Pretrained Reversible Generation as Unsupervised Visual Representation Learning

Appendix

A. Why PRG is effective?

What a model can generate, it can understand: pixel-perfect reconstruction means the visual breadcrumbs are all there. It just needs a bit of fine-tuning to tie the generative pathway to the labels. Raw-image baselines often fall prey to ”shortcut learning”. Because neighboring feature pixels have heavily overlapping receptive fields, the network may rely on trivial cues and end up with weak representations. In contrast, PRG reuses the pretrained reverse generative pathway, recycling each ODE-solver output as the input for the next iteration. This loop drives feature extraction at a semantic level without losing low-level detail and the injected noise perturbations make the representations significantly more robust. Sec. 4.3.2 show that every point along the trajectory possesses strong representational capacity, and the OOD experiments also provide compelling evidence for this claim.

In the first pre-training stage, since no downstream task information is available, it remains unclear which features are most relevant. Consequently, no compression is performed during pre-training. Instead, the model aims to approximate the optimal representation as closely as possible in the absence of downstream information by leveraging unconditional flow matching to learn representations, effectively maximizing the lower bound of mutual information.

According to the manifold assumption [46, Chapter 20], data lies on a low-dimensional manifold $\mathcal{M}$ of intrinsic dimension d^*, which is significantly smaller than the ambient dimension D. Although, in theory, intermediate latents encode the same information as the original data, using these latents as inputs for downstream tasks is more meaningful. A well-trained flow model effectively extracts data from the low-dimensional manifold $\mathcal{M}$ and lifts it to the ambient space $\mathbb{R}^D$, ensuring that every sample in $\mathbb{R}^D$ remains semantically meaningful. In contrast, directly sampling from the original data space does not necessarily preserve semantic coherence. For instance, generating a 64×64 image by sampling from a 64×64 multivariate Gaussian distribution would typically result in a meaningless noise-like image resembling a snowflake pattern.

Moreover, since the flow trajectories of an ordinary differential equation (ODE) do not intersect [11], points within a given region remain confined, thereby preserving topological relationships throughout the transformation process.

Subsequent fine-tuning for downstream tasks is equally crucial. By adopting techniques such as optimal transport matching and gradient guidance, the model undergoes efficient adaptation, selectively discarding redundant information while selecting and enhancing task-critical features. As shown in Figure 4, mutual information decreases during fine-tuning, whereas task accuracy improves.

A.1. What’s the difference between fine-tuning and training a classifier based on the ODE architecture?

The effectiveness of fine-tuning depends significantly on whether the model undergoes pre-training beforehand. If a classifier is trained directly using the ODE architecture without pre-training, the model must learn meaningful intermediate latents solely through a supervised loss. This approach often fails, as the model struggles to discover good latent representations in the absence of prior knowledge. Our experiments (Appendix A.1.1) also confirm that this method performs poorly.

In contrast, pre-training allows the model to extract a rich set of useful features as intermediate latents. During fine-tuning, the model can then selectively retain and enhance features relevant to downstream tasks while compressing redundant information in the latent space. This process leads to a more effective and structured representation, ultimately improving downstream task performance.

A.1.1. Comparison with Generation without pre-training

We compare PRG with classifiers trained without generative pre-training. To ensure fairness, we train the latter model for 600 epochs until its performance no longer improves. As shown in Tab. 1, PRG without pre-training, relying solely on a single supervision signal, may discard useful information, leading to suboptimal performance. This highlights that the intermediate latent variables of a pretrained flow model provide a strong initialization, requiring only slight fine-tuning for optimal feature extraction.

	CIFAR-10	**Tiny-ImageNet**	**ImageNet**
PRG w/o pre-training	0.70	0.35	0.42
PRG	0.97	0.71	0.78

Table 1. **Effect of pre-training:** Performance comparison of PRG with and without pre-training on the CIFAR-10 and Tiny-ImageNet.

A.2. Model-agnosticity

Our method follows the model-agnostic principle as defined in Appendix D.5 of [57], where different architectures achieve comparable encoding quality through flow matching. This indicates that architectural constraints stem from implementation choices rather than the framework itself.

Notably, while early diffusion models primarily relied on U-Nets for practical stability, recent work [3] has demonstrated that flow matching can be successfully achieved with minimal architectural requirements, such as shallow skip connections. In this work, we achieve state-of-the-art performance across both U-Net (Sec. 4.4.1) and Transformer (Sec. 4.4.3) architectures, further validating the generality of our approach.

A.3. Infinite-Layer Expressiveness

The infinite-layer extractor boosts accuracy by extending training rather than inference steps (Fig.6: CIFAR 0.46 vs. 0.97, Tiny 0.19 vs. 0.71). However, gains plateau after a certain point, with more complex datasets requiring longer training for optimal performance. Meanwhile, the optimal flow-matching setup achieves full experimental coverage in just 5 inference steps. **Full evaluation takes 3 seconds for CIFAR and 9 s for the Tiny-ImageNet testing set.**

B. Additional Implementation Details

B.1. Training Process Details

Tabs. 2 and 3 list the hyperparameters used for both pre-training and fine-tuning across the CIFAR-10, Tiny-ImageNet, and ImageNet datasets.

Dataset	**CIFAR-10**	**Tiny-ImageNet**	**ImageNet**
GPU	8×A100	8×A100	8×A100
Optimizer	Adam	Adam	Adam
LR base	1e-4	1e-4	1e-4
Epochs	1000	1000	2000
Batch Size	256	128	128

Table 2. Experimental settings across datasets for **pre-training**.

To enhance the reproducibility of results across various multi-stage and multi-GPU experiments, we calculate the learning rate using Eq. (1).

$$\begin{aligned} \text{LR} &= \text{LR}_{\text{base}} \times \frac{\text{num processes} \times \text{Batch Size}}{512} \\ \text{Warmup LR} &= \text{Warmup LR}_{\text{base}} \times \frac{\text{num processes} \times \text{Batch Size}}{512} \\ \text{Min LR} &= \text{Min\_LR}_{\text{base}} \times \frac{\text{num processes} \times \text{Batch Size}}{512} \end{aligned} \tag{1}$$

B.2. Evaluation of Training Efficiency

Most research experiments, including the main experiments and ablation studies, are completed within several hours. To demonstrate training efficiency concretely, Tab. 4 reports the run-time per epoch for each data set used in our experiments.

Dataset	**CIFAR-10**	**Tiny-ImageNet**	**ImageNet**
GPU	4×A100	8×A100	32×A100
Optimizer	AdamW	AdamW	AdamW
Eps	1e-8	1e-8	1e-8
Betas	(0.9, 0.999)	(0.9, 0.999)	(0.9, 0.999)
LR base	1.25e-4	1.25e-4	1.25e-4
Weight Decay	0.05	0.05	0.05
Scheduler	CosineLR	CosineLR	CosineLR
Warmup LR base	1.25e-7	1.25e-7	1.25e-7
Min LR base	1.25e-6	1.25e-6	1.25e-6
Epochs	200	200	200
Warmup Epochs	5	10	10
Image Size	32	64	64
Batch Size	256	128	128
T Span	20	32	64
Solver	Euler	Euler	Euler

Table 3. Experimental settings across datasets for **fine-tuning**.

$t_{\text{span}}/t_{\text{cutoff}}$	**CIFAR-10**	**TinyImageNet**	**ImageNet***
$20/5$	49s	307s	N/A
$20/10$	103s	681s	2220s
20	187s	1437s	3480s
$32/16$	170s	1137s	N/A

Table 4. Mean time cost per epoch during the training process. * means that the model is trained on $4*8$ A100 GPUs. t_{span} represents the total sampling length, while t_{cutoff} indicates the point along the trajectory where fine-tuning begins. For example, $20/10$ means a trajectory spanning 20 steps from x_0 to x_1, with fine-tuning starting from the midpoint of the trajectory.

B.3. Evaluation of Inference Efficiency

Table 5 presents the inference efficiency of our method on the CIFAR-10 and Tiny-ImageNet test sets.

Model	**CIFAR-10 (s)**	**Tiny-ImageNet (s)**
PRG-GVP-S	4	10
PRG-ICFM-S	3	9
PRG-OTCFM-S	3	8

Table 5. Mean inference time per epoch on the CIFAR-10 and Tiny-ImageNet test datasets.

C. Ablation Studies

C.1. Loss Type

Tab. 6 shows the results of different loss types. Compared to the standard cross entropy loss, label smoothing reduces overconfident predictions, improves model calibration, and improves robustness.

Dataset	LabelSmooth Loss	Cross-Entropy Loss
CIFAR-10	97.59	96.18
TinyImageNet	71.12	70.15

Table 6. **(Loss Type)** Comparison of LabelSmooth Loss and Cross-Entropy Loss on different datasets.

C.2. ODE Solver Type

During fine-tuning, we evaluated different ODE solvers for the reverse process: Euler (first-order), Midpoint (second-order via midpoint evaluations), RK4 (fourth-order Runge-Kutta), and Dopri5 (adaptive step sizes with a fifth-order method). Tabs. 7 and 8 compares their performance on the Cifar-10, Tiny-ImageNet dataset. The results show no significant performance differences, underscoring the method's consistent effectiveness across various solvers.

solver_type	Euler	Midpoint	RK4	Dopri5
OTCFM	97.65	97.63	97.66	97.72
ICFM	97.59	97.55	97.56	97.61
GVP	97.35	97.30	97.32	97.41

Table 7. **(ODE Solver)** Performance of various solvers on Cifar-10. Different solvers don't yield obvious differences.

Solver Type	Euler	Midpoint	RK4	Dopri5
PRG-OTCFM	71.33	71.29	71.30	71.36
PRG-ICFM	71.12	71.11	71.13	71.23
PRG-GVP	70.89	70.99	70.84	70.85

Table 8. **(ODE Solver)** Performance of various solvers on Tiny-ImageNet. Different solvers don't yield obvious differences.

C.3. Details of Out-of-Distribution Experiments

There is no direct correspondence between the test images of Tiny ImageNet and Tiny ImageNet-C, and the images in Tiny ImageNet-C do not overlap with the training images of Tiny ImageNet. We report the comparative results on Tiny ImageNet-C in Tab. 9.

	Tiny-ImageNet-C		
Method	**Clean**	**Average**	**Corruption-5**
Adversarial Training			
PGD [43]	51.08	33.46 $\downarrow$ 17.62	24.00 $\downarrow$ 27.08
PLAT [31]	51.29	37.92 $\downarrow$ 13.37	29.05 $\downarrow$ 22.24
Noise Injection			
RSE [37]	53.74	27.99 $\downarrow$ 25.75	18.92 $\downarrow$ 34.82
ENResNet [65]	49.26	25.83 $\downarrow$ 23.43	19.01 $\downarrow$ 30.25
Data Augment			
AugMix [25]	52.82	37.74 $\downarrow$ 15.08	28.66 $\downarrow$ 24.16
AutoAug [15]	52.63	35.14 $\downarrow$ 17.49	25.36 $\downarrow$ 27.27
Generative Methods			
PDE+ [70]	53.72	39.41 $\downarrow$ 14.31	30.32 $\downarrow$ 23.40
PRG-ICFM-S (ours)	56.85	46.93 $\downarrow$ 9.92	33.32 $\downarrow$ 23.53

Table 9. **(OOD: extrapolated datasets)** Performance on Tiny-ImageNet-C.Averge represents the accuracy across all corruption levels, with corruption severity ranging from 1 to 5.

C.4. The Number of Timesteps

Our findings in Tab. 10 show that longer time spans generally lead to better accuracy. On CIFAR-10 with the ICFM flow model, a t-span of 10 achieves accuracy comparable to the best result at $t = 100$. In contrast, TinyImageNet requires a t-span of 15 to achieve similar performance.

T-span	2	5	10	50	100
GVP CIFAR-10	30.54	90.23	93.26	97.50	97.55
GVP Tiny ImageNet	5.06	48.95	53.24	71.05	71.18
ICFM CIFAR-10	31.18	92.35	97.02	97.60	97.61
ICFM Tiny ImageNet	6.01	60.06	65.16	71.20	71.58

Table 10. Comparison of Performance Over Varying Time Spans.

D. Reverse Generation Process

Fig. 1 illustrates the reverse generation process from x_0 to x_1 after fine-tuning. Furthermore, Figs. 2 and 3 present the reverse generation results on the TinyImageNet dataset after pre-training and fine-tuning, respectively. Finally, Fig. 4 demonstrates the reverse process before and after applying fog corruption to the images.

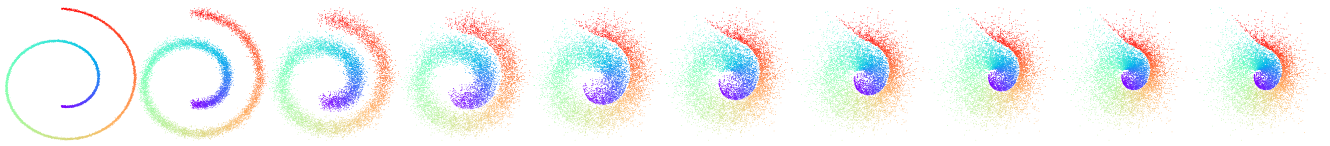
Figure 1. Reverse Generation Process on the Swiss Roll Dataset. Each color represents a different class. After diffusion, samples from the same class become more clustered, while the previously unoccupied white space, corresponding to out-of-class regions, is pushed outward.

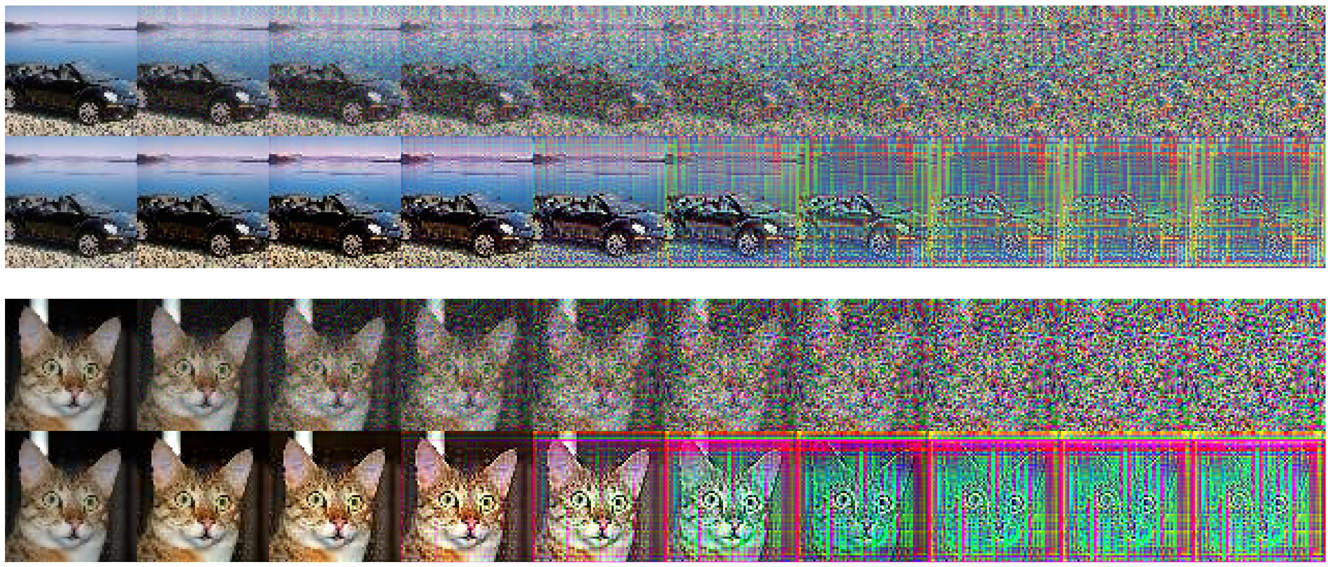
Figure 2. Reverse Generation Process on the TinyImageNet val set. The first row represents the fully pretrained reverse generative process, the second row shows the reverse generative process after extensive fine-tuning.

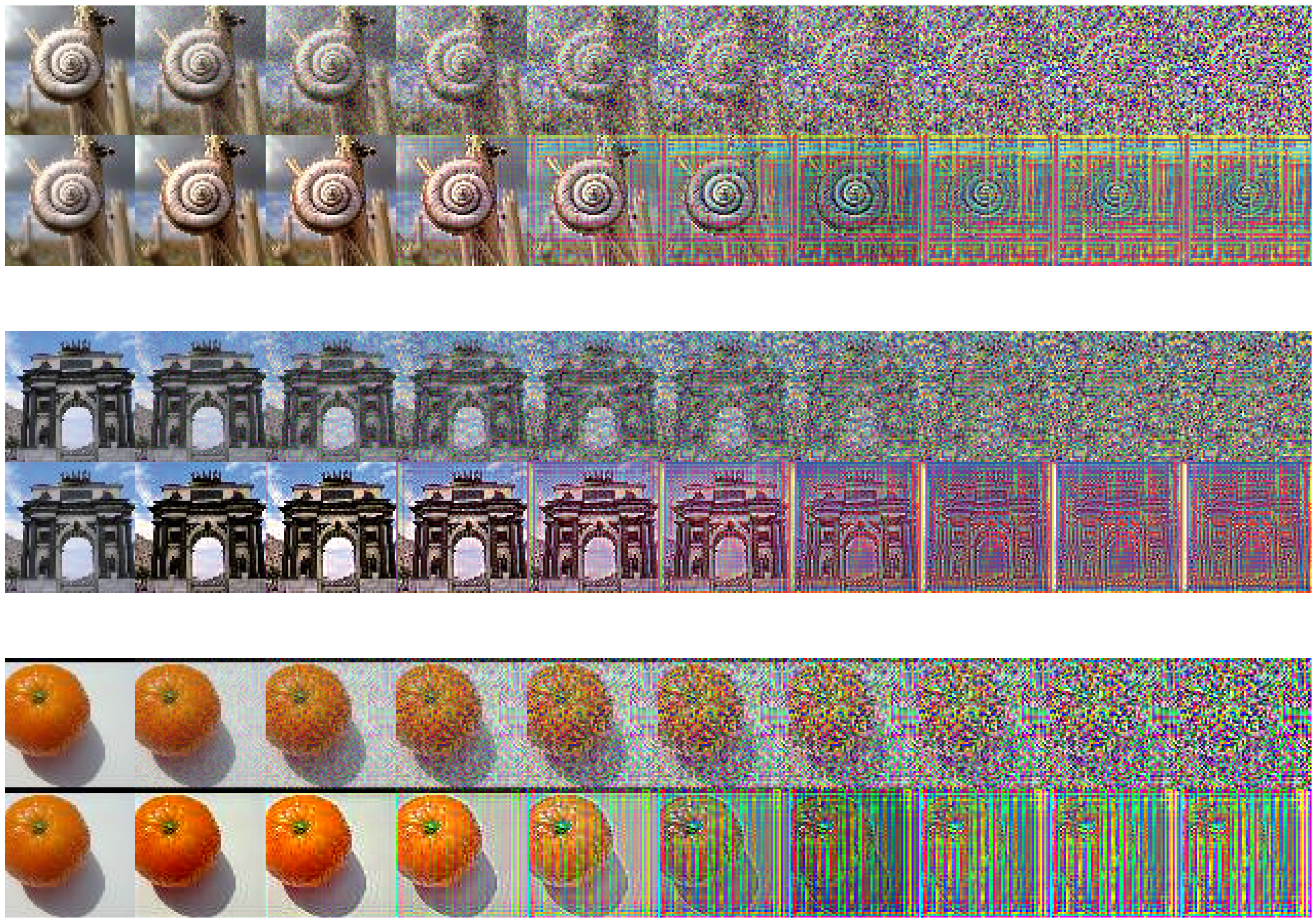

Figure 3. Reverse Generation Process on the TinyImageNet train set. The first row represents the fully pretrained reverse generative process, the second row shows the reverse generative process after extensive fine-tuning.

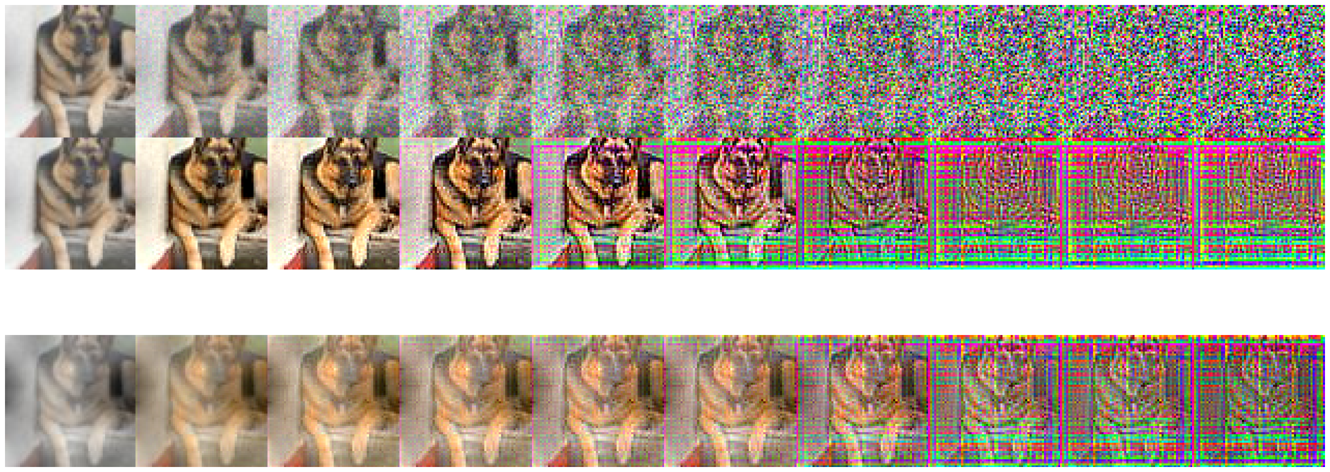

Figure 4. Reverse Generation Process on the TinyImageNet-C Dataset. The first row represents the fully pretrained reverse generative process, the second row shows the reverse process after extensive fine-tuning, and the third row illustrates the reverse generative process under fog corruption.