config	BiFlow	improved TARFlow			
	B/2	B/2	M/2	L/2	XL/2
# Params (M)	133	120	296	448	690
block	9	8	10	12	15
layer	8	8	9	9	9
hidden dim	384	384	512	576	640
attn heads	6	6	8	9	10
patch size	2	2			
class tokens	8	—			
guidance tokens	4† / 1	—			
epochs	160† / 350	960	640	640	480
batch size	256	256			
learning rate	4e-4	4e-4	4e-4	2e-4	1e-4
lr schedule	constant	constant			
lr warmup	10 epochs	10 epochs			
optimizer	Adam	Adam [28]			
Adam (β_1, β_2)	(0.9, 0.95)	(0.9, 0.95)			
weight decay	0.0	0.0			
dropout	0.0	0.0			
ema decay	0.9999	0.9999			
label drop	0.1	0.1			
adaptive weight p	1	—			
w_{VGG}	1.0† / 0.8	—			
w_{ConvNeXt}	0.4† / 0.6	—			
clip range c	—	1.0	1.0	3.0	3.0
noise level σ	—	0.3			

Table 5. **Configurations and training hyperparameters on ImageNet 256×256**. † indicates the setting in the ablation study.

A. Implementation Details

We implement all experiments using the JAX framework [2] on Google TPU hardware. All reported results are obtained on TPU v4, v5p, and v6e cores. The configurations and training hyperparameters for improved TARFlow and BiFlow are provided in Tab. 5. For the MSE-only ablation in Sec. 5, we employ adaptive weighting with exponent $p = 2$; for all other experiments we use $p = 1$ (see Appendix C.3 for detailed ablations).

FID Evaluation. For generative evaluation, we compute the Fréchet Inception Distance (FID) [23] between 50,000 generated images and training images, without applying any data augmentation. We use the Inception-V3 model [54] provided by StyleGAN3 [27], converted into a JAX-compatible implementation. We sample 50 images per class for all 1000 ImageNet classes, following the protocol in [69].

Inference Cost Evaluation. In Fig. 2 and Tab. 3, we report inference cost across three hardware configurations: GPU, TPU, and CPU. For all metrics, we report the average per-image runtime in seconds, averaged over multiple runs to ensure stability. All measurements include the overhead of CFG and VAE decoding time when applicable. We also provide a comparison with prior Normalizing Flow models [65, 21] in Tab. 6. All autoregressive models utilize KV-cache to accelerate inference, and Gflops in Tab. 3 is estimated using JAX's `cost_analysis` function.

Method	# Params	Time (ms)	Speed	VAE?
TARFlow-XL/8@pix [65]	1.3B	1192	1×	✗
STARFlow-XL/1 [21]	1.4B	677+1.3	1.76×	✓
iTARFlow-B/2	120M	65+1.3	18.0×	✓
iTARFlow-M/2	296M	85+1.3	13.8×	✓
iTARFlow-L/2	446M	165+1.3	7.17×	✓
iTARFlow-XL/2	675M	202+1.3	5.86×	✓
BiFlow-B/2 (Ours)	133M	**0.29+1.3**	**750×**	✓

(a) TPU inference time comparison, benchmarked on 8 TPU v4 cores with a pre-compiled JAX sampling function.

Method	# Params	Time (ms)	Speed	VAE?
TARFlow-XL/8@pix [65]	1.3B	3452	1×	✗
STARFlow-XL/1 [21]	1.4B	2193+2.7	1.57×	✓
iTARFlow-B/2	120M	129+2.7	26.2×	✓
iTARFlow-M/2	296M	208+2.7	16.4×	✓
iTARFlow-L/2	446M	349+2.7	9.82×	✓
iTARFlow-XL/2	675M	400+2.7	8.57×	✓
BiFlow-B/2 (Ours)	133M	**2.15+2.7**	**712×**	✓

(b) GPU inference time comparison, benchmarked on 1 NVIDIA H200 core with PyTorch and `torch.compile` optimization if beneficial.

Method	# Params	Time (ms)	Speed	VAE?
TARFlow-XL/8@pix [65]	1.3B	512000	1×	✗
STARFlow-XL/1 [21]	1.4B	276700+240	1.85×	✓
iTARFlow-B/2	120M	9040+240	55.2×	✓
iTARFlow-M/2	296M	16200+240	31.1×	✓
iTARFlow-L/2	446M	20400+240	24.8×	✓
iTARFlow-XL/2	675M	26300+240	19.3×	✓
BiFlow-B/2 (Ours)	133M	**80+240**	**1600×**	✓

(c) CPU inference time comparison, benchmarked on 1 AMD EPYC 7B12 node with 120 physical CPU cores and 400GB RAM. We reuse the PyTorch implementations with `torch.compile` optimization if beneficial.

Table 6. **Comparison of NF models' inference wall-clock time on TPU, GPU, and CPU**. The wall-clock time is evaluated per image on average in milliseconds. All models include the overhead of CFG at inference time, as well as the VAE decoding time when applicable. All autoregressive models utilize KV-cache to accelerate inference. See Appendix A for further details.

For TPU wall-clock time, all models are evaluated using a pre-compiled JAX sampling function on 8 TPU v4 cores. Reported times exclude compilation overhead. We use a local device batch size of 10 for model inference, and 200 for VAE decoding.

For GPU wall-clock time, all models are re-implemented in PyTorch and evaluated on a single NVIDIA H200 GPU with a batch size of 128. The VAE decoding time is obtained with `torch.compile` optimization.

For CPU wall-clock time, we reuse the PyTorch implementation on a single AMD EPYC 7B12 node (120 physical CPU cores and 400 GB RAM). We use a smaller batch size of 64 for most models; however, TARFlow and STARFlow are restricted to a batch size of 4 due to efficiency concerns. We observe that batch size has a negligible impact on per-image CPU inference time. All other experimental settings remain consistent with the GPU evaluation.

Algorithm 1 BiFlow: Training.

```
# x: training batch, (N, H, W, C)
# F: forward model (B blocks), frozen
# G: reverse model (B + 1 blocks)
# phi: projection heads

# noise injection
e = randn_like(x)
x_tilde = x + noise_level * e

# get forward trajectory xs and prior z
xs, z = F(x_tilde)

# get reverse trajectory hs and
    reconstructed x'
hs, x_prime = G(z)

# project hidden into input space
for i in range(B):
   hs[i] = phi[i](hs[i])

# compute loss
loss_align = mse(xs, hs)
loss_recon = metric(x, x_prime)
loss = loss_align + loss_recon
```

Algorithm 2 BiFlow: 1-NFE Sampling.

```
e = randn(x_shape)
_, x = G(e)
```

B. Method Details

B.1. Pseudocode

We provide the pseudocode for training our BiFlow model with hidden alignment in Alg. 1, as well as the 1-NFE sampling procedure in Alg. 2.

In the algorithm, the forward model $\mathcal{F}$ produces the entire forward trajectory `xs`, *i.e.*, $\tilde{x}, x^1, x^2, \ldots, x^{B-1}$, along with the prior $z = x^B$. Similarly, the reverse model $\mathcal{G}$ outputs the sequence of intermediate hiddens states `hs` as reverse trajectory: $h^{B-1}, h^{B-2}, \ldots, h^0$, along with the reconstructed clean input `x_prime`.

The final loss function consists of alignment loss between forward and reverse hidden states in Eq. (2), and reconstruction loss between the clean input `x` and reconstructed output `x_prime`.

B.2. BiFlow with Guidance

Training-time CFG. As discussed in Sec. 4.5, to enable guided sampling within a single forward pass (1-NFE), we directly train a guided reverse model $\mathcal{G}_\phi^{\text{cfg}}$ defined as

$$\mathcal{G}_\phi^{i,\text{cfg}}(h^i \mid \mathbf{c}) = (1 + w_i)\mathcal{G}_\phi^i(h^i \mid \mathbf{c}) - w_i \mathcal{G}_\phi^i(h^i).$$

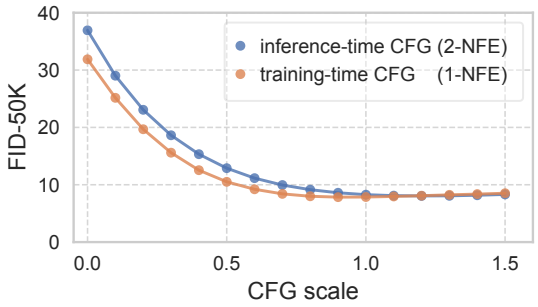


Figure 7. **Comparison between training-time and inference-time CFG of BiFlow**. Training-time CFG achieves similar or better performance compared to inference-time CFG while requiring only half inference compute and retaining full post-hoc tuning flexibility. (Settings: BiFlow-B/2, 160 epochs, MSE-only baseline.)

where w_i is the guidance scale at block i. The unconditional output of $\mathcal{G}_\phi^{i,\text{cfg}}$ matches that of the original $\mathcal{G}_\phi^i$. Therefore, the unguided block output can be expressed as

$$h^{i+1} = \frac{\mathcal{G}_\phi^{i,\text{cfg}}(h^i \mid \mathbf{c}) + w_i \mathcal{G}_\phi^{i,\text{cfg}}(h^i)}{1 + w_i}. \tag{5}$$

During training, we compute our hidden-alignment loss directly on h^{i+1} from Eq. (5). At inference time, this formulation allows us to use $\mathcal{G}_\phi^{i,\text{cfg}}$ directly, producing guided samples with only a 1-NFE forward pass. We add stop gradient to the unconditional output to stabilize training.

Guidance conditioning. To retain the ability to adjust the CFG scale at inference time, we explicitly condition the reverse model on the guidance scale [39, 18], *i.e.*, $\mathcal{G}_\phi^{i,\text{cfg}}(h^i \mid \mathbf{c}, w_i)$. During training, we sample w_i from a uniform distribution $\mathcal{U}(0, w_{\max})$ and apply Eq. (5) to compute the unguided output h^{i+1} for hidden alignment loss.

We compare this training-time CFG scheme with the more conventional inference-time CFG in Fig. 7. Training-time CFG achieves similar (even better) performance while preserving the 1-NFE efficiency and the flexibility to sweep CFG scales at inference time.

B.3. Distance Metric

Adaptive weighting. We adopt the adaptive loss reweighting strategy from [16, 52, 17]. Given a prediction x and target y, the adaptive-weighted distance is defined as:

$$\mathcal{D}_p = \text{sg}(w_p) \cdot \mathcal{D}(x, y), \quad w_p = (\mathcal{D}(x, y) + c)^{-p},$$

where c is a small constant and $p \geq 0$ is adaptive weight. $\text{sg}(\cdot)$ denotes the stop-gradient operator. We apply adaptive weighting to all loss terms in our training objective.

VGG feature. For the perceptual loss based on VGG features, we follow the LPIPS formulation [68]. Since our model operates in the latent space of a pre-trained VAE, we decode the predicted latent x' back into image space and compute the LPIPS loss against the ground-truth image.

	MSE	LPIPS
naive distillation	0.115	0.331
hidden distillation	0.156	0.392
hidden alignment	**0.111**	**0.321**

Table 7. **Reverse learning methods: reconstruction fidelity.** We report MSE and LPIPS between the original sample x and the reconstructed sample x' produced by the learned reverse model. Among the three strategies for approximating the inverse transformation, the hidden alignment method achieves the most accurate reconstruction. The corresponding FIDs are shown in Tab. 1 (Settings: BiFlow-B/2, 160 epochs, no CFG, adaptive-weighted MSE loss only. All three rows share the same forward model.)

ConvNeXt feature. In addition to VGG features, we incorporate ConvNeXt V2 [61] (ImageNet-22K pre-trained, base-size) as a complementary perceptual feature extractor. Similar to the LPIPS, both the reconstructed image x' and the ground-truth image x are passed through the ConvNeXt network after VAE decoding. The perceptual distance is computed using the extracted features, excluding the final classification head.

Usage of loss terms. In the ablation studies in Sec. 5, we use only the adaptively weighted MSE loss unless otherwise noted. In Tab. 4, we combine all three loss terms:

$$\mathcal{L}(x) = \sum_i \mathcal{L}_{\text{MSE}}(x^i, \varphi_i(h^i)) + w_{\text{VGG}}\mathcal{L}_{\text{VGG}}(x, x') + w_{\text{ConvNeXt}}\mathcal{L}_{\text{ConvNeXt}}(x, x'),$$

where w_{VGG} and w_{ConvNeXt} are tunable hyperparameters. We observe that the final performance is particularly sensitive to w_{ConvNeXt}. Concrete weights are specified in Appendix A.

Normalized Trajectory. As described in Sec. 4.4, the reverse model is trained to align with a normalized forward trajectory. Specifically, we pre-compute the squared norm $\|x_i\|^2$ of each trajectory point and average it over the entire dataset. During reverse model training, the intermediate trajectory points are divided by $\sqrt{\mathbb{E}[\|x_i\|^2]}$, ensuring scale consistency across different blocks.

We do not use normalized trajectories in any experiments except the one in Tab. 2c, as we observe no significant difference when combined with iTARFlow. Nonetheless, normalized trajectories are worth noting for scenarios where one wishes to use a pre-trained NF model without clipping.

C. Additional Experiments

C.1. Learning to Approximate the Inverse

In Tab. 1, we compare the empirical performance of the three reverse learning approaches introduced in Sec. 4. Here, we further provide quantitative results on their reconstruction fidelity in Tab. 7. Specifically, we evaluate the reconstruction distance $\mathcal{D}(x, x')$ using MSE and LPIPS (VGG-based) as metrics.

$w_d \backslash w$	0.5	0.6	0.7
0.5	10.51	9.45	8.77
0.6	10.21	9.21	8.59
0.7	9.93	8.99	8.41
3.5	7.05	6.87	6.89
4.0	6.94	6.81	6.87
4.5	6.88	**6.79**	6.87
5.0	6.86	6.80	6.89

Table 8. **Separate guidance scale for the denoising block**. BiFlow eliminates the score-based denoising step in TARFlow by learning a dedicated denoising block, jointly trained with other blocks. This denoising block serves a different purpose from the rest NF reverse process. Using a separate, larger guidance scale for the denoising block improves sample quality. (Settings: BiFlow-B/2, 160 epochs, adaptively-weighted MSE only, training-time CFG. FID w/o CFG: 31.88.)

We observe that the proposed hidden-alignment strategy achieves the lowest regression loss across both metrics. This indicates that hidden alignment provides a more accurate mapping between x and x', leading to a better-behaved reverse learning process.

C.2. BiFlow with Guidance

As discussed in Sec. 4.2, the additional denoising block in our reverse model functions as a dedicated denoiser, while the preceding B blocks focus on inverting the forward sub-transformations. This structure naturally motivates applying CFG differently across these two components. We empirically validate this design choice in Tab. 8.

For training-time CFG, we use a shared guidance scale across all blocks, sampling w from a simple uniform prior $\mathcal{U}[0, 0.5]$. In ablation studies that use MSE loss only (Sec. 5), we decouple the guidance scales for the inverse blocks and the denoising block, since the optimal pair (w, w_d) typically satisfies $w_d \gg w$. In this case, we sample $w \sim \mathcal{U}[0, 1]$ and $w_d \sim \mathcal{U}[0, 8]$.

C.3. Distance Metric

In Sec. 4.3, we discuss different choices of distance metrics for training BiFlow. We ablate the choice of perceptual distance terms in Tab. 9. First, we compare different feature extractors, including a ResNet-101 [22] pre-trained for classification and a DINOv2-B model [40], as reported in Tab. 9a. For ResNet-101, we extract features by removing the final MLP head, following the same procedure as ConvNeXt. Among all tested feature extractors, ConvNeXt achieves the best empirical performance. We further evaluate the combination of VGG and ConvNeXt features in Tab. 9b. The results indicate that using both features together yields better FID scores than using either one individually.

Furthermore, we study the effect of adaptive weighting in the MSE loss in Tab. 10. MSE with adaptive weighting consistently outperforms the naive MSE loss.

feature model	FID, w/o CFG	FID, w/ CFG
none	31.88	6.79
VGG + ResNet	9.69	4.34
VGG + DINO	9.33	4.36
ConvNeXt + DINO	3.19	3.19
VGG + ConvNeXt	**2.46**	**2.46**

(a) Feature model ablation.

w_{VGG}	w_{ConvNeXt}	FID, w/o CFG	FID, w/ CFG
0.0	0.0	31.88	6.79
1.0	0.0	16.97	5.31
0.0	0.4	2.62	2.62
1.0	0.4	**2.46**	**2.46**

(b) VGG and ConvNeXt weight ablation.

Table 9. **Ablation on perceptual loss for BiFlow-B/2**. FID-50K with/without CFG are reported. (Settings: BiFlow-B/2, 160 epochs, training-time CFG, weight for two perceptual losses are 1.0 and 0.4 by default.)

adaptive weight p	FID, w/o CFG	FID, w/ CFG
0.0	38.23	7.49
0.5	34.74	7.08
1.0	34.43	**6.70**
2.0	**31.88**	6.79

Table 10. **Ablation on adaptive weighting**. Adaptive weighted MSE loss works better than naive MSE ($p = 0.0$). (Settings: BiFlow-B/2, 160 epochs, adaptive weighted MSE only, training-time CFG.)

C.4. Improved TARFlow Norm Control

To mitigate the imbalance in loss magnitudes across different blocks of BiFlow, we introduce a simple yet effective modification to the original TARFlow [65] in Sec. 4.4: clipping the parameters μ and α within a fixed range. This adjustment stabilizes training and improves final FID performance.

In Fig. 8, we visualize the norms of intermediate trajectory states during training of the improved TARFlow. Without clipping, the norms across blocks diverge sharply and continue to grow as training progresses. With clipping, the norms remain stable and well-controlled within a reasonable range. Such normalization substantially benefits the training of the reverse model in the downstream.

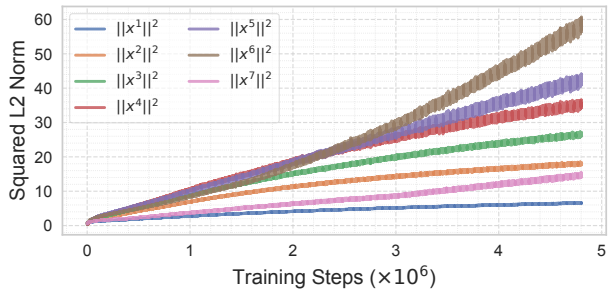
C.5. Improved TARFlow CFG

For completeness, we also examine classifier-free guidance (CFG) designs for TARFlow, although this component is orthogonal to our main contributions. In the original TARFlow [65], the reverse update rule at block i is

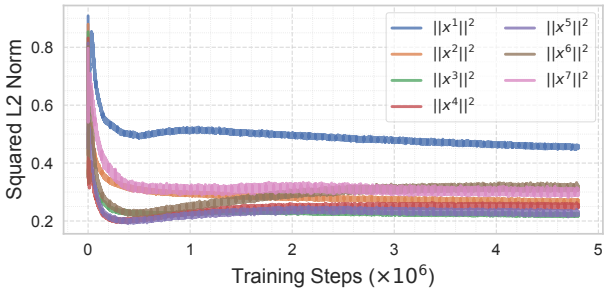
$$z^i_{t,\text{cfg}} = z^{i+1}_t \odot \exp(\alpha^i_{t,\text{cfg}}) + \mu^i_{t,\text{cfg}},$$

where guidance is applied to the predicted parameters by

$$\alpha^i_{t,\text{cfg}} = (1+w_t)\,\alpha^i_t(\cdot \mid \mathbf{c}) - w_t\,\alpha^i_t(\cdot),$$
$$\mu^i_{t,\text{cfg}} = (1+w_t)\,\mu^i_t(\cdot \mid \mathbf{c}) - w_t\,\mu^i_t(\cdot),$$



(a) Improved TARFlow w/o clipping.



(b) Improved TARFlow w/ clipping.

Figure 8. Comparison of intermediate states' norm during TARFlow training between training with and without clipping. **Left**: without clipping, the norms at different blocks diverge significantly, and continue to increase as training proceeds. **Right**: with clipping, the norms are well controlled within a reasonable range, stabilizing training and improving final FID scores.

	Linear		Const	
	μ, α	z	μ, α	z
Online	**6.83**$_{(2.8)}$	**6.82**$_{(2.8)}$	7.26$_{(1.3)}$	7.24$_{(1.3)}$
Offline	22.14$_{(1.2)}$	22.03$_{(1.2)}$	18.23$_{(1.0)}$	18.11$_{(1.0)}$

Table 11. Improved TARFlow CFG ablation. Online guidance substantially outperforms the offline variants, whereas the choice of guidance schedule (linear vs. constant) and the level at which guidance is applied (μ, α vs. z) has only minor impact. Numbers in gray parentheses denote the corresponding optimal CFG scale. (Settings: improved TARFlow-B/2, FID w/o CFG: 44.46.)

with a linearly increasing guidance schedule $w_t = \frac{t}{T-1}w$ along the token dimension. Here, subscript t denotes the token dimension, superscript i denotes the block dimension, and $(\cdot \mid \mathbf{c})$ and $(\cdot)$ represent the conditional and unconditional counterparts, respectively.

Following prior CFG studies in diffusion models, we decompose the design space into three orthogonal choices:

Schedule. We can replace the original linearly increasing w_t with a constant guidance scale: $w_t = w$.

Space for applying guidance. Parameter-space CFG (μ, α) vs. pixel-space CFG applied directly to z:

$$z^i_{t,\text{cfg}} = (1+w_t)z^i_t(\cdot \mid \mathbf{c}) - w_t z^i_t(\cdot).$$

We denote these two settings by "μ, α" and "z", respectively.

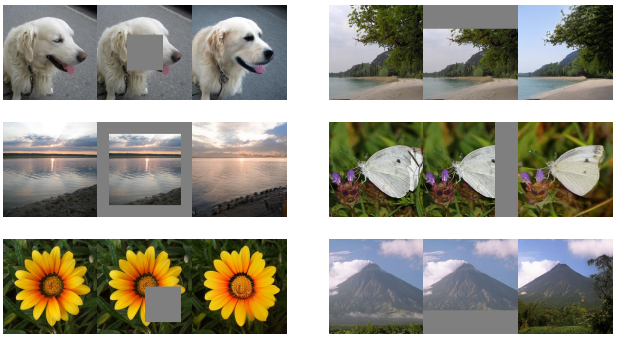


Figure 9. **Inpainting**. BiFlow enables efficient image inpainting by leveraging its bidirectional mapping between images and noise. By resampling the masked part of the noise, BiFlow can perform training-free inpainting on various image masks. Each triplet contains ground-truth image (left), masked image (middle), and reconstructed image (right).

Online vs. Offline. We distinguish between online and offline CFG strategies. The online approach (TARFlow's practice) applies guidance at each generation step; the offline approach generates the entire conditional and unconditional sequences independently and performs extrapolation only once on the final outputs. The difference lies only in how guidance interacts with intermediate states.

While both approaches have similar runtimes, Tab. 11 shows that online CFG significantly outperforms the offline variant. Regarding other hyperparameters, a linear schedule offers a slight advantage over a constant one, while applying guidance in parameter space versus pixel space yields similar performance. Overall, TARFlow's original CFG formulation is close to optimal.

Based on these results, we use the original TARFlow CFG formulation (Online, Linear, Parameter-space) as our baseline. It is important to note that this CFG setting only affects the inference quality of the forward model; the training of our BiFlow reverse model always relies on the unguided forward trajectory.

D. Training-free Image Editing

BiFlow naturally supports several training-free image editing applications by explicitly modeling a bidirectional mapping between the data and noise domains. We showcase two representative applications: inpainting and class editing. For brevity, we omit the VAE encoder/decoder in the following descriptions, as they always serve as pre-/post-processing steps in our experiments.

Inpainting. The forward model $\mathcal{F}_\theta$ encodes an image x into noise $z = \mathcal{F}_\theta(x)$. We empirically observe that localized perturbations in z predominantly affect corresponding spatial regions in the reconstructed image.

Figure 10. **Class Editing**. BiFlow constructs an explicit bidirectional mapping between images and noise. With this property, BiFlow is able to conduct training-free class editing by modifying only the label condition in the forward and reverse process.

Based on this property, BiFlow enables inpainting with an arbitrary binary mask $\mathcal{M} \in \{0,1\}^{H\times W}$. Given a masked image $x_{\text{mask}} = \mathcal{M} \odot x$, we first map it to the noise domain using the forward model: $z_{\text{mask}} = \mathcal{F}_\theta(x_{\text{mask}})$. We then *resample* the masked portion of the prior as

$$z' = \mathcal{M} \odot z_{\text{mask}} + (1 - \mathcal{M}) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}).$$

Finally, the modified noise z' is mapped back to the image domain by the reverse model $\mathcal{G}_\phi$. This procedure fills the masked region with content coherent with the context. Representative examples are shown in Fig. 9.

Class Editing. The reverse model $\mathcal{G}_\phi$ allows us to generate images from noise z under different class conditions. For a fixed z, changing the class label $\mathbf{c}$ primarily modifies the class-dependent appearance while largely preserving the global spatial structure.

Concretely, given an image x with label $\mathbf{c}$, we obtain its prior variable $z = \mathcal{F}_\theta(x \mid \mathbf{c})$, and reconstruct it using a different label $\mathbf{c}'$, writing $x' = \mathcal{G}_\phi(z \mid \mathbf{c}')$. As illustrated in Fig. 10, BiFlow effectively alters class-specific attributes while maintaining the overall structure, enabling intuitive class editing without retraining.

Efficiency. Both inpainting and class editing require only a single forward pass from data to noise and a single reverse pass from noise to data, making BiFlow a lightweight and efficient tool for training-free image manipulation.

E. Visualizations

We provide *uncurated* 1-NFE generation results of BiFlow-B/2 in Fig. 11 to Fig. 13.

Acknowledgements. We greatly thank Google TPU Research Cloud (TRC) for granting us access to TPUs. Q. Sun, X. Wang, Z. Jiang, and H. Zhao are supported by the MIT Undergraduate Research Opportunities Program (UROP). We thank Zhengyang Geng, Tianhong Li, and our other group members for their helpful discussions and feedback on the draft.

class 12: house finch, linnet, Carpodacus mexicanus

class 81: ptarmigan

class 207: golden retriever

class 279: Arctic fox, white fox, Alopex lagopus

class 309: bee

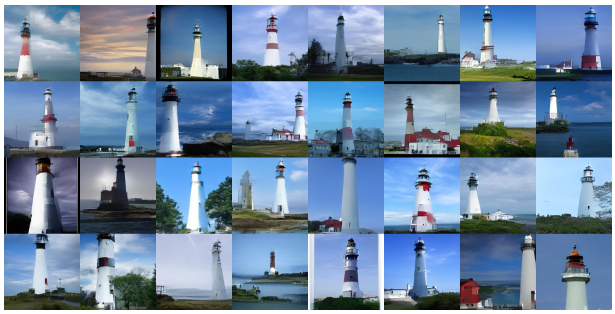
class 323: monarch, monarch butterfly, milkweed butterfly, Danaus plexippus

class 327: starfish, sea star

class 417: balloon

Figure 11. *Uncurated* 1-NFE class-conditional generation samples of BiFlow-B/2 on ImageNet 256×256. CFG scale: 2.0

class 425: barn



class 437: beacon, lighthouse, beacon light, pharos

class 449: boathouse

class 497: church, church building

class 538: dome

class 554: fireboat

class 562: fountain

class 616: knot

Figure 12. *Uncurated* 1-NFE class-conditional generation samples of BiFlow-B/2 on ImageNet 256×256. CFG scale: 2.0

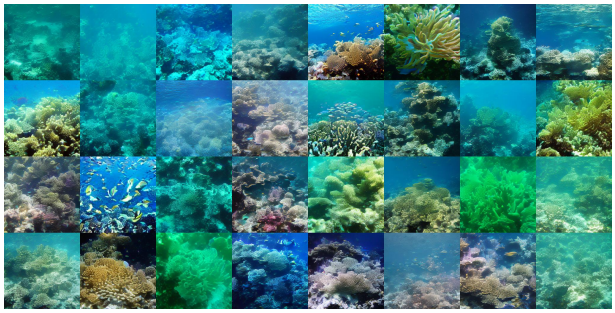
class 646: maze, labyrinth

class 649: megalith, megalithic structure

class 888: viaduct

class 952: fig

class 970: alp



class 973: coral reef

class 975: lakeside, lakeshore

class 985: daisy

Figure 13. *Uncurated* 1-NFE class-conditional generation samples of BiFlow-B/2 on ImageNet 256×256. CFG scale: 2.0