Simpler Diffusion: 1.5 FID on ImageNet512 with pixel-space diffusion

Supplementary Material

A. Connection between sigmoid loss and low-bit training

Weighted diffusion losses can be closely related to low-bit training, as shown in [25]. In earlier normalizing flow work, low-bit training has been shown to achieve higher perceptual quality [24]. The reason that low-bit training improves sample fidelity is quite intuitive: by throwing away less significant (and thus important) bits from the training data, the model spends more capacity on modeling more significant bits. Here we will show how sigmoid loss is related to low-bit training, and therefore provide a theoretical understanding on how sigmoid loss balances the bits from the data of different importance.

Training diffusion models on different bit precisions and their loss contributions may give results that are entangled with optimization hyperparameters. We instead consider a simplified data distribution setting: assume the data is univariate, with a uniform distribution over the 2^n possible values, where n is the bit precision. We further assume the model to be optimal for all noise levels, which is a mixture-of-Gaussians whose means are the 2^n possible values and variance is determined by the log signal-to-noise ratio λ. A diffusion loss over the entire dataset can be expressed as:

$$L = \mathbb{E}_{q(\boldsymbol{x})}\mathbb{E}_{\lambda\sim q(\lambda)} w(\lambda)\|\boldsymbol{x} - \hat{\boldsymbol{x}}_\lambda\|^2, \tag{6}$$

$$= \mathbb{E}_{q(\boldsymbol{x})}\mathbb{E}_{\lambda\sim q(\lambda)} \tilde{w}(\lambda)\|\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}_\lambda\|^2, \tag{7}$$

$$= \mathbb{E}_{\lambda\sim q(\lambda)} \tilde{w}(\lambda)\mathbb{E}_{q(\boldsymbol{x})}\|\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}_\lambda\|^2, \tag{8}$$

where $q(\lambda)$ is determined by the mapping λ_t and $t \sim \mathcal{U}(0, 1)$, and $q(\boldsymbol{x})$ is the data distribution. Given the simplified data distribution and the optimal model, we can compute $\mathbb{E}_{q(\boldsymbol{x})}\|\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}_\lambda\|^2$ analytically. Figure 11 shows its value over λ for multiple data distributions of different bit precision n.

We can also plot the weighted loss with weighting function $\tilde{w}(\lambda)$ at different noise levels $\tilde{w}(\lambda)\mathbb{E}_{q(\boldsymbol{x})}\|\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}_\lambda\|^2$. Given that the optimal sigmoid weighting at resolution 128 is $\tilde{w}(\lambda) = \sigma(1 - \lambda)$, we follow [14] to shift the weighting to $\tilde{w}(\lambda) = \sigma(10.7 - \lambda)$ for the univariate example, where the additional bias term comes from $2\log(128)$. The weighted loss is shown in Figure 12. Interestingly, if we follow the low-bit training but instead of assuming the data is a uniform distribution at a certain precision, we assume it is a mixture of multiple precisions, and then the loss ends up being very similar to the one with the sigmoid weighting (Figure 12). The mixture we visualize here is $[8, 7, 6, 5]$-bit with a mixture proportion of $[1, 4, 4, 6]$. Informally speaking, the loss of the sigmoid weighting is very similar training on a lower-bit distribution of the data, where the data is only modelled at full precision occasionally.

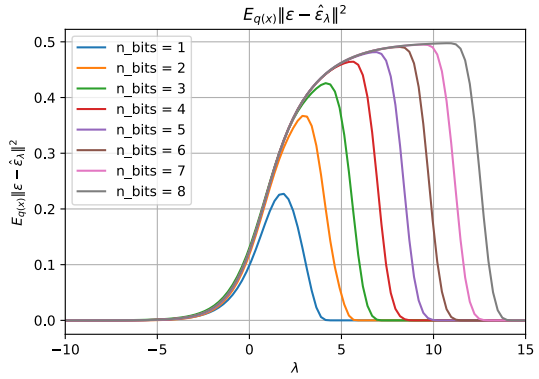


Figure 11. Expected loss over log signal-to-noise ratio λ for multiple data distributions of different bit precision.

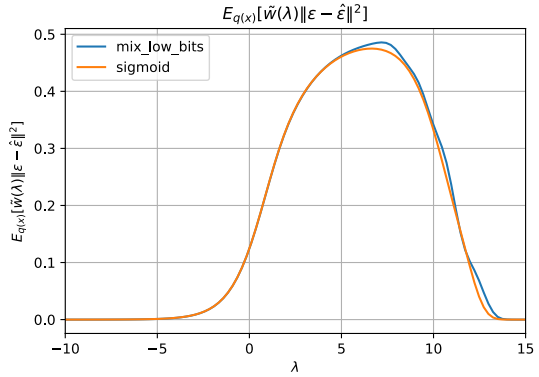


Figure 12. Weighted loss with a sigmoid weighting, compared with a low-bit training of a mixture of precisions.

B. Experimental Details

B.1. Sigmoid Loss

The sigmoid loss can be easily implemented in the following way:

```
def sigmoid_loss(x, model_x, logsnr_fn, t, bias):
    logsnr = logsnr_fn(t)
    dlogsnr_dt = jax.jvp(
        logsnr_fn, t (ones_like(t),))[1]
    weight = -0.5 * dlogsnr_dt * exp(
          bias) * sigmoid(logsnr - bias)
    return weight * mean_except_batch((x-model_x)**2)
```

For completeness, recall that a cosine interpolated logsnr schedule function from [14] can be defined as:

```
def cosine_interpolated(
    t, lognsr_min=-10, logsnr_max=10,
    image_res=512, noise_res_low=32,
    noise_res_high=512)
  log_change_high = log(image_res) - log(noise_res_high)
  log_change_low = log(image_res) - log(noise_res_low)

  b = arctan(exp(-0.5 * logsnr_max))
  a = arctan(exp(-0.5 * logsnr_min)) - b
  logsnr_cosine = -2. * (jnp.log(jnp.tan(a * t + b))
  logsnr_high = logsnr_cosine + log_change_high
  logsnr_low = logsnr_cosine + log_change_low
  return (1 - t) * logsnr_high + t * logsnr_low
```

Observe here that the min and max values are shifted along as well and are not the limits of the resulting function.

B.2. Power loss

Recall that the power loss aims to amplify the loss of low sub-band (smooth) signals in the spatial domain via Haar wavelets. *Averaging (ie, low passes) increases the logsnr, therefore we adjust the logsnr based on the number of low passes.* We apply the Haar wavelet transformation iteratively to the input $I = \hat{\boldsymbol{x}} - \boldsymbol{x}$. The Haar wavelet $W(I) = (L, H)$ transforms the input into a low sub-band (smooth) $s_k = (i_{2k} + i_{2k+1})/\sqrt{2}$ and a high sub-band (detail) $d_k = (i_{2k} - i_{2k+1})/\sqrt{2}$. First W is applied to the rows $W_r(I) = (L, H)$ and then to the columns on each sub-band $W_c(L) = (LL, LH)$ and $W_c(H) = (HL, HH)$. To give an example, for a 512^2 image, applying this two times results in the sub-bands $S = \{LLLL, LLLH, LLHL, LLHH, LH, HL, HH\}$ where the first four sub-bands have a resolution of 128^2 and the last three have a resolution of 256^2.

Intuitively we want to emphasize high frequencies at the higher resolutions and low frequencies at the lower resolutions. Therefore we shift the logsnr per sub-band roughly according to the expected increase in signal-to-noise. We calculate the shift for a sub-band as $b_s = \log(2) \cdot l(s) - b$ where $l(s)$ is the number of low passes on the sub-band s (ie, the number of L's in the sub-band). Just like the sigmoid loss we cancel out the effect of weighting of the noise schedule on the loss $w_s = \sigma(\lambda_t + b_s) \cdot -\frac{\mathrm{d}\lambda_t}{\mathrm{d}t}$. Lastly we sum over each sub-band and multiply the sub-band weight with the squared sum of the sub-band $PL(I) = \sum_{s \in S} w_s(\lambda_t)||s||^2$.

B.3. Estimating Flops

Although FLOPs can be estimated directly using accelerator tooling, it turns out that the most significant operations outweigh all other minor operations (within 1%). These most significant operations are matrix multiplications (including convolutional layers) and the self-attention dot product. The compute footprint of these operations for our architecture can be calculated using the following equations. Consistent with [20], we assume a training step has the computational cost of a forward pass times 3, and that multiply-adds are counted as *one* flop.

```
def transformer_gflops(size, num_channels, blocks):
    # q, k, v, attn_out, mlp in (4), mlp out (4).
    linears = 12 * num_channels ** 2 * blocks * size ** 2
    attn = 2 * size ** 4 * blocks * num_channels
    return (linears + attn) / 1000**3

def resblock_gflops(size, num_channels, blocks):
  flops = 2 * 3**2 * blocks  # 2 layers with 3x3 conv
  flops *= num_channels ** 2 # channels
  flops *= size ** 2  # spatial resolution
  return flops / 1000**3
```

B.4. Guidance Intervals

We use guidance intervals [27], in which classifier-free guidance is only applied for noise levels within the interval and disabled elsewhere. Based on a grid search we found that the combination of guidance 1.0 on logsnr (-3, +5) obtained the best FID quality on ImageNet512. Without further tuning we found that the minimum logsnr could simply be shifted by approximately 1.5 for a 2x resolution change, similar to how loss biases are shifted. The logsnr max value is typically less sensitive [27], and was kept constant.

B.5. A note on auto-guidance

Besides applying guidance on intervals [27], there is another technique named autoguidance [21] which can even produce better FIDs with EDM-XXL (1.2 from 1.4 with guidance intervals). Autoguidance does not use an unconditional model as negative signal. Instead, the negative signal is provided by a *worse* model, either earlier in training or a small version. Autoguidance increases the hyperparameter space of guidance even further, and we found that this made autoguidance more difficult to tune. For that reason we opted to compare all methods in literature on guidance intervals if available, and also provided the performance of our model with constant guidance to compare with older methods.

B.6. Experimental settings

The small model variant uses the following settings.

```
channels = [128, 256, 512, 1024]
num_updown_blocks = [3, 3, 3],
num_mid_blocks = 16,
block_dropout = [0., 0., 0.1, 0.1],
block_type = ['ResBlock', 'ResBlock',
              'Transformer', 'Transformer'],
mean_type = v
loss_type = sigmoid:-3  # 512^2
patching_size = 4
loss_type = sigmoid:-1  # 256^2
patching_size = 2
loss_type = sigmoid:0   # 128^2
patching_size = 1
```

The flop heavy variant uses the following settings:

```
channels = [128, 256, 512, 1024]
num_updown_blocks = [3, 3, 3]
num_mid_blocks = 16
block_dropout = [0., 0., 0.1, 0.1]
block_type = ['ResBlock', 'ResBlock',
              'Transformer', 'Transformer']
mean_type = v
loss_type = sigmoid:-3  # 512^2
patching_size = 2
loss_type = sigmoid:-1  # 256^2
```

```
patching_size = 1

loss_type = sigmoid:0   # 128^2
patching_size = 1
# for 128^2 resolutions the top-level
# layer (128 channels) is entirely removed.
```

To our surprise, a grid search over guidance strength and guidance interval determined that the same settings were optimal for the small and flop heavy variant on ImageNet512. We then shifted the lower bound of the guidance interval based on resolution shift. Resulting in:

```
guidance_interval = (-3, 5)     # 512^2
guidance_interval = (-1.5, 5)   # 256^2
guidance_interval = (0., 5)     # 128^2
guidance = 1.0
num_steps = 512
sampler = 'ddpm'
clip_x = 'static'
logvar_type = '0.3'
```

The training settings are:

```
batch_size=2048
optimizer='adam'
adam_beta1 = 0.9
adam_beta2 = 0.99
adam_eps = 1.e-12
diffusion_schedule =
    'cosine_interpolated_low_32_high_512'
learning_rate=1e-4
learning_rate_warmup_steps=10_000
weight_decay=0.0
ema_decay=0.9999
max_train_steps = 1_000_000  # for small variants
max_train_steps = 800_000    # for flop heavy
```

B.7. Kinetics600

The training settings are:

```
batch_size=512
optimizer='adam',
adam_beta1=0.9
adam_beta2=0.98
adam_eps=1.e-10
learning_rate=1e-4
learning_rate_warmup_steps=100
diffusion_schedule =
    'cosine_interpolated_low_32_high_128'
weight_decay=0.0
ema_decay=0.9999
max_train_steps = 500_000
```

And the model settings are:

```
channels = [128, 256, 512, 1024]
num_updown_blocks = [3, 3, 4],
num_mid_blocks = 8,
block_dropout = [0., 0., 0.1, 0.1],
block_type = ['ResBlockConv3D', 'ResBlockConv3D',
              'LocalTransformer_19_19_19',
              'Transformer'],
mean_type = v
loss_type = sigmoid:0
guidance = 0.  # unconditional
num_steps = 128
sampler = 'aDDIM'
sampler_noise = 'data_0.9'
clip_x = 'static'
```

Using the same gflop calculations as before adapted to 3D Convs, our 3570 GFLOPs per forward pass, and about 2.5 zettaflops for training. For reference, the base model of

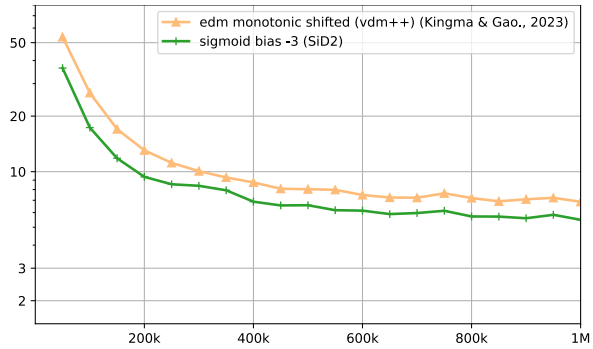
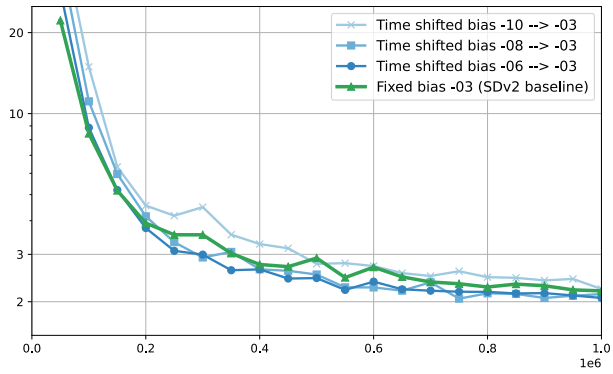
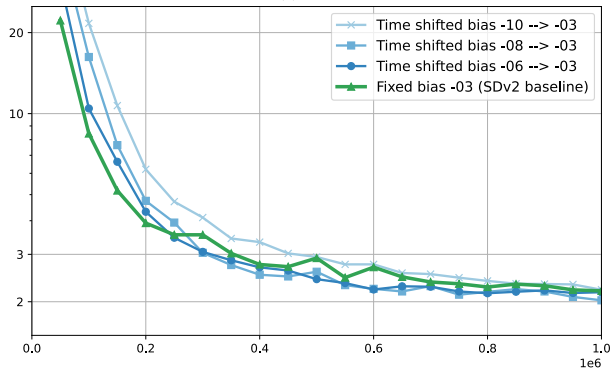


Figure 13. The sigmoid $b = -3$ loss versus the EDM monotonic loss, without any guidance with the small architecture variant.



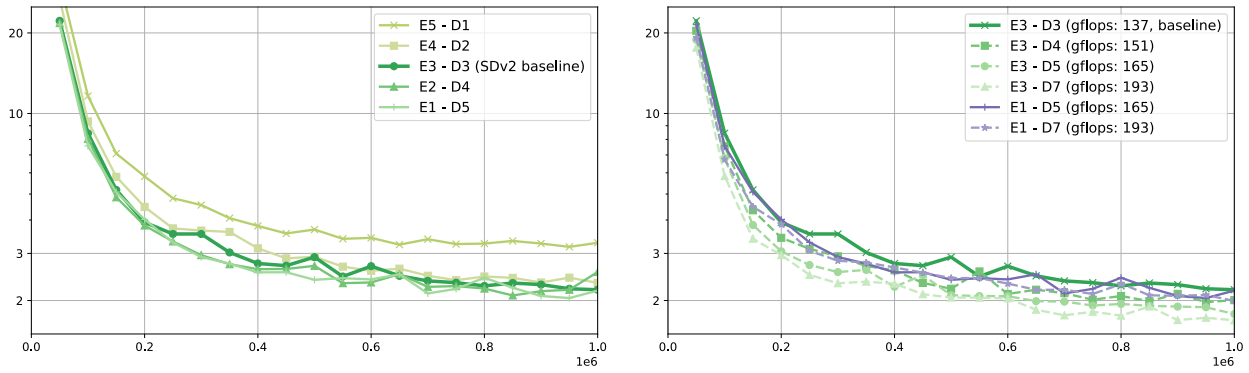
(a) 100k



(b) 200k

Figure 14. Timeshift results. Setting the bias more aggressively and then annealing to the best known setting improves performance somewhat, although the differences are relatively small.

W.A.L.T. uses approximately 0.5 zettaflops (not counting the autoencoder training). This shows that pixel-based models can outperform latent approaches and scale much better using our tuning than before. On the other hand, they still require more compute to achieve this performance that latent approaches. During sampling we use aDDIM [10] because we found that it allowed fewer sampling steps (128) with acceptable performance without much noise schedule tuning, making evaluation during training a bit faster.



(a) fixed depth (b) increasing depth

Figure 15. Encoder-Decoder Experiments

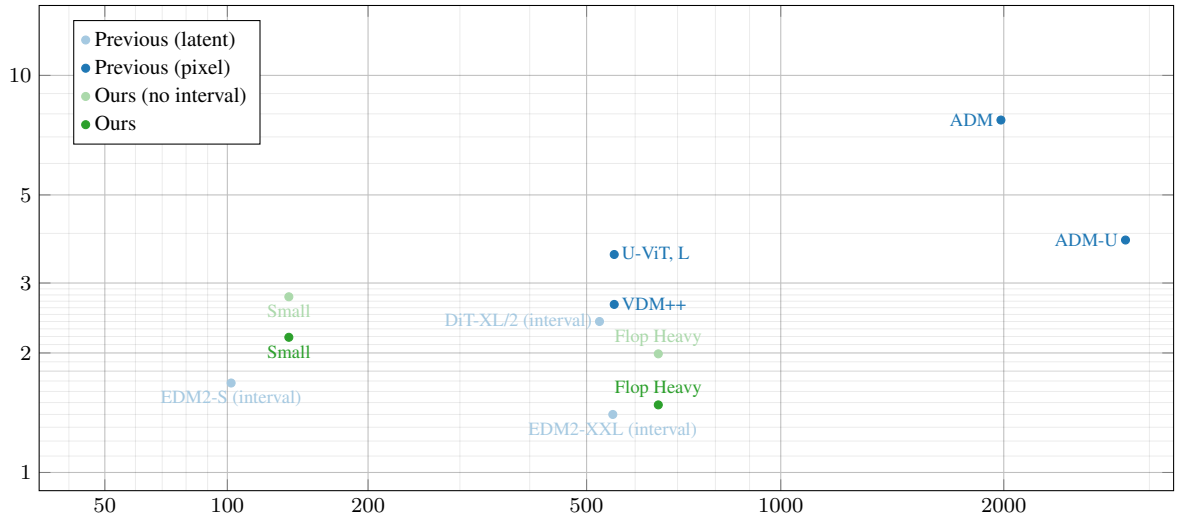


Figure 16. Model forward pass complexity (gigaflops) on ImageNet512, figure adapted from [20]. *Note: both axis are in log-scale.* See also Table 5 for more details.

C. Additional Results

C.1. No Guidance

To validate the difference between the sigmoid and edm-monotonic loss even further, we analyze the FID performance over training iterations without applying any guidance to either model. As can be seen in Figure 13, the difference in performance is even more pronounced when no guidance is applied.

C.2. Timeshifted Sigmoid Losses

The bias term in the sigmoid loss function could be used to emphasize on the low-level information (bits) of the generated images. One could argue that during the course of training, it is important to first focus on the structure of the image and only later penalise mismatches in the high-frequency details. Therefore, we ran a set of experiments with a time shifted bias, where the bias is increased with a linear warmup, we start with bias b_{start} and interpolate over t_b steps towards b_{end}, yielding: $b_t = b_{\text{start}} + (b_{\text{end}} - b_{\text{start}}) \cdot \min(t/t_{\text{b}}, 1)$. So the the bias is interpolated from a starting value to the (fixed) end value (-3) in 100k or 200k training steps.

In Figure 14, we report the FID over the course of training. From the results we observe that shifting the loss during training generally leads to a small improvement, albeit resulting in two additional hyper parameters (starting value and number of steps). Especially when the bias is not too low: shifting from bias -8/-6 to -3 is a good idea, as long as the warmup is relatively quick (100k iterations). Surprisingly, for warmups to 200k iterations the time shifted models have worse performance in early training of stages.

C.3. Asymmetric U-ViTs

In our U-ViT design only a single skip connection per encoding/decoding level is used, allowing for different number of blocks in each level. Here we experiment using differ-

Table 5. Comparison on sampling cost in FLOPs and generation methods.

Method	NFE	Inference cost (GFLOPs)	Sampling cost (TFLOPs)	Sampling time (s)	FID
StyleGAN-XL [38]	1	-	-	0.10	2.40
EDM2-S [20]	63	102	42.5	-	1.68
EDM2-XXL [20]	63	552	61.6	-	**1.40**
DiT-XL/2 [31]	250	525	262.5	-	2.40
Pagoda [22]	1	-	-	**0.05**	1.80
SiD2 16-step distilled (ours)	16	653	10.4	0.29	1.50

ent numbers of encoding blocks versus decoding blocks at the same level. The results are in Figure 15. We see that increasing depth helps, and decoder could be a little heavier than the encoder, although symmetric scaling (E3-D3, our baseline) already acts as a strong baseline.

C.4. Forward pass complexity

In addition to the training cost in the main text, Figure 16 shows the forward pass complexity. Here we see that again our method outperforms all existing pixel approaches by a large margin, and performs similar albeit somewhat worse than latent approaches.

C.5. Additional samples

We provide more illustrative examples for ImageNet512 generation (Figure 17) and Text-to-Image (Figure 18 and Figure 19).

Figure 17. Some *random (**not** cherry-picked)* class-conditional samples of 512×512 that have been generated by SiD2 flop heavy. Guidance 2.0 on interval logsnr $\in$ (-8, 5). Every row uses the same class-conditioning, every column uses the same rng.

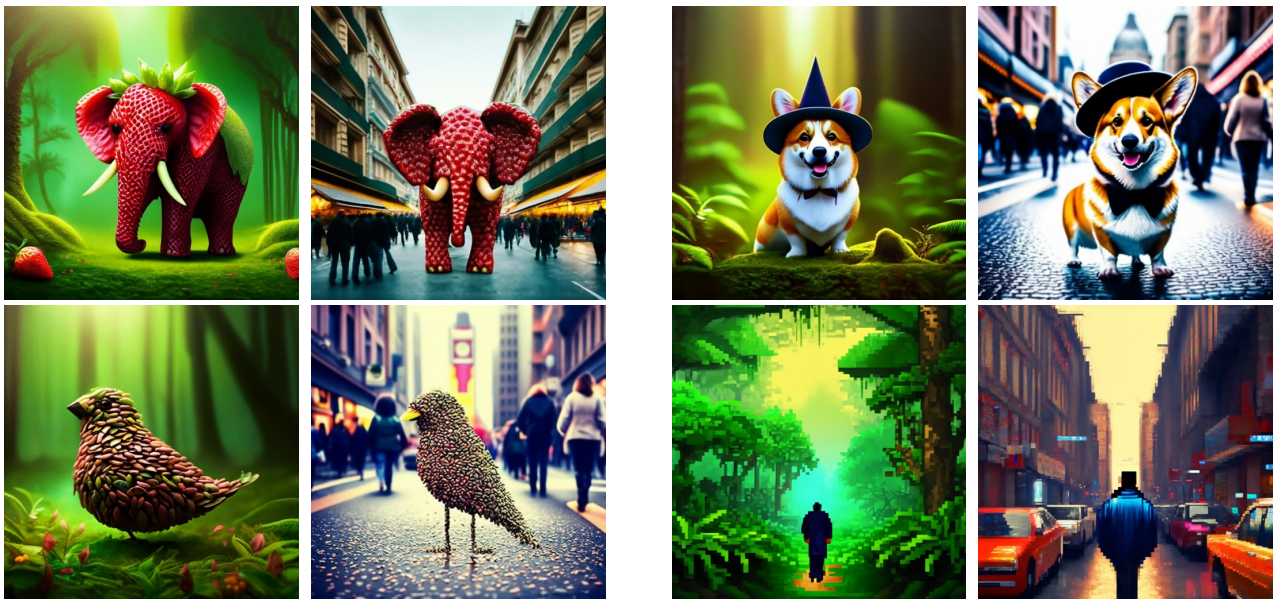


Figure 18. Generations from the text-to-image model. **Used prompts** (the Cartesian product of the sets): "*{A strawberry elephant, A corgi wearing a hat, A bird made of seeds, Pixel art: a man walking}* in a *{lush forest, busy street}*."

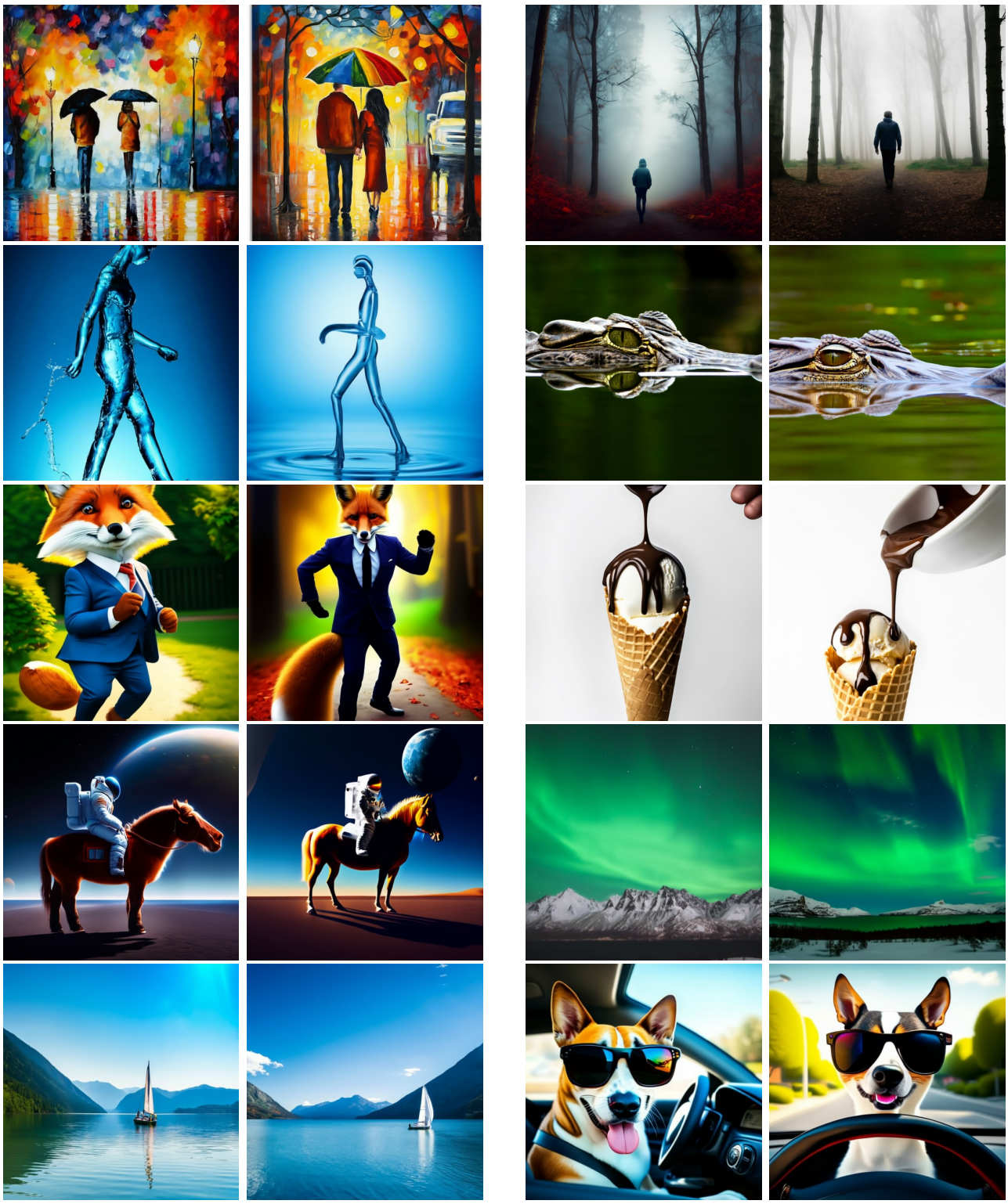


Figure 19. Generations from the distilled text-to-image model. Guidance is set to +3. For each prompt two images selected from four generated images. **Used prompts**: (1) *A couple gets caught in the rain, oil on canvas*, (2) *A lone traveller walks in a misty forest*, (3) *A walking figure made out of water*, (4) *In the swamp, a crocodile stealthily surfaces, revealing only its eyes and the tip of its nose as it moves forward*, (5) *A fox dressed in suit dancing in park*, (6) *Pouring chocolate sauce over vanilla ice cream in a cone, studio lighting*, (7) *An astronaut riding a horse*, (8) *Aurora Borealis Green Loop Winter Mountain Ridges Northern Lights*, (9) *Sailboat sailing on a sunny day in a mountain lake*, (10) *A dog driving a car on a suburban street wearing funny sunglasses*.