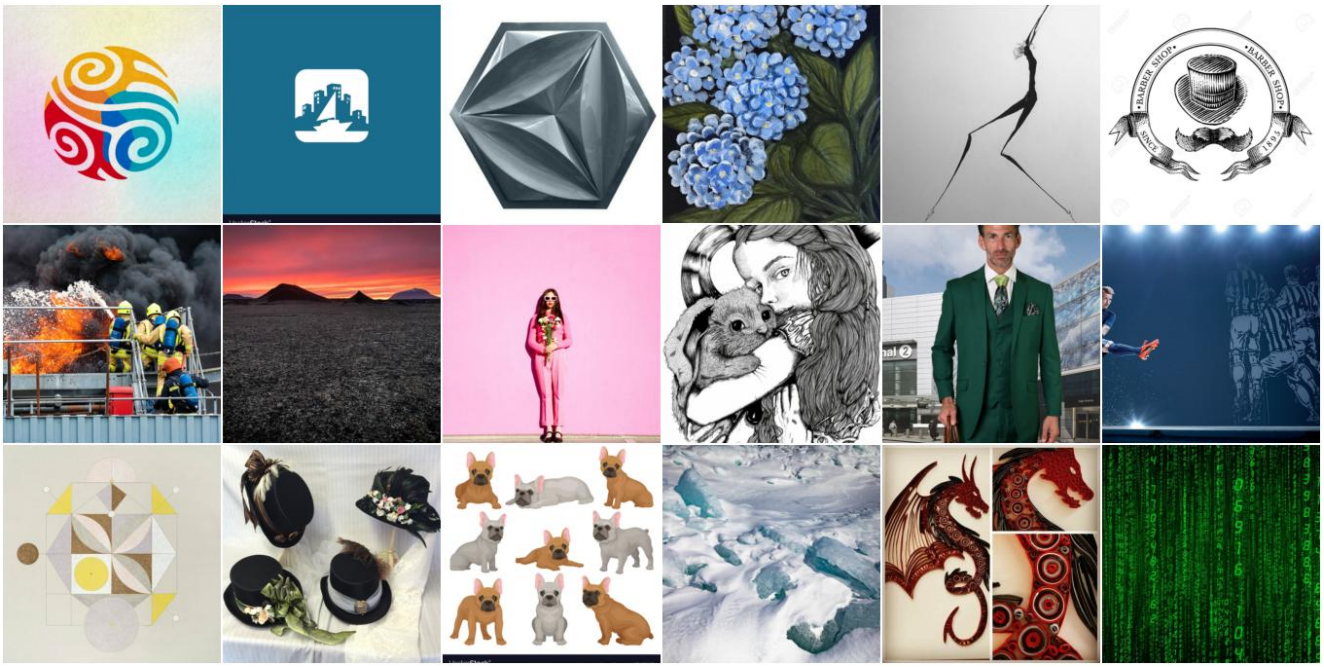
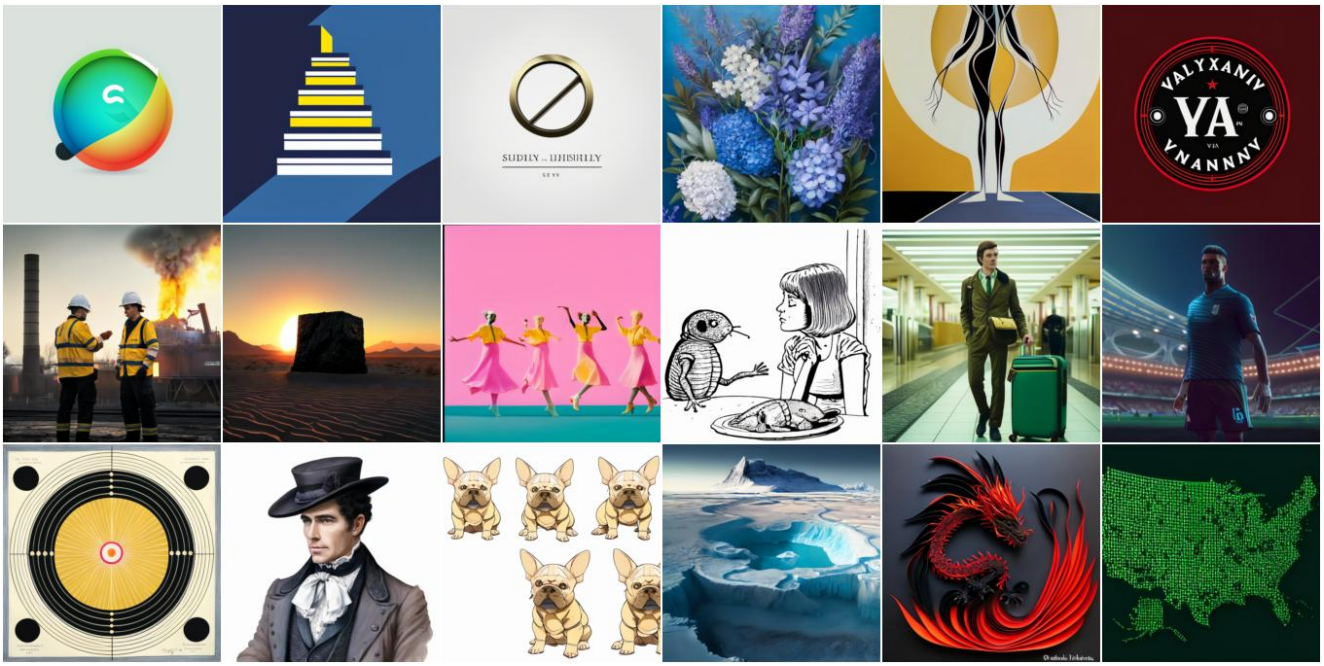


Figure F.14. **Top**: A sample of *fake* Segmind Vega [21] images we generated for our dataset. **Bot**: Corresponding *real* images found via RIS.

Fake (Würstchen 2 [39])



Real (Reverse Image Search)

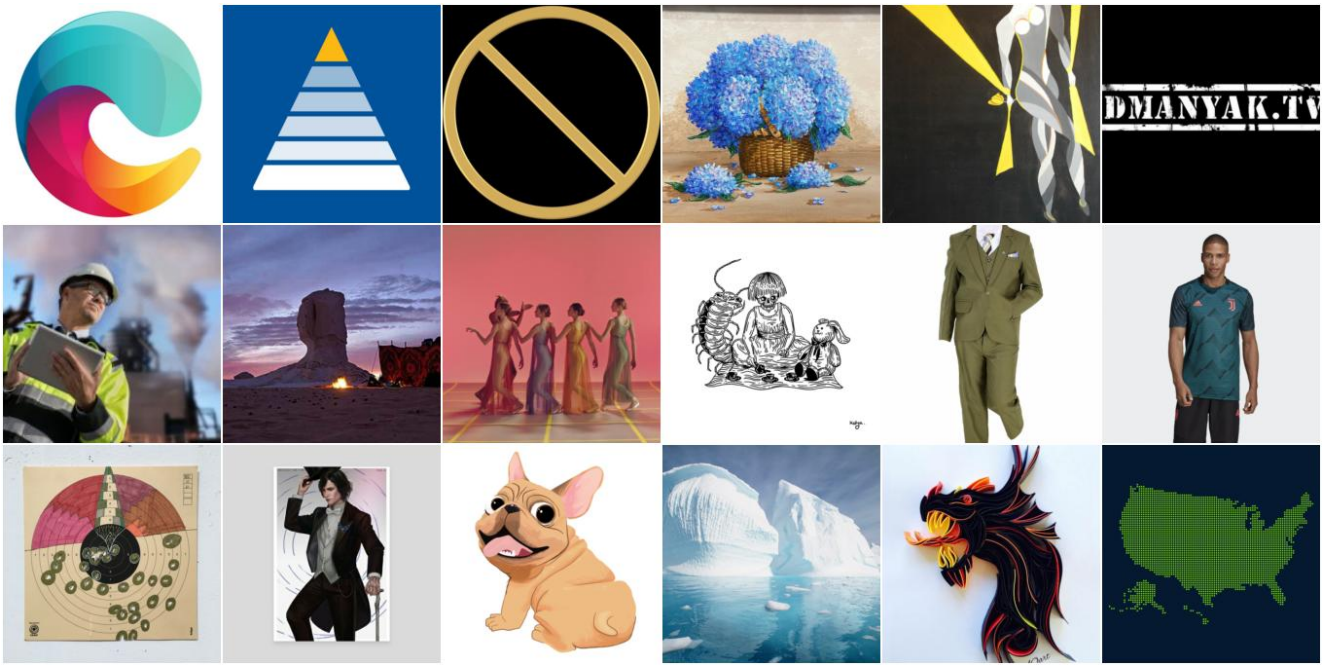


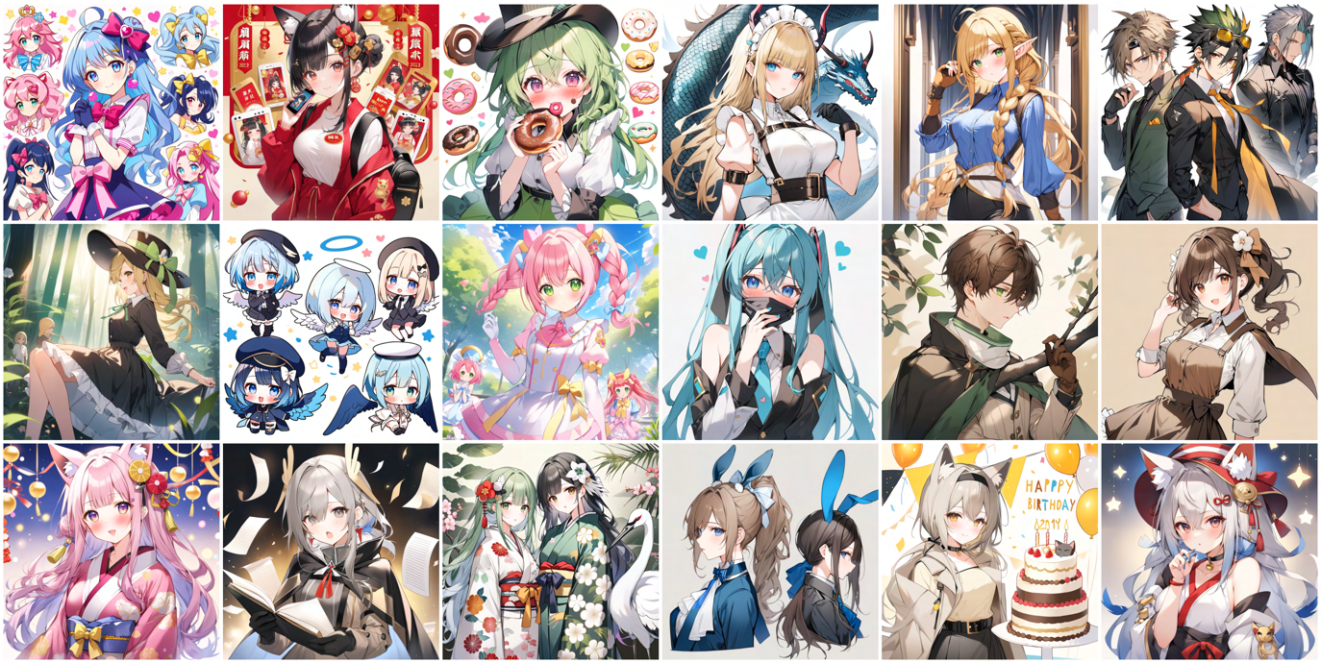
Figure F.15. **Top**: A sample of *fake* Würstchen 2 [39] images we generated for our dataset. **Bot**: Corresponding *real* images found via RIS.

Fake (SD Pokémon Diffusers [4])

Real (Pokémon [5])

Figure F.16. **Top**: A sample of *fake* Pokémon images we generated for our dataset. **Bot**: Corresponding *real* images. The above *fake* images were generated using the BLIP [32] captions of these *real* images as prompts [40].

Fake (Animagine XL 2.0 [7])



Real (Danbooru [6])

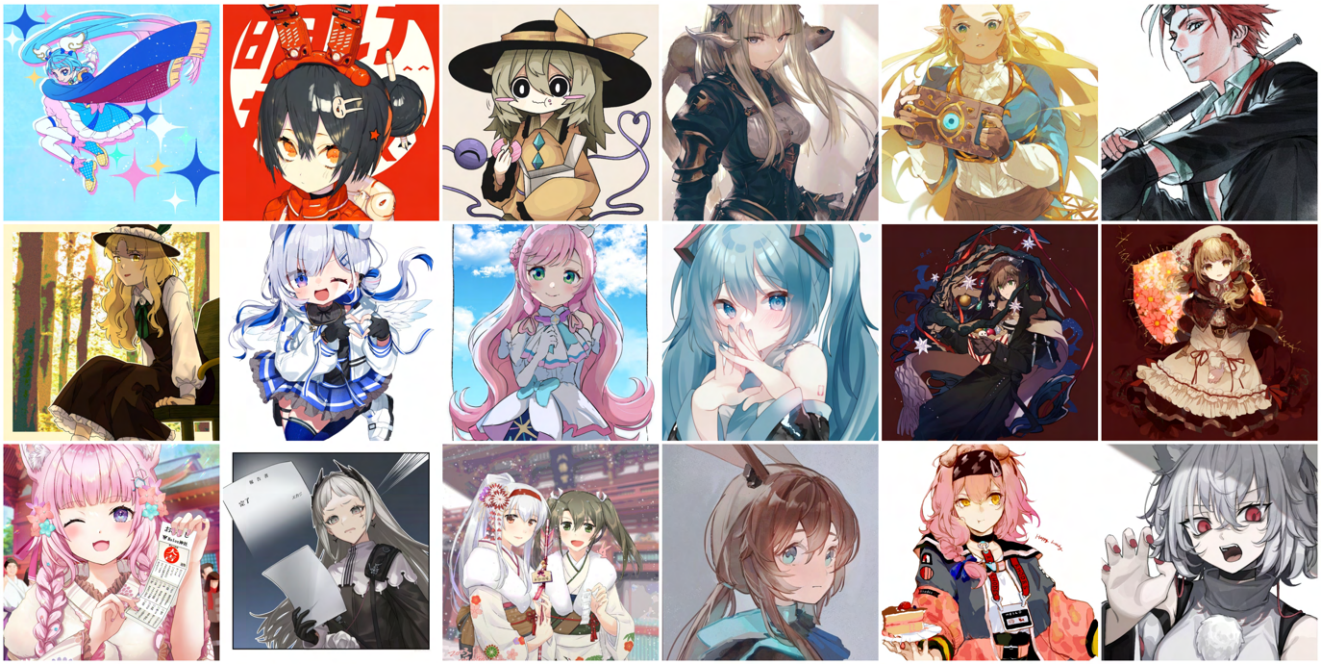


Figure F.17. **Top**: A sample of *fake* Anime images we generated for our dataset. **Bot**: Corresponding *real* images from the (SFW) Danbooru 2022 dataset [6]. The above *fake* images were generated using the Danbooru tags of these *real* images as prompts.