

Bidirectional Normalizing Flow: From Data to Noise and Back

Yiyang Lu^{1,2,*†‡} Qiao Sun^{1,*†} Xianbang Wang^{1,*} Zhicheng Jiang¹ Hanhong Zhao¹ Kaiming He¹

*Equal technical contribution †Project lead

¹MIT ²Tsinghua University

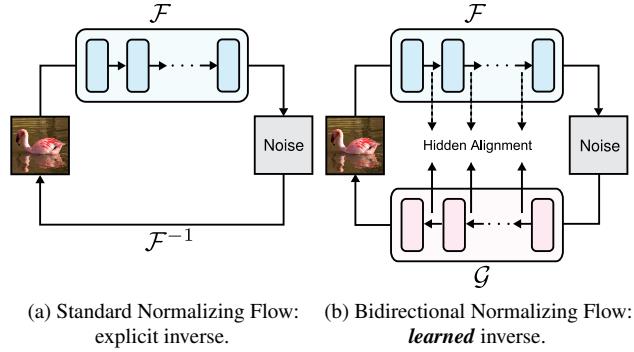
Abstract

Normalizing Flows (NFs) have been established as a principled framework for generative modeling. Standard NFs consist of a forward process and a reverse process: the forward process maps data to noise, while the reverse process generates samples by inverting it. Typical NF forward transformations are constrained by explicit invertibility, ensuring that the reverse process can serve as their exact analytic inverse. Recent developments in TARFlow and its variants have revitalized NF methods by combining Transformers and autoregressive flows, but have also exposed causal decoding as a major bottleneck. In this work, we introduce Bidirectional Normalizing Flow (**BiFlow**), a framework that removes the need for an exact analytic inverse. BiFlow learns a reverse model that approximates the underlying noise-to-data inverse mapping, enabling more flexible loss functions and architectures. Experiments on ImageNet demonstrate that BiFlow, compared to its causal decoding counterpart, improves generation quality while accelerating sampling by up to two orders of magnitude. BiFlow yields state-of-the-art results among NF-based methods and competitive performance among single-evaluation (“1-NFE”) methods. Following recent encouraging progress on NFs, we hope our work will draw further attention to this classical paradigm.

1. Introduction

Normalizing Flows (NFs) are a long-standing family of generative models [45, 10, 30]. They contain two processes: a *forward process* that learns to transform data into noise, and a *reverse process* that generates samples by inverting this transformation. A notable property of NFs is that the underlying flow trajectories from data to noise are *learned* rather than imposed. This differs from their modern continuous-time counterparts [7], such as Flow Matching (FM) [36, 37, 1], whose ground-truth trajectories are predetermined via time-scheduling. However, this advantage of NFs comes at the cost of increased learning difficulty, typically leading to more demanding constraints on forward architectures and objective formulations.

[‡]Work done as an intern at MIT.



(a) Standard Normalizing Flow: (b) Bidirectional Normalizing Flow:
explicit inverse. learned inverse.

Figure 1. Conceptual comparison between standard Normalizing Flows and our proposed Bidirectional Normalizing Flow (BiFlow). Instead of constraining the forward model \mathcal{F} to be explicitly invertible and using its *exact analytic inverse* for generation, BiFlow introduces a learnable reverse model \mathcal{G} that approximates this inverse through our hidden alignment objective. This design frees BiFlow from architectural constraints and enables flexible loss design, allowing for efficient generation with improved quality in a single forward pass.

The standard NF paradigm [45, 10] requires the reverse process to be the exact analytic inverse of the forward process (Fig. 1a). This requirement restricts the range of forward model architectures that can be employed, as the model must be *explicitly invertible* and its Jacobian determinant must be computable, tractable, and differentiable. Existing work on NFs [45, 10, 57, 30, 41, 29] have largely focused on designing compound forward functions that satisfy these requirements. Despite these diverse attempts, NF-based methods remain limited in their ability to use powerful, general-purpose architectures (e.g., U-Nets [47] or Vision Transformers [12]), in contrast to many modern generative model families.

Recently, the gap between NFs and other generative models has been largely closed by TARFlow [65] and its extensions [21]. TARFlow has effectively integrated Transformers [58] with autoregressive flows [30, 41] into the NF paradigm. This design allows NF methods to benefit from the powerful Transformers, substantially mitigating a major limitation of traditional NFs. However, to maintain computable and tractable Jacobian determinants, TARFlow decomposes the forward process into a long chain (e.g., thousands of steps)

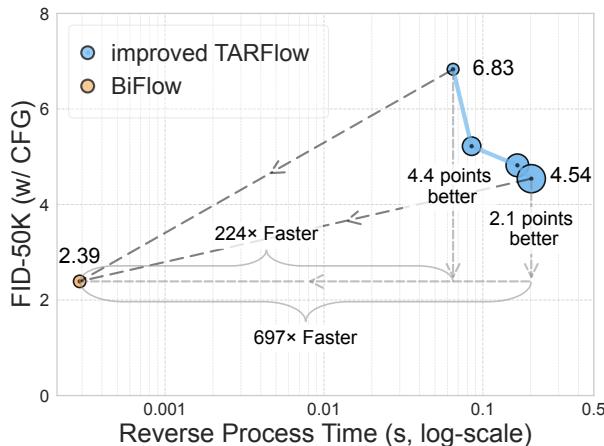


Figure 2. BiFlow surpasses our improved TARFlow baseline by a wide margin in generation quality, despite using a base-size model versus an extra-large model, and it achieves markedly faster sampling as well. The x -axis denotes the wall-clock time (log scale) for generating one image on 8 v4 TPU cores. VAE decoding is omitted from this figure; comprehensive inference cost comparison appears in Tab. 3.

of autoregressive operations. The resulting explicit inverse therefore requires a large number of causal steps at inference time, which is difficult to parallelize. This design not only slows down sampling, but also retains the undesirable architectural constraints during inference, *e.g.*, the reverse model cannot perform feedforward, non-causal attention.

In this work, we introduce *Bidirectional Normalizing Flow (BiFlow)*, a framework in which both the forward and reverse processes are learned. In our framework, the designs of the forward and reverse processes are *decoupled*: the forward process can be any NF model \mathcal{F}_θ that is computable, tractable, and easy to learn (*e.g.*, an improved TARFlow), while the reverse process learns a separate model \mathcal{G}_ϕ to approximate its inverse (Fig. 1b). In contrast to the explicit inverse, our reverse model is highly flexible: it can be a feedforward, non-causal Transformer that is both expressive and efficient to run, naturally enabling high-quality, single function evaluation (1-NFE) generation.

Learning the reverse model \mathcal{G}_ϕ is not merely a form of distillation, even though we use a pre-trained forward model \mathcal{F}_θ : in fact, our learned reverse model \mathcal{G}_ϕ can *outperform* the explicit inverse of \mathcal{F}_θ . Compared to distilling the noise-to-data trajectories, we find that *aligning* the intermediate hidden states yields results even better than the explicit inverse. In addition, our learnable reverse model can naturally eliminate the extra step of score-based denoising in TARFlow, simplifying and accelerating inference while improving quality. Such a “what-you-see-is-what-you-get” property further enables the use of *perceptual* loss [68], which is impossible or difficult to leverage with an explicit inverse. Putting these factors together, our learned reverse model can substantially outperform its explicit-inverse counterpart.

We report competitive results on the ImageNet 256×256 generation. Comparing with an improved TARFlow (which is also the forward model for BiFlow), BiFlow achieves an FID of 2.39 using a DiT-B size [42] model, while being two orders of magnitude faster (see Fig. 2; detailed in Tab. 3). This not only sets a new state-of-the-art result among NF-based methods, but also represents a strong 1-NFE result in comparison with other generative model families.

Following the progress established by TARFlow and extensions, our work on BiFlow further unleashes the potential of NFs as a strong competitor among modern generative model families. Our findings indicate that the NF principle of learning the forward trajectories, rather than pre-scheduling them, can be advantageous and need not introduce inference-time limitations. Considering that modern Flow Matching methods are continuous-time NFs with pre-scheduled trajectories, we hope our study will shed light on the potential synergy among these related methods.

2. Related Work

Normalizing Flows. Normalizing Flows (NFs) have long served as a principled framework for probabilistic generative modeling. Over the past decade, extensive research has focused on enhancing the expressivity and scalability of NFs under the constraint of invertible transformations. Planar flows [45] and NICE [10] pioneered the use of simple reversible mappings to construct deep generative models. Real NVP [11] and Glow [29] extended this framework with non-volume-preserving transformations and convolutional architectures. IAF [30] and MAF [41] introduced autoregressive flows to improve expressivity while maintaining tractable likelihoods. TARFlow [65] and STARFlow [21] further revitalized the NF family by incorporating Transformer into autoregressive flows. They demonstrated significant gains in generation quality and scalability, reaffirming NFs as a competitive paradigm in modern generative modeling.

Despite these advances, standard NFs still inherit limitations from their *invertibility* requirement. In particular, autoregressive flow formulations impose strict causal ordering and sequential dependencies, which constrain architectural design and lead to slow inference.

Continuous Normalizing Flows. Continuous Normalizing Flows (CNFs) [20, 14, 19] generalize discrete flows by modeling transformations as continuous-time dynamics governed by ordinary differential equations (ODEs) [7]. CNFs enable more flexible architectures and tractable likelihood computation via numerical ODE simulations. FM [36, 37, 13] reformulates the explicit maximum-likelihood training objective into an equivalent implicit objective. Diffusion models [25, 51, 9] can be interpreted as a special case of Flow Matching with stochastic dynamics, achieving impressive fidelity and scalability. Despite their empirical success, the

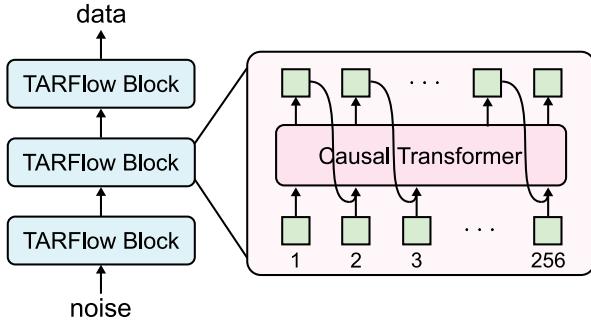


Figure 3. Illustration on the **autoregressive inference** process of TARFlow. In each block, each token is transformed one by one, depending on previous tokens. This is repeated for a sequence with length 256 for a 32×32 input with patch size 2, and is further repeated for all blocks (e.g., 8 blocks). Altogether, TARFlow inference requires 8×256 sequential function evaluations.

implicit formulation of FM and diffusion models sacrifices the learnable bidirectional mapping that characterizes NFs.

3. Background: Normalizing Flows

Normalizing Flows (NFs) are a class of generative models that establish a bijective transformation between a Gaussian prior distribution p_0 and a complex data distribution p_{data} . An NF consists of a forward process and a reverse process. Given a data sample $x \in \mathbb{R}^D \sim p_{\text{data}}$, the forward process \mathcal{F} maps it into the Gaussian prior space $z = \mathcal{F}(x)$. The model assigns the data likelihood $p(x)$ through the *change-of-variables* formula. Training is performed by optimizing \mathcal{F} to maximize the log-likelihood $\log p(x)$ over data samples.

Classical NF requires the forward process \mathcal{F} to be explicitly invertible for exact likelihood computation and efficient sampling. Once trained, its exact inverse, \mathcal{F}^{-1} , can be used for generation by transforming Gaussian noise back to the data space, *i.e.*, $x = \mathcal{F}^{-1}(z)$ where $z \sim p_0$.

In practice, to enhance expressiveness, the forward process is commonly constructed as a composition of multiple simpler bijective transformations $\mathcal{F} := f_{B-1} \circ \dots \circ f_1 \circ f_0$ (\circ denotes function composition). Under this formulation, the log-likelihood objective becomes

$$\log p(x) = \log p_0(z) + \sum_i \log \left| \det \frac{\partial f_i(x^i)}{\partial x^i} \right|, \quad (1)$$

with $x^0 = x$ and $x^{i+1} = f_i(x^i)$. Here, $\det(\cdot)$ denotes the determinant operator. Designing transformations that yield *computable* and *differentiable* determinant has been a key consideration in prior NF formulations. This requirement motivates specialized designs such as affine coupling [10, 11] and autoregressive flows [30, 41], which preserve tractable Jacobians.

Importantly, while the log-determinant term in Eq. (1) requires the forward process \mathcal{F} to be *invertible*, it does not necessitate an *explicitly invertible* formulation. The explicit

inverse is only required at *inference* time, where we need to map samples from prior back to the data space.

TARFlow. TARFlow [65] integrates Transformer architectures into autoregressive flows (AF), substantially improving their expressiveness and scalability. The core idea in AF is to further decompose each sub-transformation f_i , parameterized by a block, into T steps, where T denotes the sequence length of the input tokens. Each step transforms the i -th token only conditioned on its predecessors, which can naturally be realized through Transformer layers with causal masks. To capture bidirectional context, AF flips the sequence order in alternating blocks. By combining expressive Transformer architectures with autoregressive flows, TARFlow successfully revives NF to remain competitive with today’s state-of-the-art generative models.

However, AF parameterization introduces asymmetry between training and sampling. Similar to next-token-prediction language models, although likelihood evaluation and training can be parallelized efficiently, sampling must proceed *sequentially* due to the autoregressive nature, as illustrated in Fig. 3. In practice, this requires performing, *e.g.*, thousands of (8×256) inverse transformations *one after another*, resulting in substantial inference latency.

4. Bidirectional Normalizing Flow

We propose a Bidirectional Normalizing Flow (BiFlow) framework, which has: (i) a forward model \mathcal{F}_θ that transforms data samples into pure noise, and (ii) a *learnable*, separate reverse model \mathcal{G}_ϕ that approximates its inverse, mapping noise back to the data space. Training is performed in two stages: first, similar to classical NF, we train the forward model using maximum likelihood estimation; then, keeping the forward model fixed, we train the reverse model to approximate its inverse mapping.

Notably, our reverse model \mathcal{G}_ϕ is *not* constrained by explicit invertibility. As a result, this allows us to design the reverse model with arbitrary architectures (*e.g.*, bidirectional attention-based Transformers) and training objectives. Next, we discuss the formulation, objectives, and learning dynamics of the reverse process.

4.1. Learning to Approximate the Inverse

Given a pre-trained forward model \mathcal{F}_θ , our goal is to optimize a reverse model \mathcal{G}_ϕ that approximates its inverse. We consider three strategies: (i) naive distillation; (ii) hidden distillation; (iii) hidden alignment, as approaches to learning the reverse model. Fig. 4 illustrates the differences among these methods, as we describe next.

Naive Distillation. A straightforward strategy is to impose a direct distillation loss:

$$\mathcal{L}_{\text{naive}}(x) = \mathcal{D}(x, x'),$$

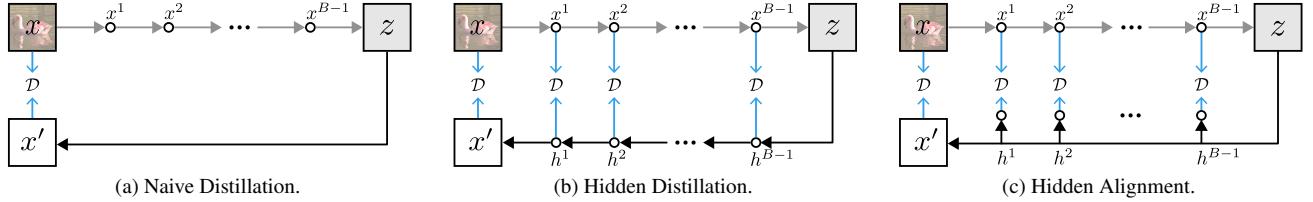


Figure 4. Comparison of three approaches for learning the reverse process. Each \circ marks a position where the model returns to the same dimension as input x . Blue arrows with \mathcal{D} refer to distance loss terms. Our hidden alignment strategy (Fig. 4c) combines the strengths of Fig. 4a and Fig. 4b, leveraging the entire trajectory for supervision without repeatedly returning to input space.

where x is a data sample, $x' = \mathcal{G}_\phi(\mathcal{F}_\theta(x))$ is the reconstructed data, and \mathcal{D} denotes a distance metric (*e.g.*, L2 distance). The reverse model is trained to minimize the reconstruction error on data samples (see Fig. 4a).

This simple approach provides supervision only at the *final* output, which may be insufficient for effectively training the reverse model. Directly mapping pure noise to data in one step is highly under-constrained, making it difficult for the reverse network to learn a reliable inverse from a single reconstruction loss.

Hidden Distillation. A typical NF is composed of a sequence of simple sub-transformations, *i.e.*, $\mathcal{F}_\theta = f_{B-1} \circ \dots \circ f_1 \circ f_0$, where each f_i is a transformation block and B denotes the total number of blocks. We can strengthen the training signal by leveraging the full sequence of intermediate states generated along the forward trajectory.

As illustrated in Fig. 4b, starting from $x \sim p_{\text{data}}$, the forward model produces a trajectory of intermediate hidden states $\{x^i\}$ with $z = \mathcal{F}_\theta(x)$ as the final output prior. Analogously, we also design the reverse model to be composed of B blocks, generating a reverse trajectory $\{h^i\}$ from z . We distill the reverse model by enforcing the two trajectories to be close. Formally, the loss is defined as:

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

where h^0 corresponds to the reconstructed output x' . Optionally, each term can be assigned a distinct weighting factor. This formulation encourages the reverse model to invert each sub-transformation individually, which could help guide the reverse model to invert the mapping \mathcal{F}_θ step by step. The intermediate hidden states $\{x^i\}$ serve as auxiliary supervision for learning the correspondence between x and z .

Although this hidden distillation strategy provides more supervision than naive distillation, it introduces structural constraints on model design. Since each intermediate state x^i has the same dimensionality as the input, the reverse model is forced to repeatedly project features down to the input space and then back up into the hidden space. This rigid requirement restricts architectural flexibility, ultimately limiting the model’s effectiveness.

Hidden Alignment. We propose a more flexible strategy,

termed hidden alignment. Crucially, it leverages the full forward trajectory for supervision while relaxing the restrictive requirement in hidden distillation that intermediate hidden states must lie in the input space.

As shown in Fig. 4c, we extract intermediate hidden states $\{h^i\}$ from the reverse model \mathcal{G}_ϕ . Unlike hidden distillation, which enforces each h^i to directly match its input-space counterpart x^i , we introduce a set of learnable projection heads $\{\varphi_i\}$ to align the projected representations $\varphi_i(h^i)$ with the corresponding forward states x^i . The training objective then becomes:

$$\mathcal{L}_{\text{align}}(x) = \sum_i \mathcal{D}(x^i, \varphi_i(h^i)), \quad (2)$$

where $h^0 = x'$ and φ_0 is the identity mapping.

This simple modification allows the reverse model to benefit from full trajectory supervision while maintaining architectural and representational flexibility. By decoupling the representation space from the input token space, hidden alignment avoids the potential semantic distortion caused by repeated projections.

4.2. Eliminating Score-based Denoising

Existing state-of-the-art NFs such as TARFlow [65] deviate from standard flow-based modeling in that they learn a noise-perturbed distribution and then denoise the output. Specifically, during training, TARFlow takes a noise-perturbed input $\tilde{x} = x + \sigma\epsilon$, where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and during inference, TARFlow first generates $\tilde{x} = \mathcal{F}_\theta^{-1}(z)$, then performs an additional *score-based denoising* step:

$$x \leftarrow \tilde{x} + \sigma^2 \nabla_{\tilde{x}} \log p(\tilde{x}), \quad (3)$$

as illustrated in Fig. 5a, where the score term is computed via a forward-backward pass. This post-processing almost doubles the inference cost, becoming a clear computational bottleneck for efficient generation.

Learned Denoising. We eliminate the explicit score-based denoising step by integrating denoising directly into the reverse model. As illustrated in Fig. 5b, we extend the forward trajectory from \tilde{x} to z by appending the clean data x at its start, and extend the reverse model with one additional block $h^0 \rightarrow x'$ that learns denoising jointly with the inverse. The

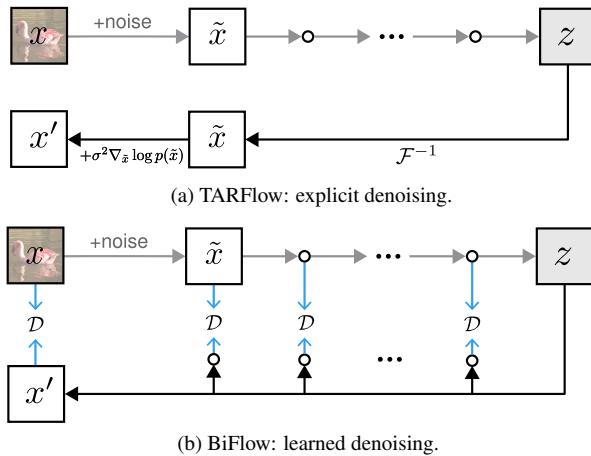


Figure 5. Incorporating the **denoising step** into our hidden alignment framework. The reverse model is extended with an additional block dedicated to denoising. Our learned denoising eliminates the need for calculating the score function through a whole forward-backward pass, incurring only a single additional block forward.

resulting reverse network, with one extra block for denoising, maps z to a clean sample x' in a single pass. As such, our reverse model directly learns the *correspondence* between z and the *clean data* x , rather than the noisy data \tilde{x} .

The training process follows the same objective as Eq. (2), with a reconstruction loss on (x, x') and hidden alignment losses on intermediate states. By integrating denoising into the reverse process itself, BiFlow achieves a unified learned formulation for generation, where inverse and denoising are seamlessly coupled within a single direct generative model, eliminating the need for any extra refinement step.

4.3. Distance Metric

BiFlow provides a flexible supervised-learning framework for tackling the generation problem. This flexibility stems from two key properties of BiFlow: (i) *1-NFE generation* — the learned reverse model produces a sample x' in a single forward pass, so generated samples are directly accessible during training; and (ii) *explicit pairing* — the forward process establishes a direct correspondence between data x and noise z , serving as training pairs for the reverse model. Together, these properties realize a *what-you-see-is-what-you-get* training regime: generated samples are available for immediate loss evaluation and backpropagation, enabling rich semantic supervision signals.

Our framework is highly flexible in the choice of loss functions: almost *any distance metric* can be used, and multiple metrics can be combined. Our default choice for the distance metric \mathcal{D} in Eq. (2) is simply mean squared error (MSE). To enhance realism, we further apply perceptual loss at the final VAE-decoded image, while intermediate hidden states remain aligned by MSE. In this work, we adopt both VGG [50] and ConvNeXt V2 [61] feature spaces for per-

ceptual loss (our implementation for VGG features follows LPIPS [68]). As in prior work [16, 52, 17], all loss terms can be adaptively re-weighted during training. Details are provided in Appendix B.3.

4.4. Norm Control

The intermediate states produced by the forward model are unconstrained under the NF formulation, often exhibiting large norm fluctuations across blocks (see Fig. 8a). These variations can lead to imbalanced supervision when using magnitude-sensitive losses such as MSE for reverse-model training. To mitigate this issue, we introduce two complementary norm-control strategies applied to the forward and reverse models to ensure stable and consistent supervision strength (details in Appendix B.3).

On the forward model, we clip the output parameters of each transformation f_i within a fixed range $[-c, c]$, limiting excessive scaling and stabilizing intermediate state norms without compromising expressiveness. On the reverse model, we normalize each intermediate state before performing hidden alignment, which equalizes the contribution across trajectory depth and promotes scale-invariant learning.

4.5. BiFlow with Guidance

Classifier-free guidance (CFG) [24] was originally proposed for diffusion models to control the trade-off between sample diversity and fidelity. Due to its effectiveness, it has been widely adopted in diffusion-based generative models. Following this success, recent Normalizing Flows [65, 21] and autoregressive models [56, 35] also incorporate CFG to further improve generation quality.

CFG can be seamlessly integrated into BiFlow’s inference process by extrapolating conditional and unconditional predictions of \mathcal{G}_ϕ at each hidden state h^i , *i.e.*,

$$h^{i+1} = (1 + w_i) \mathcal{G}_\phi^i(h^i | \mathbf{c}) - w_i \mathcal{G}_\phi^i(h^i), \quad (4)$$

where \mathbf{c} is the class condition and w_i is the guidance scale (our w definition follows the original CFG formulation [24], *i.e.*, $w = 0$ is w/o CFG). The subscript i indicates that w_i can differ among blocks, supporting CFG interval [31]. More results are provided in Appendix C.2.

Directly applying CFG doubles the computational cost during inference, since each guided block requires two forward passes. To alleviate this, following [5, 55], we incorporate CFG into the training stage, enabling inference with only one function evaluation (1-NFE) while preserving the benefits of guidance. Additionally, to retain the flexibility of adjusting guidance scales at inference time, we allow the reverse model to leverage CFG scale as *condition* [39, 18]. By training the model with a range of guidance scales, BiFlow can generate outputs corresponding to various guidance strengths within a single forward pass. Further details are provided in Appendix B.2.

5. Experiments

Experiment Settings. Our experiments are conducted on class-conditional ImageNet [8] generation at 256×256 resolution. We evaluate Fréchet Inception Distance (FID) [23] and Inception Score (IS) [48] on 50000 generated images. Following [46, 13, 21], we implement our models on the latent space of a pre-trained VAE tokenizer. For ImageNet 256×256 , the tokenizer maps images to a $32 \times 32 \times 4$ latent representation, serving as the input and output domain of our models.

Improved TARFlow as Baseline. Our BiFlow framework builds upon TARFlow [65] as our forward model. We introduce several modifications to the original TARFlow to enhance stability and performance. Specifically, we replace additive conditioning with in-context conditioning [42] and apply the norm control strategy in Sec. 4.4, while omitting STARFlow-specific components such as deep-shallow design, decoder finetuning, and customized CFG. We denote this enhanced version as *improved TARFlow* (iTARFlow). As shown in the table below, it achieves substantial gains over the original TARFlow, both with or without CFG, establishing a strong *baseline* for BiFlow.

Method	FID (\downarrow)		
	w/o CFG	w/ CFG	# Params
latent TARFlow-B/2 *	59.43	10.89	118M
+ in-context conditioning	53.87	8.25	120M
+ 160 epochs \rightarrow 960 epochs	45.48	7.05	120M
+ norm control (iTARFlow)	44.46	6.83	120M

Configurations. Our reverse model adopts a ViT backbone with modern Transformer components [53, 66] and multi-token in-context conditioning [18]. We name our model as BiFlow-B/2, where B/2 indicates a base-sized model with patch size 2, resulting in a sequence length of 256. In our ablation studies, we choose an iTARFlow as our forward model and train the reverse model with the forward model fixed. Unless otherwise specified, our ablations employ the adaptive-weighted MSE, while final comparisons in Tab. 4 incorporate perceptual distance mentioned in Sec. 4.3 for optimal performance. Details are provided in Appendix A.

5.1. Ablation: Learning to Approximate the Inverse

We evaluate three strategies for learning the reverse model, as described in Sec. 4.1, and report generation quality (FID in Tab. 1) as well as reconstruction error (see Appendix C.1).

The naive distillation approach, trained with a simple MSE objective, already *outperforms* the exact inverse baseline, indicating that a learned reverse model is a *practical and competitive* alternative to the analytic inverse.

Hidden distillation supervises the reverse model using the entire forward trajectory. However, repeated projections

	FID (\downarrow)	attention
exact inverse	44.46	causal
naive distillation	43.41 <small>-1.05</small>	bidirect
hidden distillation	55.00 <small>+10.54</small>	bidirect
hidden alignment	36.93 <small>-7.53</small>	bidirect

Table 1. **Reverse learning method.** Naive distillation can *exceed* the exact inverse with a simple MSE objective. Our hidden alignment yields the best result among the three strategies. (Settings: BiFlow-B/2, 160 epochs, adaptive weighted MSE loss, w/o CFG)

between representation and input spaces cause information loss and limit architectural expressiveness. This results in degraded performance compared to the naive distillation.

Our proposed hidden alignment method removes the repeated projections inherent in hidden distillation while retaining full trajectory-level supervision, thereby preserving both architectural flexibility and representational richness. It achieves the best performance among the three strategies and surpasses the exact inverse by a clear margin in generation quality. These results collectively demonstrate that hidden alignment is an effective and robust strategy for learning an approximate inverse in BiFlow.

5.2. Other Ablations

We ablate several key design choices in BiFlow and analyze their impact on performance in Tab. 2.

BiFlow with Guidance. BiFlow is conditioned on the CFG scale and learns across a range of CFG scales during training. This enables 1-NFE inference while preserving the benefits and flexibility of guidance. As shown in Tab. 2a, compared to standard CFG approach, our training-time CFG mechanism reduces inference cost by half while achieving better FID.

Learned Denoising. Tab. 2b demonstrates the effectiveness of our *learned* denoising strategy. By jointly training denoising with the inverse, our learned one-block denoiser improves generation quality over the score-based denoising used in TARFlow. Moreover, our approach introduces only a single additional block, whereas TARFlow’s score-based denoising requires an extra forward-backward pass (incurring $15.8 \times$ flops). This substantially reduces inference overhead.

Norm Control. We introduce two norm control strategies in Sec. 4.4 and evaluate their effectiveness in Tab. 2c. Applying either strategy alleviates imbalance in MSE loss across blocks, thereby enhancing performance. We provide visualizations of the norm statistics in Appendix C.4.

Distance Metric. Our framework supports various distance metric designs. As shown in Tab. 2d, incorporating perceptual distance [68, 61] at the image end can largely improve generation quality. Notably, when both VGG and ConvNeXt features are used for the perceptual loss, the optimal guidance scale in Eq. (4) for this model is close to 0.0, resulting in performance similar to no-CFG setting. This suggests these features already provide strong class-discriminative information. More results are provided in Appendix C.3.

*The latent TARFlow-B/2 is our TARFlow reproduction in VAE latent.

	FID, w/o CFG	FID, w/ CFG
inference-time CFG	36.93 (NFE=1)	6.90 (NFE=2)
→ training-time CFG	31.88 (NFE=1)	6.79 (NFE=1)

(a) **BiFlow with guidance.** Conditioning on the CFG scale during training improves FID both w/ and w/o CFG while preserving flexible 1-NFE inference. The baseline w/o CFG is final results in Tab. 1.

	FID, w/o CFG	FID, w/ CFG
learned denoise	31.88	6.79
→ no denoise	100.51	26.20
→ score-based denoise	42.62	10.98

(b) **Learned denoising.** Our learned denoising scheme is effective. Compared to score-based denoising in TARFlow, it eliminates an extra forward-backward calculation, and unifies the denoising step into our framework.

	FID, w/o CFG	FID, w/ CFG
norm control: clip	31.88	6.79
norm control: none	45.54	12.33
norm control: traj.	34.88	8.03

(c) **Norm control.** Either clipping the forward model’s output or normalizing the forward trajectory improves generation quality by ensuring balanced supervision strength across blocks.

	FID, w/o CFG	FID, w/ CFG
MSE	31.88	6.79
+ LPIPS	14.15	4.91
+ LPIPS + ConvNeXt	2.46	2.46

(d) **Distance metric.** Our framework enables a flexible design of distance metrics. Incorporating perceptual distance improves generation quality.

Table 2. **Ablation study on ImageNet 256×256 generation.** FID-50K with 1-NFE is reported by default. (Settings: **BiFlow-B/2**, 160 epochs. By default: adaptive weighted MSE loss without perceptual loss, training-time CFG.)

Scaling Behavior. We investigate the scaling behavior of BiFlow under different distance metrics, using iTARFlow of corresponding size as forward models. We summarize preliminary results in the table below.

FID, w/ CFG	B	XL
MSE	6.79	4.61
+ LPIPS	4.91	3.36
+ LPIPS + ConvNeXt	2.46	2.57

Overall, BiFlow exhibits clear gains from increased model capacity when trained *without* the ConvNeXt-based perceptual loss. However, after incorporating ConvNeXt features, further scaling yields diminishing returns, with FID improvements gradually saturating. We hypothesize this behavior may be related to overfitting, as evidenced by an increase in FID during training. A comprehensive investigation of BiFlow’s scaling behavior is left for future work.

5.3. BiFlow vs. improved TARFlow

We compare our learned reverse model (BiFlow) with the exact analytic inverse baseline (improved TARFlow) of the forward process. In Tab. 3, we benchmark in terms of generation quality (FID score) and inference efficiency (flops and wall-clock time for generating a single image). Details of our benchmarking setup are provided in Appendix A.

	BiFlow B/2	B/2	improved TARFlow M/2	L/2	XL/2
FID	2.39	6.83	5.22	4.82	4.54
# Params	133M	120M	296M	448M	690M
Gflops	38	152	363	552	836
Wall-clock time (ms)					
TPU	0.29±1.3	65±1.3	85±1.3	165±1.3	202±1.3
GPU	2.15±2.7	129±2.7	208±2.7	349±2.7	400±2.7
CPU	80±240	9040±240	16200±240	20400±240	26300±240
Wall-clock speedup, BiFlow-B/2 vs. iTARFlow: (VAE excluded, see also Fig. 2)					
TPU	-	224×	293×	569×	697×
GPU	-	60×	97×	162×	186×
CPU	-	113×	203×	255×	329×
Wall-clock speedup, BiFlow-B/2 vs. iTARFlow: (VAE included)					
TPU	-	42×	54×	105×	128×
GPU	-	27×	43×	73×	83×
CPU	-	29×	51×	65×	83×

Table 3. **Comparison between BiFlow and iTARFlow baseline.** We report both generation quality (FID-50K) and inference cost per image. All wall-clock time measurements are reported as “generator + VAE decoding”. Compared to iTARFlow, BiFlow achieves one to two orders of magnitude faster sampling on TPU, GPU, and CPU, while attaining superior generation quality. (The VAE decoder contains 49M parameters and requires 308 Gflops.)

Experiments show that our BiFlow-B/2 surpasses the exact inverse of the improved TARFlow-XL/2 baseline in generation quality. Remarkably, BiFlow requires only a single function evaluation (1-NFE), compared to 256×2 sequential decoding steps for the autoregressive inference of the exact analytic inverse — resulting in up to a 42× speedup for models of similar size on TPU.

Why can a learned inverse outperform the exact inverse? Our reverse model \mathcal{G}_ϕ is trained to reconstruct real images directly, rather than to replicate synthetic samples produced by the exact inverse as in conventional distillation. This encourages its predictions to align more closely with the true data distribution. In addition, \mathcal{G}_ϕ is optimized *end-to-end* with the forward map fixed, learning to directly transform noise into clean data. This *joint* optimization can help the model to learn a stable and globally consistent mapping.

Why is a learned inverse significantly faster than the exact inverse? From an algorithmic perspective, two key improvements reduce the computational cost of BiFlow. First, BiFlow eliminates the score-based denoising step required by the exact inverse of TARFlow, removing a major computational bottleneck. Second, we integrate CFG into the training stage, effectively halving the inference cost compared to applying CFG during sampling. Together, these two improvements reduce the flops by roughly 4×.

From an architectural perspective, the autoregressive design of TARFlow imposes inherent limitations on parallelism during inference. Our bidirectional attention Transformer design allows for *fully parallelized* computation across the sequence dimension, which leads to significant speedups on modern accelerators. Notably, due to the efficiency of BiFlow, the VAE decoder has become a dominant computational overhead, which is outside the scope of this work.

Method	# Params	NFE	FID(\downarrow)	IS(\uparrow)
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
I-NFE Normalizing Flow				
BiFlow-B/2 (Ours)	133M	1	2.39	303.0
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
I-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
I-NFE diffusion/flow matching (distillation)				
π -Flow-XL/2 [6]	675M	1	2.85	-
DMF-XL/2+ [32]	675M	1	2.16	-
FACTM-XL/2 [43]	675M	1	1.76	290.0
autoregressive/masking				
MaskGIG [4]	227M	*	6.18	182.1
RCG, conditional [34]	512M	*	2.12	267.7
VAR-d30 [56]	2.0B	*	1.92	323.1
MAR-H [35]	943M	*	1.55	303.7
RAR-XXL [63]	1.5B	*	1.48	326.0
xAR-H [44]	1.1B	*	1.24	301.6
Mult-I-NFE diffusion/flow matching				
ADM-G [9]	554M	250×2	4.59	-
LDM-4-G [46]	400M	250×2	3.60	247.7
DiT-XL/2 [42]	675M	250×2	2.27	278.2
SiT-XL/2 [38]	675M	250×2	2.06	252.2
JiT-G/16 [33]	2B	100×2	1.82	292.6
SiT-XL/2+ REPA [64]	675M	250×2	1.42	305.7
LightningDiT-XL/1 [62]	675M	250×2	1.35	295.3
DDT-XL/2 [59]	675M	250×2	1.26	310.6
DiT ^{DH} -XL + RAE [69]	839M	50×2	1.13	262.6

Table 4. **System-level comparison on ImageNet 256×256 class-conditional generation.** All results are reported with CFG if applicable. **Left:** Comparison with Normalizing Flow models. **Middle:** Other 1-NFE generative models, including GANs and diffusion/flow matching-based models. **Right:** Other families of generative models. Our BiFlow model is trained for 350 epochs with perceptual distance. In all tables, $\times 2$ indicates the use of CFG incurs double NFEs. *: All AR-based methods, including AR Normalizing Flow (left) and other AR models (right), involve a large number of forward evaluations, yet each evaluation is on one or a very few tokens. For example, for standard left-to-right order AR, the *average* NFE of the entire AR process is roughly 1 (or $2 \times w$ w/ CFG), that is, K evaluations with a $\frac{1}{K}$ fraction of tokens each. In addition, TARFlow/iTARFlow has an extra NFE of 2 due to the score-based denoising post-processing. Results of [52] is collected from [70]; results of [65] is collected from [21]; TARFlow-XL/8@pix denotes TARFlow on pixel-space with patch size 8.

5.4. Comparison with Prior Works

In Tab. 4, we provide system-level comparisons with previous methods on class-conditional ImageNet 256×256 generation. We categorize prior works into three groups: Normalizing Flows (Tab. 4, left), 1-NFE generative models (Tab. 4, middle), and other families of generative models (Tab. 4, right). All our models are trained to convergence.

Comparison with Normalizing Flows. Tab. 4 (left) compares BiFlow with previous state-of-the-art Normalizing Flows models. Our BiFlow-B/2, with only 133 million parameters, achieves an FID of 2.39 in a single function evaluation (1-NFE), establishing a new state-of-the-art among Normalizing Flows. In contrast, STARFlow uses thousands of sequential decoding steps due to their autoregressive sampling process. It yields a similar FID score with about 10× parameters and more than 400× inference wall-clock time (see Tab. 6 for details).

More broadly, BiFlow represents a significant advancement in Normalizing Flows, demonstrating that direct and efficient generation can coexist with high fidelity.

Comparison with Other Generative Models. We compare BiFlow with other generative model families, especially 1-NFE methods. As shown in Tab. 4, BiFlow offers an excellent balance between generation quality and sampling efficiency. These results demonstrate that BiFlow achieves performance on par with leading 1-NFE generative models.

6. Conclusion

This work revisits one of the oldest, yet most principled, foundations of generative modeling — Normalizing Flows — and redefines its boundaries. We challenge the conventional



Figure 6. **1-NFE Generation Results.** We show selected samples generated by our BiFlow-B/2 model with guidance scale 2.0 on ImageNet 256×256. BiFlow achieves high-fidelity generation with only a single function evaluation (1-NFE) from noise.

wisdom that the reverse process must be the exact analytic inverse of the forward process, and demonstrate that the long-held constraint is unnecessary. By introducing a learnable reverse model, BiFlow pushes Normalizing Flows from analytically invertible mappings to trainable bidirectional systems, from autoregressive sampling to fully parallelized, efficient 1-NFE generation, and from an implicit generative model towards a *direct generative model*. Experiments demonstrate that BiFlow achieves competitive generation quality among Normalizing Flows, while delivering up to two orders of magnitude faster inference than its explicit inverse counterpart. We hope this work can serve as a step toward rethinking and expanding the scope of Normalizing Flows, inspiring future research on direct, flexible, and efficient NF-based generation.

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.*, $\tilde{x}, x^1, x^2, \dots, x^{B-1}$, along with the prior $z = x^B$. Similarly, the reverse model \mathcal{G} outputs the sequence of intermediate hiddens states hs as reverse trajectory: $h^{B-1}, h^{B-2}, \dots, h^0$, 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

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

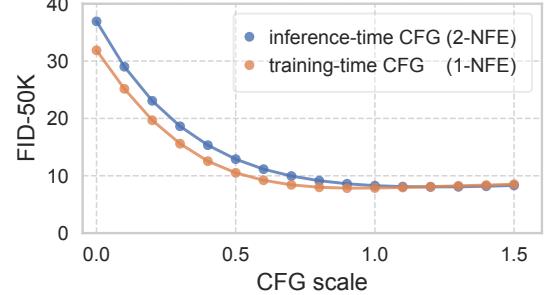


Figure 7. **Comparison between training-time and inference-time CFG of BiFlow.** Training-time CFG achieves similar or better performance compared to inference-time CFG while requiring only half inference compute and retaining full post-hoc tuning flexibility. (Settings: BiFlow-B/2, 160 epochs, MSE-only 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}_ϕ^i . Therefore, the unguided block output can be expressed as

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

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.*, $\mathcal{G}_\phi^{i,\text{cfg}}(h^i | \mathbf{c}, w_i)$. 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:

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

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.

$w_d \setminus w$	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.

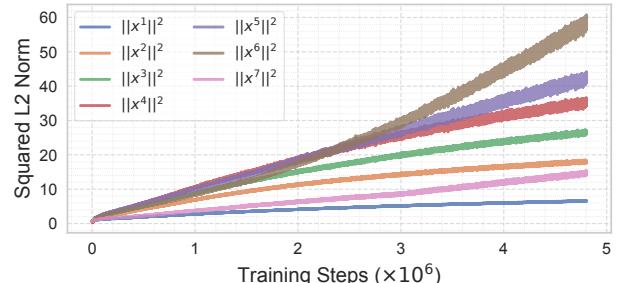
C.5. Improved TARFlow CFG

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

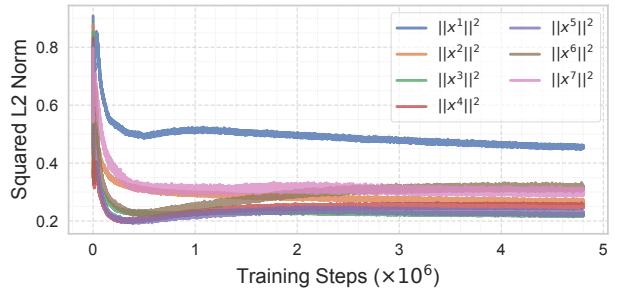
$$z_{t,\text{cfg}}^i = z_t^{i+1} \odot \exp(\alpha_{t,\text{cfg}}^i) + \mu_{t,\text{cfg}}^i,$$

where guidance is applied to the predicted parameters by

$$\begin{aligned} \alpha_{t,\text{cfg}}^i &= (1 + w_t) \alpha_t^i(\cdot | \mathbf{c}) - w_t \alpha_t^i(\cdot), \\ \mu_{t,\text{cfg}}^i &= (1 + w_t) \mu_t^i(\cdot | \mathbf{c}) - w_t \mu_t^i(\cdot), \end{aligned}$$



(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 $(\cdot | \mathbf{c})$ 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 :

$$z_{t,\text{cfg}}^i = (1 + w_t) z_t^i(\cdot | \mathbf{c}) - w_t z_t^i(\cdot).$$

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

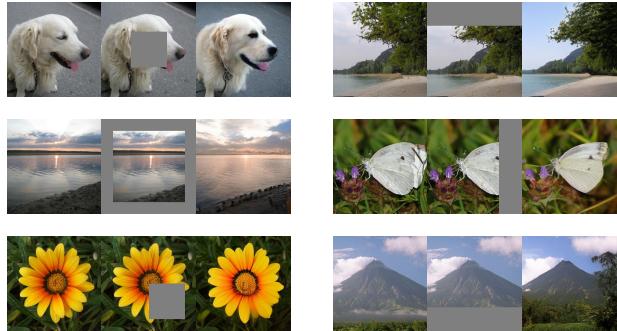


Figure 9. **Inpainting.** BiFlow enables efficient image inpainting by leveraging its bidirectional mapping between images and noise. By resampling the masked part of the noise, BiFlow can perform training-free inpainting on various image masks. Each triplet contains ground-truth image (left), masked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

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

Finally, the modified noise z' is mapped back to the image domain by the reverse model \mathcal{G}_ϕ . 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}_ϕ allows us to generate images from noise z under different class conditions. For a fixed z , changing the class label c primarily modifies the class-dependent appearance while largely preserving the global spatial structure.

Concretely, given an image x with label c , we obtain its prior variable $z = \mathcal{F}_\theta(x \mid c)$, and reconstruct it using a different label c' , writing $x' = \mathcal{G}_\phi(z \mid c')$. 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.

References

- [1] Michael S. Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. In *ICLR*, 2023.
- [2] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.
- [3] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In *ICLR*, 2018.
- [4] Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T. Freeman. Maskgit: Masked generative image transformer. In *CVPR*, 2022.
- [5] Huayu Chen, Kai Jiang, Kaiwen Zheng, Jianfei Chen, Hang Su, and Jun Zhu. Visual generation without guidance. In *ICML*, 2025.
- [6] Hansheng Chen, Kai Zhang, Hao Tan, Leonidas Guibas, Gordon Wetzstein, and Sai Bi. pi-flow: Policy-based few-step generation via imitation distillation. *arXiv preprint arXiv:2510.14974*, 2025.
- [7] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *NeurIPS*, 2018.
- [8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.
- [9] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. In *NeurIPS*, 2021.
- [10] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. In *ICLR Workshop*, 2015.
- [11] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. In *ICLR*, 2017.
- [12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.
- [13] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entzari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, Kyle Lacey, Alex Goodwin, Yannik Marek, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis. In *ICML*, 2024.
- [14] Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M Oberman. How to train your neural ode: the world of jacobian and kinetic regularization. In *ICML*, 2020.
- [15] Kevin Frans, Danijar Hafner, Sergey Levine, and Pieter Abbeel. One step diffusion via shortcut models. In *ICLR*, 2024.
- [16] Zhengyang Geng, Ashwini Pokle, Weijian Luo, Justin Lin, and J. Zico Kolter. Consistency models made easy. In *ICLR*, 2024.
- [17] Zhengyang Geng, Mingyang Deng, Xingjian Bai, J. Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling. In *NeurIPS*, 2025.
- [18] Zhengyang Geng, Yiyang Lu, Zongze Wu, Eli Shechtman, J. Zico Kolter, and Kaiming He. Improved mean flows: On the challenges of fastforward generative models. *arXiv preprint arXiv:2512.02012*, 2025.
- [19] Arnab Ghosh, Harkirat Singh Behl, Emilien Dupont, Philip H. S. Torr, and Vinay Namboodiri. Steer: Simple temporal regularization for neural ode. In *NeurIPS*, 2020.
- [20] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Fjord: Free-form continuous dynamics for scalable reversible generative models. In *ICLR*, 2018.
- [21] Jiatao Gu, Tianrong Chen, David Berthelot, Huangjie Zheng, Yuyang Wang, Ruixiang Zhang, Laurent Dinh, Miguel Angel Bautista, Josh Susskind, and Shuangfei Zhai. Starflow: Scaling latent normalizing flows for high-resolution image synthesis. In *NeurIPS*, 2025.
- [22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.
- [23] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, 2017.
- [24] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS Workshop*, 2021.
- [25] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.
- [26] Minguk Kang, Jun-Yan Zhu, Richard Zhang, Jaesik Park, Eli Shechtman, Sylvain Paris, and Taesung Park. Scaling up gans for text-to-image synthesis. In *CVPR*, 2023.
- [27] Tero Karras, Miika Aittala, Samuli Laine, Erik Hätkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. In *NeurIPS*, 2021.
- [28] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.
- [29] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *NeurIPS*, 2018.
- [30] Diederik P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *NeurIPS*, 2016.
- [31] Tuomas Kynkänniemi, Miika Aittala, Tero Karras, Samuli Laine, Timo Aila, and Jaakko Lehtinen. Applying guidance in a limited interval improves sample and distribution quality in diffusion models. In *NeurIPS*, 2024.
- [32] Kyungmin Lee, Sihyun Yu, and Jinwoo Shin. Decoupled meanflow: Turning flow models into flow maps for accelerated sampling. *arXiv preprint arXiv:2510.24474*, 2025.
- [33] Tianhong Li and Kaiming He. Back to basics: Let denoising generative models denoise. *arXiv preprint arXiv:2511.13720*, 2025.
- [34] Tianhong Li, Dina Katabi, and Kaiming He. Return of unconditional generation: A self-supervised representation generation method. In *NeurIPS*, 2024.
- [35] Tianhong Li, Yonglong Tian, He Li, Mingyang Deng, and Kaiming He. Autoregressive image generation without vector quantization. In *NeurIPS*, 2024.

- [36] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *ICLR*, 2023.
- [37] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *ICLR*, 2023.
- [38] Nanye Ma, Mark Goldstein, Michael S. Albergo, Nicholas M. Boffi, Eric Vanden-Eijnden, and Saining Xie. Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers. In *ECCV*, 2024.
- [39] Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik P. Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *CVPR*, 2023.
- [40] Maxime Oquab, Timothée Darcret, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaddin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision. *TMLR*, 2024.
- [41] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *NeurIPS*, 2017.
- [42] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *ICCV*, 2023.
- [43] Yansong Peng, Kai Zhu, Yu Liu, Pingyu Wu, Hebei Li, Xiaoya Sun, and Feng Wu. Flow-anchored consistency models. *arXiv preprint arXiv:2507.03738*, 2025.
- [44] Sucheng Ren, Qihang Yu, Ju He, Xiaohui Shen, Alan L Yuille, and Liang-Chieh Chen. Beyond next-token: Next-x prediction for autoregressive visual generation. In *ICCV*, 2025.
- [45] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *ICML*, 2015.
- [46] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.
- [47] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.
- [48] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *NeurIPS*, 2016.
- [49] Axel Sauer, Katja Schwarz, and Andreas Geiger. Stylegan-xl: Scaling stylegan to large diverse datasets. In *SIGGRAPH*, 2022.
- [50] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.
- [51] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *ICLR*, 2020.
- [52] Yang Song and Prafulla Dhariwal. Improved techniques for training consistency models. In *ICLR*, 2023.
- [53] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 2024.
- [54] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.
- [55] Zhicong Tang, Jianmin Bao, Dong Chen, and Baining Guo. Diffusion models without classifier-free guidance. *arXiv preprint arXiv:2502.12154*, 2025.
- [56] Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, and Liwei Wang. Visual autoregressive modeling: scalable image generation via next-scale prediction. In *NeurIPS*, 2024.
- [57] Jakub M. Tomczak and Max Welling. Improving variational auto-encoders using householder flow. *arXiv preprint arXiv:1611.09630*, 2016.
- [58] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.
- [59] Shuai Wang, Zhi Tian, Weilin Huang, and Limin Wang. Ddt: Decoupled diffusion transformer. *arXiv preprint arXiv:2504.05741*, 2025.
- [60] Zidong Wang, Yiyuan Zhang, Xiaoyu Yue, Xiangyu Yue, Yangguang Li, Wanli Ouyang, and Lei Bai. Transition models: Rethinking the generative learning objective. *arXiv preprint arXiv:2509.04394*, 2025.
- [61] Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, and Saining Xie. Convnext v2: Co-designing and scaling convnets with masked autoencoders. In *CVPR*, 2023.
- [62] Jingfeng Yao, Bin Yang, and Xinggang Wang. Reconstruction vs. generation: Taming optimization dilemma in latent diffusion models. In *CVPR*, 2025.
- [63] Qihang Yu, Ju He, Xueqing Deng, Xiaohui Shen, and Liang-Chieh Chen. Randomized autoregressive visual generation. In *ICCV*, 2025.
- [64] Sihyun Yu, Sangkyung Kwak, Huiwon Jang, Jongheon Jeong, Jonathan Huang, Jinwoo Shin, and Saining Xie. Representation alignment for generation: Training diffusion transformers is easier than you think. In *ICLR*, 2025.
- [65] Shuangfei Zhai, Ruixiang Zhang, Preetum Nakkiran, David Berthelot, Jiatao Gu, Huangjie Zheng, Tianrong Chen, Miguel Angel Bautista, Navdeep Jaitly, and Joshua Susskind. Normalizing flows are capable generative models. In *ICML*, 2024.
- [66] Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *NeurIPS*, 2019.
- [67] Huijie Zhang, Aliaksandr Siarohin, Willi Menapace, Michael Vasilkovsky, Sergey Tulyakov, Qing Qu, and Ivan Skorokhodov. Alphaflow: Understanding and improving mean-flow models. *arXiv preprint arXiv:2510.20771*, 2025.
- [68] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.
- [69] Boyang Zheng, Nanye Ma, Shengbang Tong, and Saining Xie. Diffusion transformers with representation autoencoders. *arXiv preprint arXiv:2510.11690*, 2025.
- [70] Linqi Zhou, Stefano Ermon, and Jiaming Song. Inductive moment matching. In *ICML*, 2025.



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