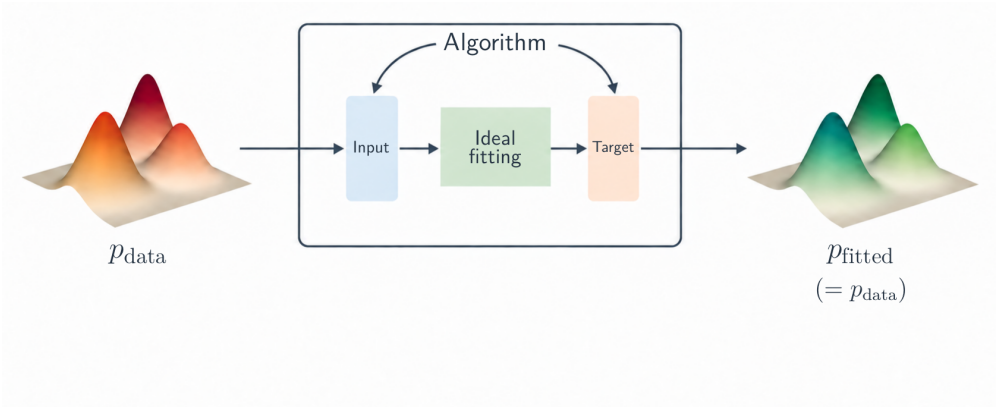
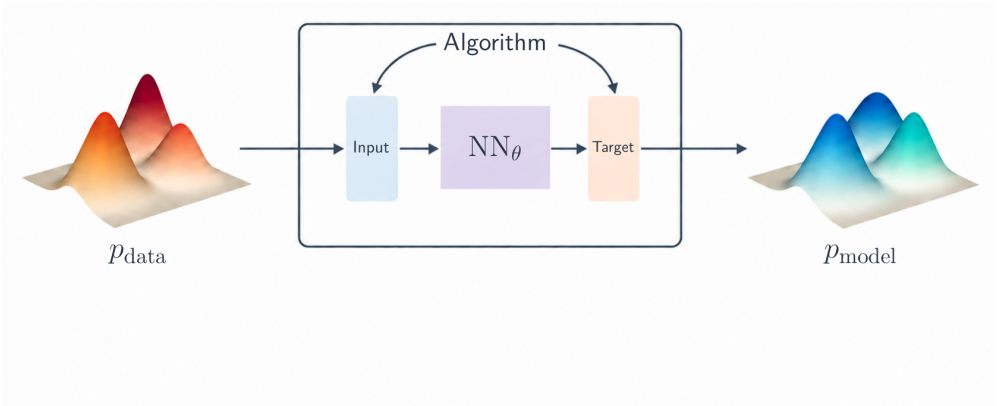
On the Bindings Between Generative Modeling Algorithms and Neural Networks

Hanhong Zhao MIT

A Generative Algorithm Fits Data to a Target

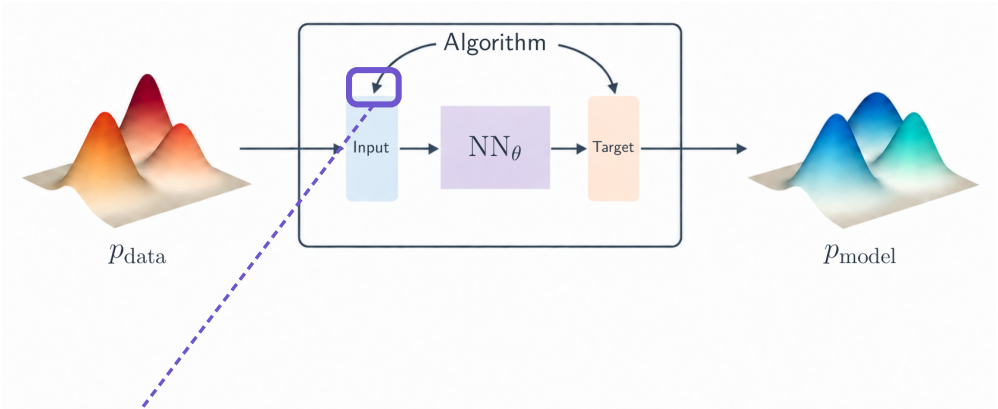


Generative Algorithms Bind to Neural Networks



How do we connect a neural network to a generative modeling algorithm?

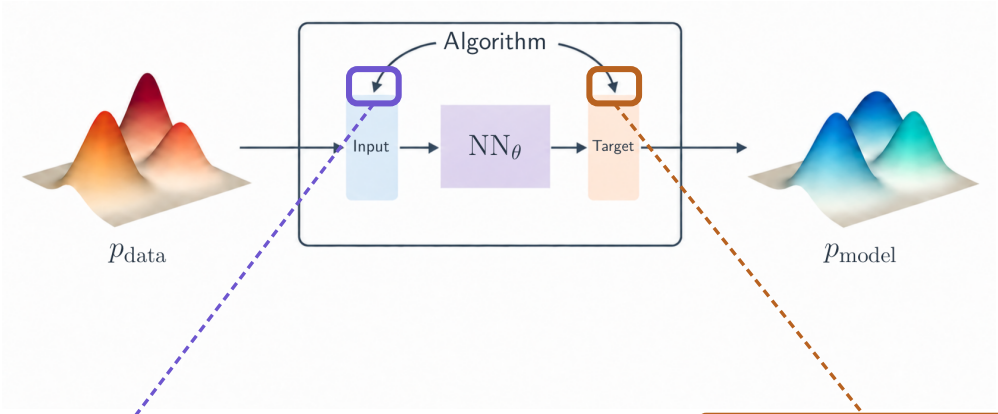
Generative Algorithms Bind to Neural Networks



Input

How is the algorithm's input fed into the model?

Generative Algorithms Bind to Neural Networks



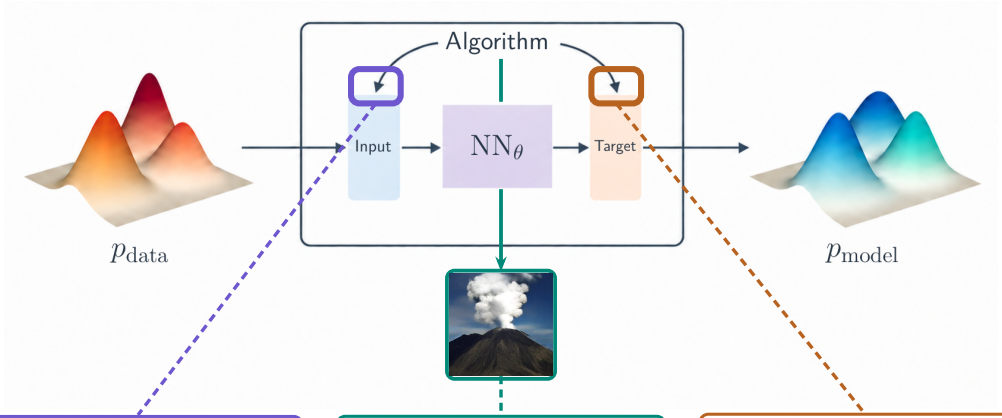
Input

How is the algorithm's input fed into the model?

Output

How is the model's prediction turned into the output?

Generative Algorithms Bind to Neural Networks



Input

How is the algorithm's input fed into the model?

Generation

How the generative process utilizes the network?

Output

How is the model's prediction turned into the output?

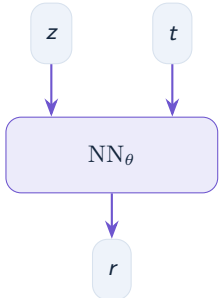
01

Input binding

The algorithm uses the noise level —
must the network receive it?

Q. Sun, Z. Jiang, **H. Zhao**, K. He. *Is Noise Conditioning Necessary for Denoising Generative Models?* arXiv:2502.13129v2.

Diffusion, then dropping the noise-level input

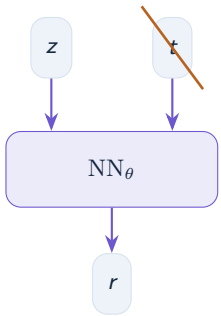


- Noise a data point x into z, then train NN_θ to regress a target r, conditioned on the noise level t:

$$z = a(t)\,x + b(t)\,\epsilon, \qquad r = c(t)\,x + d(t)\,\epsilon$$

$$\mathcal{L}(\theta) = \mathbb{E}_{x,\epsilon,t}\big[\, w(t)\,\|\mathrm{NN}_\theta(z,\,t) - r\|^2 \,\big]$$

Diffusion, then dropping the noise-level input

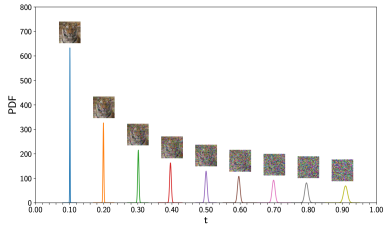


- Noise a data point x into z, then train NN_θ to regress a target r, conditioned on the noise level t:

$$z = a(t)\,x + b(t)\,\epsilon, \qquad r = c(t)\,x + d(t)\,\epsilon$$

$$\mathcal{L}(\theta) = \mathbb{E}_{x,\epsilon,t}\left[\, w(t)\,\|\mathrm{NN}_\theta(z, t) - r\|^2 \,\right]$$

Intuition. $p(t \mid z)$ is sharply peaked: z **already encodes** its own noise level t.



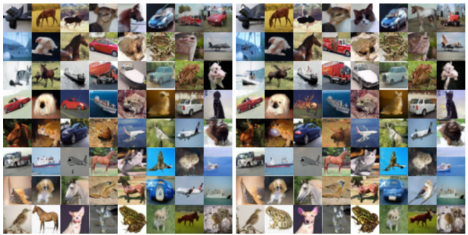
Removing the conditioning

Diffusion does not need to feed t to the network:

$$\mathcal{L}(\theta) = \mathbb{E}_{x,\epsilon,t}\big[\, w(t)\, \|\mathbf{NN}_\theta(\mathbf{z}) - r\|^2 \,\big]$$

Method	with t	without t
EDM	1.99	3.36
Flow Matching	3.01	2.61
uEDM	2.04	2.23
iCT	2.59	3.57
DDIM	3.99	40.90

FID-50k on CIFAR-10



with t (left) vs. without t (right)

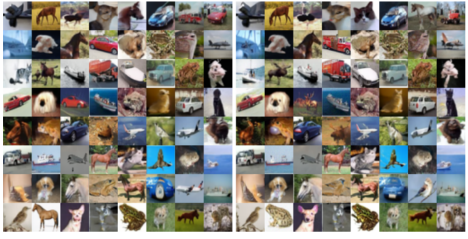
Removing the conditioning

Diffusion does not need to feed t to the network:

$$\mathcal{L}(\theta) = \mathbb{E}_{x,\epsilon,t}\big[\, w(t)\, \|\mathbf{NN}_\theta(\mathbf{z}) - r\|^2 \,\big]$$

Method	with t	without t
EDM	1.99	3.36
Flow Matching	3.01	2.61
uEDM	2.04	2.23
iCT	2.59	3.57
DDIM	3.99	40.90

FID-50k on CIFAR-10



with t (left) vs. without t (right)

What the algorithm knows need not be fed to the network explicitly, if the network can already infer it.

02

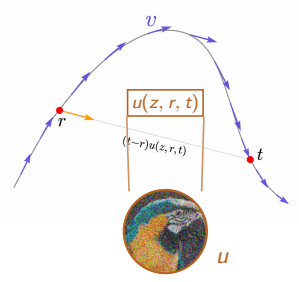
Output binding

The algorithm needs velocity — must the network predict it?

Y. Lu, S. Lu, Q. Sun, **H. Zhao**, Z. Jiang, X. Wang, T. Li, Z. Geng, K. He. *One-step Latent-free Image Generation with Pixel Mean Flows.* arXiv:2601.22158.

One-step in pixel space: make the two sides meet

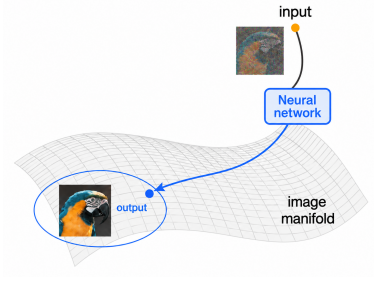
Goal: one-step generation directly in pixel space



Algorithm's u is a noisy velocity

$\neq$

no direct map

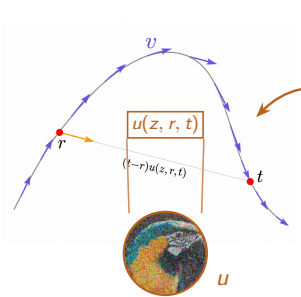


Network outputs pixels

- MeanFlow needs the average **velocity** u between two times r, t.
- However, the neural network **cannot learn** such a high-resolution velocity.

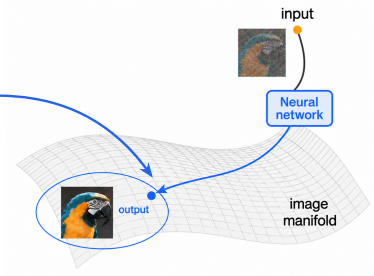
One-step in pixel space: make the two sides meet

Goal: one-step generation directly in pixel space



Algorithm's u is a noisy velocity

$$u = \frac{z_t - x_\theta}{t}$$



Network outputs pixels

Bridge: predict the clean image, then convert to velocity.

- The network predicts x_θ — a clean image on a **low-dimensional** manifold, easy to learn.
- Convert to what the algorithm needs: $u = (z_t - x_\theta)/t$.

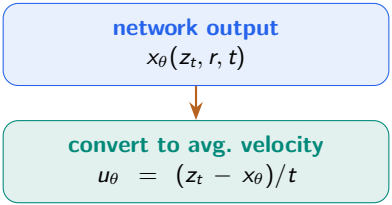
pMF: separate the prediction space from the loss space

x-prediction, velocity-space loss

network output
$x_\theta(z_t, r, t)$

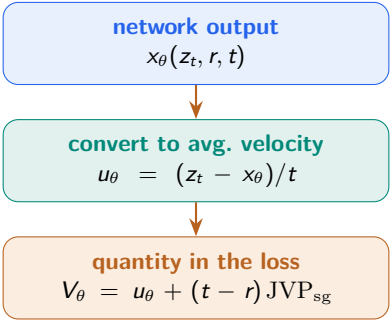
pMF: separate the prediction space from the loss space

x-prediction, velocity-space loss



pMF: separate the prediction space from the loss space

x-prediction, velocity-space loss



$0 \le r \le t \le 1$; the loss compares V_θ to $v = \epsilon - x$; $\mathrm{JVP}_{\mathrm{sg}}$ is a trajectory derivative with stop-gradient.

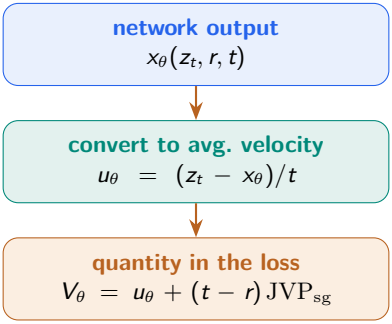
Algorithm 1 pixel MeanFlow: training.

Note: in PyTorch and JAX, `jvp` returns the function output and JVP.

```
# net: x-prediction network
# x: training batch in pixels

t, r = sample_t_r()
e = randn_like(x)

z = (1 - t) * x + t * e

# average velocity u from x-prediction
def u_fn(z, r, t):
   return (z - net(z, r, t)) / t

# instantaneous velocity v at time t
v = u_fn(z, t, t)

# predict u and dudt
u, dudt = jvp(u_fn, (z, r, t), (v, 0, 1))

# compound function V
V = u + (t - r) * stopgrad(dudt)

loss = metric(V, e - x)
```

pMF: separate the prediction space from the loss space

x-prediction, velocity-space loss



$0 \le r \le t \le 1$; the loss compares V_θ to $v = \epsilon - x$; $\mathrm{JVP}_{\mathrm{sg}}$ is a trajectory derivative with stop-gradient.

Algorithm 1 pixel MeanFlow: training.

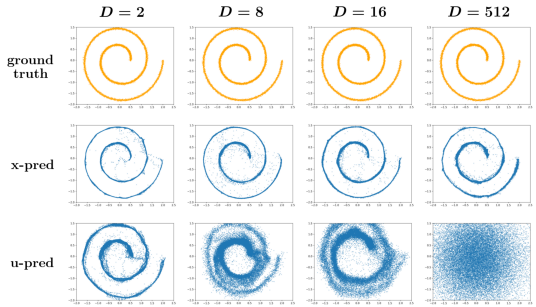
Note: in PyTorch and JAX, `jvp` returns the function output and JVP.

```
# net: x-prediction network
# x: training batch in pixels

t, r = sample_t_r()
e = randn_like(x)

z = (1 - t) * x + t * e

# average velocity u from x-prediction
def u_fn(z, r, t):
   return (z - net(z, r, t)) / t

# instantaneous velocity v at time t
v = u_fn(z, t, t)

# predict u and dudt
u, dudt = jvp(u_fn, (z, r, t), (v, 0, 1))

# compound function V
V = u + (t - r) * stopgrad(dudt)

loss = metric(V, e - x)
```

- Network output is on the image space; the MeanFlow constraint still acts in velocity space.

Prediction target is important in high dimension

Ablation of prediction target: on a Toy setting, and on ImageNet



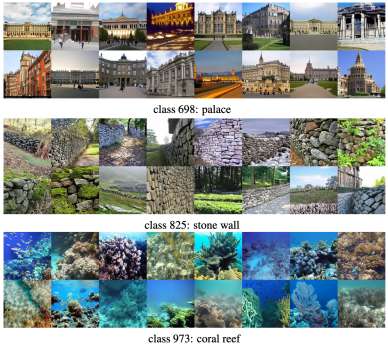
- **Toy experiment.** As D grows, x-prediction keeps the data distribution while u-prediction collapses to noise by D=512.
- **Confirmed on ImageNet** (1-NFE FID↓):

	64^2 (dim 48)	256^2 (dim 768)
x-pred	3.80	**9.56**
u-pred	3.82	**164.89**

System-level comparison

ImgNet 256×256	NFE	space	params	Gflops	FID ↓	IS ↑
Multi-step Latent-space Diffusion/Flow						
DiT-XL/2 [1]	250×2	latent	675M (49M)	119 (310)	2.27	278.2
SiT-XL/2 [2]	250×2	latent	675M (49M)	119 (310)	2.06	277.5
SiT-XL/2 + REPA [3]	250×2	latent	675M (49M)	119 (310)	1.42	305.7
RAE + DiT^{DH}-XL/2 [4]	50×2	latent	839M (415M)	146 (106)	1.13	262.6
Multi-step Pixel-space Diffusion/Flow						
ADM-G [5]	250×2	pixel	554M	1120	4.59	186.7
SiD, UViT [6]	1000×2	pixel	2.5B	555	2.44	256.3
VDM++ [7]	256×2	pixel	2.5B	555	2.12	267.7
SiD2, Flop Heavy [8]	512×2	pixel	-	653	1.38	-
JiT-G/16 [9]	100×2	pixel	2B	383	1.82	292.6
PixelDiT-XL/16 [10]	100×2	pixel	797M	311	1.61	292.7
1-NFE Latent-space Diffusion/Flow						
iCT-XL/2 [11]	1	latent	675M (49M)	119 (310)	34.24	-
Shortcut-XL/2 [12]	1	latent	676M (49M)	119 (310)	10.60	102.7
MeanFlow-XL/2 [13]	1	latent	676M (49M)	119 (310)	3.43	247.5
iMF-XL/2 [14]	1	latent	610M (49M)	175 (310)	1.72	282.0
1-NFE Pixel-space GAN						
BigGAN-deep [15]	1	pixel	56M	59	6.95	171.4
StyleGAN-XL [16]	1	pixel	166M	1574	2.30	260.1
GigaGAN [17]	1	pixel	569M	-	3.45	225.5
1-NFE Pixel-space Diffusion/Flow						
EPG-L/16 [18]	1	pixel	540M	113	8.82	-
pMF-B/16 (ours)	1	pixel	118M	33	3.12	254.6
pMF-L/16 (ours)	1	pixel	410M	117	2.52	262.6
pMF-H/16 (ours)	1	pixel	956M	271	**2.22**	268.8

ImageNet 256^2: pMF is 1-NFE and pixel-space; **pMF-H/16** reaches **2.22** FID.



One-step samples generated directly in raw pixels — no VAE, no distillation.

The network need not directly predict the algorithm's own quantity.

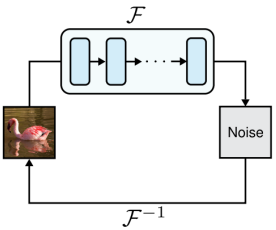
03

Generation binding

Must modeling and generation be the same network?

Y. Lu, Q. Sun, X. Wang, Z. Jiang, **H. Zhao**, K. He. *Bidirectional Normalizing Flow: From Data to Noise and Back.* CVPR 2026 (arXiv:2512.10953v1).

Generation without the analytic inverse



(a) Standard Normalizing Flow: explicit inverse.

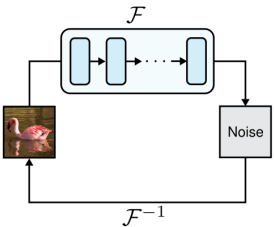
Standard NF generates by inverting $\mathcal{F}$ analytically.

The cost of an exact inverse $\mathcal{F}^{-1}$

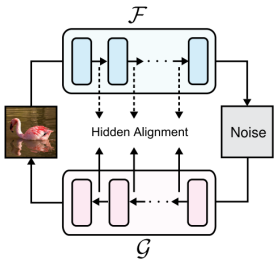
– $\mathcal{F}$ must be explicitly invertible with a tractable Jacobian — no general architecture.

– Generation *is* the causal inverse: many sequential steps, hard to parallelize $\Rightarrow$ slow.

Generation without the analytic inverse



(a) Standard Normalizing Flow: explicit inverse.



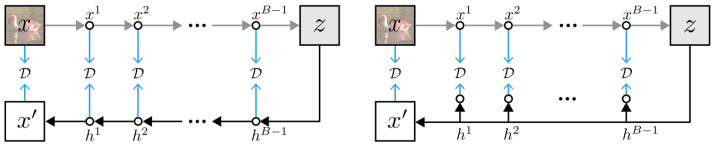
(b) Bidirectional Normalizing Flow: ***learned*** inverse.

Standard NF generates by inverting $\mathcal{F}$ analytically.

BiFlow learns a *separate* reverse net $\mathcal{G}$, tied to $\mathcal{F}$ by **Hidden Alignment** — a non-causal Transformer with 1-NFE generation.

Hidden alignment frees the representation

Learning the reverse network: hidden distillation vs. hidden alignment



(b) Hidden Distillation. (c) Hidden Alignment.

- Learnable projections φ_i align reverse hidden states h^i to regress forward states x^i.

$$\mathcal{L}_{\text{align}}(x) = \sum_i D\big(x^i, \varphi_i(h^i)\big)$$

- Full supervision, and no bottlenecks in modeling

Reverse generation	FID ↓
Exact inverse	44.46
Naive distillation	43.41
Hidden distillation	55.00
Hidden alignment	**36.93**

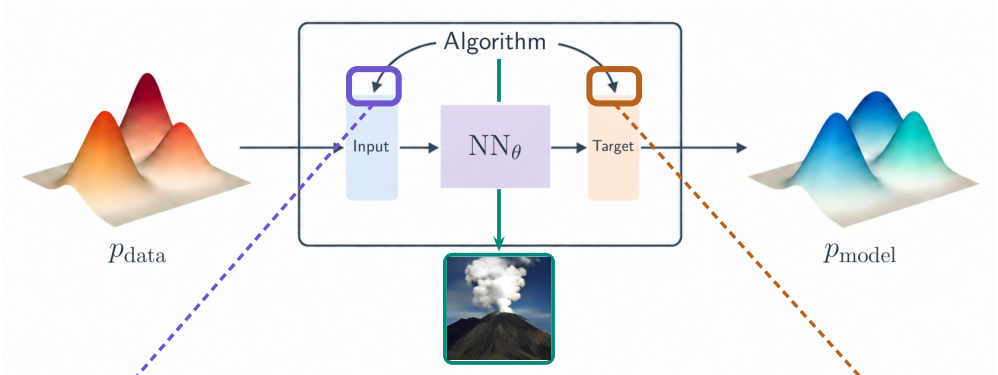
System-level comparison

Method	# Params	NFE	FID(↓)	IS(↑)
Autoregressive Normalizing Flow				
TARFlow-XL/8@pix [65]	1.3B	⋆	5.56	-
STARFlow-XL/1 [21]	1.4B	⋆	2.40	-
Autoregressive Normalizing Flow (our impl.)				
iTARFlow-B/2	120M	⋆	6.83	226.2
iTARFlow-M/2	296M	⋆	5.22	255.5
iTARFlow-L/2	448M	⋆	4.82	254.8
iTARFlow-XL/2	690M	⋆	4.54	259.3
1-NFE Normalizing Flow				
BiFlow-B/2 (Ours)	133M	1	**2.39**	**303.0**

Method	# Params	NFE	FID(↓)	IS(↑)
GANs				
BigGAN-deep [3]	112M	1	6.95	202.6
GigaGAN [26]	569M	1	3.45	225.5
StyleGAN-XL [49]	166M	1	2.30	265.1
1-NFE diffusion/flow matching from scratch				
iCT-XL/2 [52]	675M	1	34.24	-
Shortcut-XL/2 [15]	675M	1	10.60	-
MeanFlow-XL/2 [17]	676M	1	3.43	247.5
TiM-XL/2 [60]	664M	1	3.26	210.3
α-Flow-XL/2+ [67]	676M	1	2.58	-
iMF-XL/2 [18]	610M	1	1.72	282.0
1-NFE diffusion/flow matching (distillation)				
π-Flow-XL/2 [6]	675M	1	2.85	-
DMF-XL/2+ [32]	675M	1	2.16	-
FACM-XL/2 [43]	675M	1	1.76	290.0

The generation network need not be the function the algorithm specifies.

Audit the bindings, free the network design



Input

What the algorithm knows **need not** be an explicit network input.

Generation

Modeling and generation need not be carried out by the **same network**.

Output

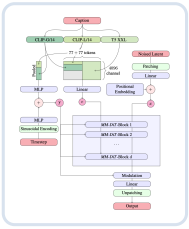
The predicted quantity can **differ from** the one the algorithm uses.

Beyond Image Generation: Which Bindings Are Necessary?

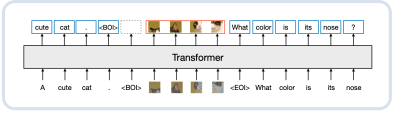
Simplify. Which **components** and interfaces are necessary for text-to-image generation?

Unify. Must understanding and generation use **separate** representations and architectures?

Co-train. Can generative and understanding objectives improve shared **representations?**



SD3 · text-to-image



Transfusion · understand & generate

References

- Q. Sun*, Z. Jiang*, **H. Zhao***, K. He. *Is Noise Conditioning Necessary for Denoising Generative Models?* arXiv:2502.13129.
- Y. Lu*, S. Lu*, Q. Sun*, **H. Zhao***, Z. Jiang, X. Wang, T. Li, Z. Geng, K. He. *One-step Latent-free Image Generation with Pixel Mean Flows.* arXiv:2601.22158.
- Y. Lu*, Q. Sun*, X. Wang*, Z. Jiang, **H. Zhao**, K. He. *Bidirectional Normalizing Flow: From Data to Noise and Back.* CVPR 2026 (arXiv:2512.10953).
- P. Esser*, S. Kulal, A. Blattmann, et al. *Scaling Rectified Flow Transformers for High-Resolution Image Synthesis.* ICML 2024 (arXiv:2403.03206).
- C. Zhou*, L. Yu*, A. Babu, et al. *Transfusion: Predict the Next Token and Diffuse Images with One Multi-Modal Model.* arXiv:2408.11039.