FlowChef: Steering of Rectified Flow Models for Controlled Generations

Supplementary Material

7. Supplementary Overview

This supplementary material contains proofs, detailed results, discussion, and qualitative results:

- Section 8: Proposition 4.1 proof.
- Section 9: Theorem 4.3 proof.
- Section 10: Numerical accuracy analysis.
- Section 11: Error dynamics for Diffusion Models.
- Section 12: Extended related works.
- Section 13: Empirical study of pixel and latent models.
- Section 14: Detailed algorithms.
- Section 15: Experimental setup details.
- Section 17: Hyperparameter study.
- Section 18: Qualitative Results.
- Section 19: Limitations & Future Work

8. Proof of the Proposition

Proposition 4.1. Let $p_1 \sim \mathcal{N}(0, \mathcal{I})$ be the noise distribution and p_0 be the data distribution. Let x_t denote an intermediate sample obtained from a predefined forward function q as $x_t = q(x_0, x_1, t)$, where $x_0 \sim p_0$ and $x_1 \sim p_1$. Define a ODE sampling process $dx(t) = f(x_t, t)dt$ and quadratic $\mathcal{L} = ||\hat{x}_0 - x_0^{ref}||_2^2$, where $f : \mathcal{R}^d \times [0, T] \to \mathcal{R}^d$ is a nonlinear function parameterized by θ. Then, under Assumption 1, the error dynamics of ODEs for controlled image generation are governed by:

$$\frac{dE(t)}{dt} = -4sE(t) + 2e(t)^T \epsilon(t),$$

where $e(t)$ is $\hat{x}_0 - x_0^{ref}$, $E(t) = e(t)^T e(t)$ is the squared error magnitude, $s > 0$ is the guidance strength, and $\epsilon(t)$ represents the accumulated errors due to non-linearity and trajectory stochasticity.

Proof. Consider the sampling process described by the ODE:

$$\frac{dx(t)}{dt} = f(x(t), t), \tag{15}$$

where $f(x(t), t)$ is a nonlinear function often parameterized via neural network θ. To guide the sampling process toward minimizing a loss function $\mathcal{L}(\hat{x}_0, x_0^{ref})$, we can adjust the dynamics by adding the gradient ∇_{x_t} to the vector field (see Eq. 2) as:

$$\frac{dx(t)}{dt} = f(x(t), t) - s \cdot \nabla_{x_t} \mathcal{L}(\hat{x}_0, x_0^{\text{ref}}), \tag{16}$$

where s is the guidance strength. Let $e(t) = \hat{x}_0 - x_0^{ref}$ be the error between the estimated and target samples. Since $\hat{x}_0(t) = x(t) + \int_t^0 f(x(\tau), \tau)d\tau$, differentiating $e(t)$ with respect to t yields:

$$\begin{aligned} \frac{de(t)}{dt} &= \frac{d\hat{x}_0(t)}{dt} && (17) \\ &= \frac{dx(t)}{dt} - f(x(t), t) && (18) \\ &= -s \cdot \nabla_{x_t} \mathcal{L}(\hat{x}_0, x_0^{\text{ref}}). && (19) \end{aligned}$$

However, this requires the compute-intensive backpropagation through ODESolver. Therefore, it is important to find an approximation of ∇_{x_t}. And the most convenient approximation is: $\nabla_{x_t} \approx \nabla_{\hat{x}_0}$. However, this derivation assumes that the integral $\int_t^0 f(x(\tau), \tau)d\tau$ is well-behaved and that $\hat{x}_0(t)$ depends smoothly on $x(t)$. In the presence of nonlinearity and stochasticity, small changes in $x(t)$ can lead to disproportionately large changes in $\hat{x}_0(t)$, due to the sensitivity of the integral to the path taken. Moreover, potential stochasticity in the trajectory mean that the mapping from $x(t)$ to $\hat{x}_0(t)$ is not injective; different trajectories $x(t)$ may lead to the same $\hat{x}_0(t)$ or vice versa. This non-unique mapping complicates the error dynamics because $\nabla_{\hat{x}_0} \mathcal{L}$ may not provide a consistent or effective direction for updating $x(t)$. Including the effects of nonlinearity and stochasticity, the error dynamics become:

$$\frac{de(t)}{dt} = -s \cdot \nabla_{\hat{x}_0} \mathcal{L}(\hat{x}_0, x_0^{\text{ref}}) + \epsilon(t), \tag{20}$$

where $\epsilon(t)$ represents the errors introduced by the nonlinearity in $f(x(t), t)$ and the sensitivity of $\hat{x}_0$ to $x(t)$ due to potential stochasticity. In other words, the approximation error $\epsilon(t)$ can be represented as:

$$\epsilon(t) = s \cdot \left(\nabla_{x_t} \mathcal{L}(\hat{x}_0, x_0^{ref}) - \nabla_{\hat{x}_0} \mathcal{L}(\hat{x}_0, x_0^{ref}) \right). \tag{21}$$

Assuming a quadratic loss function $\mathcal{L} = ||\hat{x}_0 - x_0^{ref}||_2^2$, we have $\nabla_{\hat{x}_0} \mathcal{L} = 2e(t)$, leading to:

$$\frac{de(t)}{dt} = -2se(t) + \epsilon(t). \tag{22}$$

To understand the convergence of the error, we analyze the evolution of the error magnitude $E(t) = e(t)^T e(t)$. Differentiating $E(t)$ with respect to time t, we get:

$$\frac{dE(t)}{dt} = \frac{d}{dt}\left(e(t)^\top e(t)\right) \tag{23}$$
$$= 2e(t)^\top \frac{de(t)}{dt} \tag{24}$$
$$= 2e(t)^\top (-2se(t) + \epsilon(t)) \tag{25}$$
$$= -4se(t)^\top e(t) + 2e(t)^\top \epsilon(t) \tag{26}$$
$$= -4sE(t) + 2e(t)^\top \epsilon(t). \tag{27}$$

This completes the proof. □

Notably, we derive this behavior of the ODE processes under the assumption that the error rate cannot be calculated accurately. This can either come from the incorrect estimation of $\hat{x}_0$ or the nonlinearity of ODESolver itself. In the next section, we further concretize this with respect to the RFMs.

9. Proof for Theorem

Theorem 4.3 (Update Rule for Steering the RFMs)**.** Let $u_\theta : \mathbb{R}^d \times [0,T] \to \mathbb{R}^d$ be a velocity field with constant Jacobian J_{u_θ}. Define the estimated initial state $\hat{x}_0$ from an intermediate state x_t by

$$\hat{x}_0 = x_t + t \cdot u_\theta(x_t, t).$$

Consider the quadratic loss function $\mathcal{L} = \|\hat{x}_0 - x_0^{\text{ref}}\|^2$, where x_0^{ref} is a reference sample. Then, the update rule for controlled generation is given by

$$x_{t-\Delta t} = x_t + \Delta t u_\theta(x_t, t) - s' \nabla_{\hat{x}_0}\mathcal{L},$$

where:
- $\nabla_{\hat{x}_0}\mathcal{L} = 2(\hat{x}_0 - x_0^{\text{ref}})$,
- $s' \approx (I + \Delta t \cdot J_{u_\theta})(I + t \cdot J_{u_\theta})^\top$,
- I is the identity matrix.

Proof. By lemma 4.2 and Assumption 2, we can further approximate the Eq. 8:

$$\nabla_{x_t}\mathcal{L} = (I + t \cdot J_{u_\theta})^T \nabla_{\hat{x}_0}\mathcal{L} \approx K^T \nabla_{\hat{x}_0}\mathcal{L}, \tag{28}$$

where K is the constant matrics as $\Delta t \to 0$ and $t \to 0$. Under this formulation, we can perform controlled image generation in three steps:

$$\begin{aligned} &\text{Step 1:} && \hat{x}_0 = x_t + t \cdot u_\theta(x_t, t) \\ &\text{Step 2:} && \hat{x}_t = x_t - K^T \nabla_{\hat{x}_0}\mathcal{L} \\ &\text{Step 3:} && x_{t-\Delta t} = \hat{x}_t + \Delta t \cdot u_\theta(\hat{x}_t, t). \end{aligned} \tag{29}$$

However, this will require additional forward passes. But according to Assumption 2 if Δt is sufficiently small, then by Taylor series approximation, we get:

$$x_{t-\Delta t} = x_t - K^T \nabla_{\hat{x}_0}\mathcal{L} + \Delta t \cdot u_\theta\left(x_t - K^T \nabla_{\hat{x}_0}\mathcal{L}, t\right) \tag{30}$$
$$= x_t - K^T \nabla_{\hat{x}_0}\mathcal{L} + \Delta t \left[u_\theta(x_t, t) - J_{u_\theta} \cdot K^T \cdot \nabla_{\hat{x}_0}\mathcal{L}\right] \tag{31}$$

Now, as J_{u_θ} is constant *w.r.t.* Δt. Hence, we get:

$$x_{t-\Delta t} = x_t - (I + \Delta t \cdot J_{u_\theta}) K^T \nabla_{\hat{x}_0}\mathcal{L} + \Delta t \cdot u_\theta(x_t, t) \tag{32}$$
$$= x_t + \Delta t \cdot u_\theta(x_t, t) - s' \nabla_{\hat{x}_0}\mathcal{L}, \tag{33}$$

where $s' = (I + \Delta t \cdot J_{u_\theta}) K^T$ is constant and it can predetermined.

Hence, this concludes the proof that for appropriate guidance scale s', we can perform the controlled generation as derived above. □

10. Numerical Accuracy for Model Steering

In our controlled generation framework, we aim to steer the generation process towards a reference sample x_0^{ref} by solving the modified ODE:

$$\frac{dx(t)}{dt} = f(x(t), t) = u_\theta(x(t), t) - s' \nabla_{\hat{x}_0}\mathcal{L}. \tag{34}$$

The accuracy of this numerical integration is crucial, as errors can accumulate over time, leading to deviations from the desired trajectory. The smoothness of the modified velocity field $f(x(t), t)$ significantly impacts this accuracy. Specifically, a smaller magnitude of $\left|\frac{d}{dt} f(x(t), t)\right|$ reduces local truncation errors. The following Proposition formalizes this relationship, stating that the numerical accuracy improves as $\left|\frac{d}{dt} f(x(t), t)\right|$ decreases.

Proposition 10.1. *(Informal). Given the prior notations, Assumptions, and Theorem, for any p-th order numerical method solving Eq.* (34)*, the accuracy of the numerical solution increases as the magnitude of* $\left|\frac{d}{dt} f(x(t), t)\right|$ *decreases.*

Proof. To analyze the local truncation error, consider the Taylor series expansion of the exact solution around time t when integrating backward in time from t to $t - \Delta t$:

$$x(t - \Delta t) = x(t) - \Delta t\, f(x(t), t) + \frac{(\Delta t)^2}{2} \frac{d}{dt} f(x(t), t) - \frac{(\Delta t)^3}{6} \frac{d^2}{dt^2} f(x(t), t) + O\left((\Delta t)^4\right).$$

The numerical method updates the solution using:

$$x_{t-\Delta t} = x_t + \Delta t\, \phi(x_t, t), \tag{35}$$

where $\phi(x_t, t)$ is the increment function. The local truncation error τ is the difference between the exact solution and the numerical approximation:

$$\begin{aligned}\tau &= x(t-\Delta t) - x_{t-\Delta t} \\ &= \Big[x(t) - \Delta t\, f(x(t),t) + \frac{(\Delta t)^2}{2}\frac{d}{dt} f(x(t),t) \\ &\quad - \frac{(\Delta t)^3}{6}\frac{d^2}{dt^2} f(x(t),t) + O\left((\Delta t)^4\right)\Big] \\ &\quad - \left[x_t + \Delta t\, \phi(x_t,t)\right].\end{aligned}$$

The first p-order terms cancel out, and we have:

$$||\tau|| \le \left\| \frac{(\Delta t)^{p+1}}{(p+2)!}\frac{d^{p+1}}{dt^{p+1}} f(x(t),t) \right\| \tag{36}$$

According to the Mean Value Theorem, we have

$$||\tau|| \le C(\Delta t)^{p+1} \max_{t\in[t_n,t_{n+1}]} \left\| \frac{d}{dt} f(x(t),t) \right\| \tag{37}$$

where C is a constant depending on the method. The global error $e(t) = x(t) - x_t$ accumulates these local errors over the integration interval. Under standard assumptions (e.g., Lipschitz continuity of f), the global error is bounded by:

$$\|e(t)\| \le K(\Delta t)^p \left(e^{L(T-t)} - 1\right) \max_{t\in[0,T]} \left\| \frac{d}{dt} f(x(t),t) \right\|, \tag{38}$$

where K is a constant depending on the Lipschitz constant L of f and the total integration time T. □

As the magnitude of $\left\| \frac{d}{dt} f(x_t,t) \right\|$ decreases, both the local truncation error and the global error decrease, enhancing the accuracy of the numerical solution. In the context of controlled generation, ensuring that $f(x_t,t)$ changes smoothly over time leads to more accurate integration and better alignment with the reference point x_0^{ref}. This insight and prior assumptions require that the guidance scale s' and Δt be sufficiently smaller, where higher NFEs lead to the lower Δt. Hence, we increase the NFEs significantly to stabilize the steering (see Section 17). By carefully selecting s', we ensure that the additional term $s' \cdot \nabla_{\hat{x}_0}\mathcal{L}$ does not introduce excessive variability into $f(x(t),t)$, maintaining the smoothness necessary for accurate numerical integration.

11. Error Dynamics for Diffusion Models

In this section, we derive the error dynamics for diffusion-based gradient-free guidance methods (e.g., MPGD and RB-Modulation).

Proposition 11.1. *In guided diffusion without manifold projection, the error $e_t = \tilde{x}_0^{(t)} - x_0^{ref}$ evolves as $e_{t-1} = (1-\eta)e_t + (1-\eta)\gamma_t\Delta\epsilon_t$, where $\gamma_t = \frac{\sqrt{1-\alpha_{t-1}}}{\sqrt{\alpha_{t-1}}}$ and $\Delta\epsilon_t = \epsilon_\theta(x_t,t) - \epsilon_\theta(x_{t-1},t-1)$. The error norm satisfies $||e_{t-1}|| \le (1-\eta)||e_t|| + (1-\eta)\gamma_t||\Delta\epsilon_t||$, indicating potential growth when $\gamma_t||\Delta\epsilon_t||$ is large.*

Proof. Assuming a quadratic loss $\mathcal{L} = \frac{1}{2}||\hat{x}_0^{(t)} - x_0^{ref}||_2^2$, the gradient is:

$$\nabla_{\hat{x}_0^{(t)}}\mathcal{L} = \hat{x}_0^{(t)} - x_0^{ref}, \tag{39}$$

and the guided update becomes:

$$\begin{aligned}\tilde{x}_0^{(t)} &= \hat{x}_0^{(t)} - \eta\nabla_{\hat{x}_0^{(t)}}\mathcal{L} &(40)\\ &= (1-\eta)\hat{x}_0^{(t)} + \eta x_0^{ref}, &(41)\end{aligned}$$

where $\eta > 0$ is the guidance strength. The error at timestep t is then:

$$e_t = \tilde{x}_0^{(t)} - x_0^{ref} = (1-\eta)(\hat{x}_0^{(t)} - x_0^{ref}). \tag{42}$$

Next, the DDIM update computes the subsequent noisy sample:

$$x_{t-1} = \sqrt{\alpha_{t-1}}\tilde{x}_0^{(t)} + \sqrt{1-\alpha_{t-1}}\epsilon_\theta(x_t,t), \tag{43}$$

where $\alpha_{t-1} \in (0,1)$ controls the noise schedule, and $\epsilon_\theta(x_t,t)$ is the noise prediction from the pretrained model. At timestep $t-1$, the clean sample prediction is:

$$\hat{x}_0^{(t-1)} = \frac{x_{t-1} - \sqrt{1-\alpha_{t-1}}\epsilon_\theta(x_{t-1},t-1)}{\sqrt{\alpha_{t-1}}}. \tag{44}$$

This can be further simplified to:

$$\hat{x}_0^{(t-1)} = \tilde{x}_0^{(t)} + \gamma_t\Delta\epsilon_t, \tag{45}$$

where $\gamma_t = \frac{\sqrt{1-\alpha_{t-1}}}{\sqrt{\alpha_{t-1}}}$ and $\Delta\epsilon_t = \epsilon_t heta(x_t,t) - \epsilon_\theta(x_{t-1},t-1)$. Applying the guidance step at $t-1$, we have:

$$\tilde{x}_0^{(t-1)} = (1-\eta)\hat{x}_0^{(t-1)} + \eta x_0^{ref}, \tag{46}$$

and the error becomes:

$$e_{t-1} = \tilde{x}_0^{(t-1)} - x_0^{ref} \tag{47}$$
$$= (1-\eta)\left((\tilde{x}_0^{(t)} - x_0^{ref}) + \gamma_t \Delta\epsilon_t\right) \tag{48}$$
$$= (1-\eta)e_t + (1-\eta)\gamma_t \Delta\epsilon_t. \tag{49}$$

To understand the evolution of the error magnitude, following the triangle inequality, we get:

$$||e_{t-1}|| \leq (1-\eta)||e_t|| + (1-\eta)\gamma_t||\Delta\epsilon_t||. \tag{50}$$

In idealized case (i.e., $\Delta\epsilon_t = 0$), the first term decays exponentially if $0 < \eta < 1$. However, in practice, $\Delta\epsilon_t \neq 0$ and second term introduces a perturnation that affects the convergence. In diffusion models, the noise schedule typically has α_t increasing from near 0 to 1 as t decreases from T to 1. Thus γ_t is large for large t (early steps) and decreases as t approaches 1. Meanwhile, $||\Delta\epsilon_t||$, the inconsistency in noise predictions, may be significantly early in the process due to high noise levels and diminish later as the sample refines. Hence, cumulative effect of these perturbations may prevent consistent error reduction across all steps. □

For reference, RFMs observes the straight trajectories and second term minimizes by default as shown in the prior theoretical and empirical analysis.

12. Extended Related Works

Generative Models. Recent advances in generative models, especially diffusion models like Latent Diffusion Model (LDM) [44], GLIDE [35], and DALL-E2 [43], have significantly improved photorealism compared to GAN-based methods such as StackGAN [65] and BigGAN [4]. Pretrained diffusion models have been successfully applied to diverse tasks, including image editing [17], personalization [36], and style transfer [56], but their inference flexibility remains limited, and they demand substantial resources [14, 37]. Distillation-based strategies like Latent Consistency Models [31] and Distribution Matching Distillation [63] address some limitations but lack control and broader applicability. Rectified Flow Models (RFMs) [27, 28], exemplified by Flux[1], SD3 [13], and InstaFlow [29], show promise but face challenges in downstream tasks due to inversion inaccuracies and other limitations. This work addresses these gaps, extending RFMs to downstream tasks in a training-, gradient-, and inversion-free manner.

[1]https://huggingface.co/black-forest-labs/FLUX.1-dev

Conditional Sampling. Song *et al.* introduced noise-aware classifiers for controlling sampling in diffusion models [12], but these require task-specific training. Classifier-free guidance (CFG) [18] avoids this but necessitates an additional pretraining stage. FreeDoM [64] and MPGD [15] improve sampling control but remain computationally intensive. Initial extensions of conditional sampling to flow models face similar challenges, such as compute-heavy gradient backpropagation and limited applicability to latent space models. Our method, **`FlowChef`**, eliminates these issues, seamlessly enabling gradient- and inversion-free conditional sampling in latent-space models.

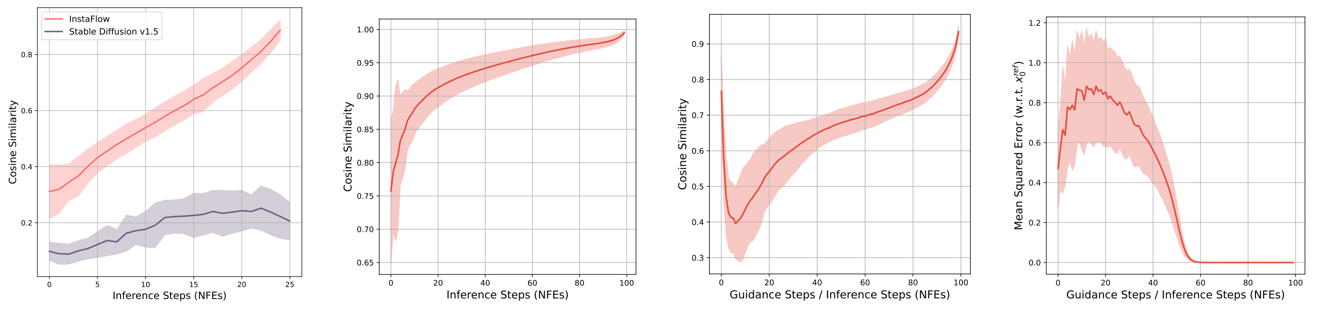
Inverse Problems. Inverse problems, dominated by diffusion-based methods [11], include pixel-space solutions such as DPS [8], Π-GDM [38], and BlindDPS [9]. PSLD [47] extends support to latent-space models, while manifold-based methods [15, 51] further enhance performance. Flow-based approaches like OT-ODE [39] and D-Flow [2] improve speed and quality but remain resource-intensive. Recent advancements like PnP-Flow [32] achieve training- and gradient-free solutions for pixel-space models but face issues like smoothness artifacts. Existing solutions are resource-intensive and unsuitable for large-scale latent models. **`FlowChef`** leverages vector field properties of RFMs to enhance performance, generalization, and scalability for state-of-the-art models like Flux.

Image Editing. Image editing typically involves guiding a model to combine a reference image with an editing instruction, often through inversion [17, 20, 21, 34]. Inversion-free methods like DiffEdit [10], InfEdit [61], and TurboEdit [59] are rare, and none apply to flow models. Most state-of-the-art methods rely on cross-attention mechanisms [3, 34], which we do not prioritize. Our approach, **`FlowChef`**, introduces the first inversion-free image editing method for RFMs, achieving competitive results with state-of-the-art methods.

13. Empirical Findings

In Section 4, we provided theoretical insights into **`FlowChef`** along with an intuitive algorithm. To complement the theory, we conducted an empirical analysis on large-scale RFMs to validate the Assumptions, Propositions, Lemmas, and Theorems presented. The results are summarized in Figure 6.

In Figure 6a, we compare the gradient cosine similarity with and without backpropagation through the ODESolver for InstaFlow and Stable Diffusion v1.5. For all denoising steps, the gradients of SDv1.5 behave nearly randomly, indicating that the stochasticity of the base model significantly impacts gradients, even when using the ODE sam-



(a) Gradient Similarity in InstaFlow *vs.* Stable Diffusion v1.5. (b) Gradient Similarity in Rectified Flow ++ model. (c) Gradient Similarity in Rectified Flow ++ during model steering. (d) Convergence in Rectified Flow ++ during model steering.

Figure 6. Empirical analysis of gradient similarity (a, b, and c) and convergence rate. (a) and (b) analyzes the gradients without model steering. (c) contains the gradient similarity during the active model steering. And (d) shows the trajectory similarity at each timestep t *w.r.t.* the inversion based trajectory.

pling process during inference. In contrast, for InstaFlow, as denoising progresses ($t \to 0$), gradient alignment improves, supporting our derivation in Lemma 4.2, which states that as $t \to 0$, we achieve $\nabla_{x_t} \approx \nabla_{\hat{x}_0}$.

Further analysis was performed on the Rectified Flow++ model, which is designed for straight trajectories with zero crossovers. As shown in Figure 6b, well-trained models exhibit high gradient similarity even at the initial stages of denoising. However, as illustrated in Figure 6c, during active steering, the gradient direction initially diverges before improving. This behavior is also reflected in the convergence plot in Figure 6d.

We hypothesize that this phenomenon arises due to the proximity to the Gaussian noise space ($p_1 \sim N(0, I)$), where model steering is more error-prone since minor adjustments can disproportionately affect future trajectories. As denoising progresses and the distribution moves further from the noise (p_1), these errors diminish, and convergence is achieved. These observations align well with our theoretical predictions, further reinforcing the validity of **FlowChef**.

Algorithm 2: FlowChef *vs.* Baseline FreeDoM.

```
1  Input: Pretrained Rectified-flow model $u_\theta$, input
     noise sample $x_T \sim N(0, I)$, target data sample
     $x_0^{ref}$, and $\mathcal{L}$ cost function.
2  for $t \in \{T...0\}$ do
3      $v \leftarrow u_\theta(x_t, t)$   $dt \leftarrow 1/T$
4      $x_t \leftarrow x_t.require\_grad\_(True)$
5      for $N$ steps do
6          $v \leftarrow u_\theta(x_t, t)$
7          $\hat{x}_0 \leftarrow x_t + t \cdot v$
8          $loss \leftarrow \mathcal{L}(\hat{x}_0, x_0^{ref})$
9          $\nabla_{x_t} \leftarrow$ grad(loss, $x_t$)   // Compute heavy BP
10         $x_t \leftarrow$ Optimize($x_t$, loss)   // Lemma 4.2
11     $v \leftarrow u_\theta(x_t, t)$
12     $x_{t-1} \leftarrow x_t + dt \cdot v$   // Theorem 4.3
13 RETURN $x_0$
```

14. Algorithms

This section provides an overview of the algorithms underpinning **FlowChef** for image editing and its comparison to baseline methods for a comprehensive understanding.

Image Editing. As described in Section 4.2, **FlowChef** can be easily extended to image editing. Revisiting the core concept, **FlowChef** modifies random trajectories to align with a target sample. Image editing involves balancing similarity with the target sample while introducing deviations to achieve desired edits.

Figure 3 and Section 17 illustrate how **FlowChef** progressively transfers characteristics from high-level structure to finer details like color composition. However, editing requirements vary by task. For example, adding an object benefits from trajectory adjustments earlier in the denoising process, while color changes require gradual learning at later stages. We can optimize parameters for diverse tasks using the generalized **FlowChef**, as detailed in Algorithm 1.

To simplify the process, we extend **FlowChef** to support off-the-shelf editing tasks, such as those in the PIE-Benchmark, as detailed in Algorithm 2. Assume a non-edit region mask, M_{edit}, derived from cross-attention or human annotation. To steer the trajectory towards the desired edits, we modify the velocity (v) using a classifier-free guidance strategy:

Algorithm 3: : **FlowChef** optimized for a wide range of image editing tasks.

```
1  Input: Pretrained Rectified-flow model $u_\theta$, input noise sample $x_T \sim N(0, I)$, target data sample $x_0^{ref}$, $c^{edit}$ is edit prompt, $c^{base}$ is base prompt, $M$ is user-provided input mask, and $\mathcal{L}$ cost function.
2  for $t \in \{T...0\}$ do
3      $dt \leftarrow 1/T$
4      $c \leftarrow [c^{edit}, c^{base}]$
5      $v \leftarrow u_\theta(x_t, t, c)$
6      $v_{edit}, v_{base} = v.chunk(2)$
7      $v = v_{edit} + \neg mask \cdot (v_{edit} - v_{base}) \cdot s$
8      $M_{edit} \leftarrow M$
9      $x_t \leftarrow x_t.require\_grad\_(true)$
10     if $t < min_T$ then
11         for $N$ steps do
12             $\hat{x}_0 \leftarrow x_t + t \cdot v$
13             if $t < max\_full\_steps_T$ then
14                 $M_{edit} \leftarrow I$
15             $loss \leftarrow \mathcal{L}(\hat{x}_0, x_0^{ref}) \cdot M_{edit}$
16             $x_t \leftarrow$ Optimize($x_t$, loss)   // Lemma 4.2
17     $x_{t-1} \leftarrow x_t + dt \cdot v$   // Theorem 4.3
18 RETURN $x_0$
```

$$v = v_{edit} + \neg mask \cdot (v_{edit} - v_{base}) \cdot s, \tag{51}$$

where v_{edit} corresponds to the edit prompt and v_{base} to the base (negative) prompt. This adjustment ensures the trajectory reflects the desired edits.

To maintain alignment of non-edited regions with the target sample, we modify the cost function as follows:

$$\mathcal{L}(\hat{x}_0, x_0^{ref}) = ||(\hat{x}_0 - x_0^{ref}) \cdot M_{edit}||_2^2. \tag{52}$$

Preserving the original image structure is crucial for edits such as color or material changes. To achieve this, we introduce the parameter $max\_full\_steps\_T$, which determines the number of steps that apply full **FlowChef** guidance with an identity mask. This ensures structural preservation while facilitating edits. Section 17 details a comprehensive reference for hyperparameters.

FlowChef vs. Baseline FreeDoM. Algorithm 2 compares **FlowChef** to the baseline FreeDoM, a diffusion model method that modifies the score function using a classifier guidance-like approach. FreeDoM requires estimating velocity and calculating gradients (∇_{x_t}) through backpropagation via the ODESolver u_θ, as marked in red. In contrast, as highlighted in green, **FlowChef** eliminates the need for backpropagation while still achieving convergence.

Hyperparameter	OT-ODE	D-Flow	PnP-Flow	FlowChef
Iterations / NFEs	200	20	50	200
Optimization per iteration	1	-	-	1
Optimization per denoising	-	50	-	-
Avg. sampling steps	-	-	5	-
Guidance scale	1	1	1	500
Cost function	L1	L1**2	L1	MSE
initial time (1 means noise)	0.8	-	-	-
blending strength	-	0.05	-	-
inversion	×	✓	×	×
learning rate	1	1	1	1

Table 5. Hyperparameters for solving inverse problems using pixel-space models.

Hyperparameter	D-Flow	RectifID	FlowChef
Iterations / NFEs	10	4	100
Optimization per iteration	-	-	1
Optimization per denoising	20	400	-
Blending strength	0.1	-	-
Guidance scale	0.5	0.5	0.5
Cost function	MSE	MSE	MSE
Learning rate	0.5	1	0.02
Optimizer	Adam	SGD	Adam
loss multiplier (latent/pixel)	0.000001	0.0001 / 100000	0.001/1000
inversion	✓	×	×

Table 6. Hyperparameters for solving inverse problems using latent-space models (InstaFlow).

This simplification makes **FlowChef** a more efficient and practical solution without sacrificing performance.

15. Experimental Setup

This section outlines the hyperparameters used for **FlowChef** and baseline methods in solving inverse problems.

Pixel-Space Models. All evaluations were conducted using the Rectified Flow++ checkpoint. Since public implementations of OT-ODE and D-Flow are unavailable, we implemented these methods manually based on the provided pseudocode and performed hyperparameter tuning to ensure optimal performance. Notably, DPS and FreeDoM hyperparameters are the same as the **FlowChef**. Table 5 provides a detailed overview of the hyperparameters used for each baseline.

Latent-Space Models. For latent-space models, we extended D-Flow to the InstaFlow pretrained model, repurposed RectifID for inverse problems, and fine-tuned the hyperparameters for optimal results. The best-performing hyperparameters for each baseline are listed in Table 6. We utilized their baseline implementations for diffusion model-based approaches such as Resample and PSLD-LDM, modifying only the number of inference steps. Specifically, we used 100 NFEs for Resample and 100/500 NFEs for PSLD.

Model	Hyperparameters	Chage Object	Add Object	Remove Object	Change Attrbiute	Chage Pose	Change Color	Change Material	Change Background	Change Style
FlowChef (InstaFlow)	Learning rate	0.5	0.5	0.5	0.5	0.5	0.5	0.5	1.0	0.5
	Max setps	50	50	50	50	20	30	50	50	30
	Optimization steps	1	1	3	2	2	2	2	4	1
	Inference steps	50	50	50	50	50	50	50	50	50
	Full source steps	30	30	0	10	10	20	20	0	30
	Edit guidance scale	2.0	2.0	2.0	4.5	8.0	8.0	4.0	3.0	6.0
FlowChef (Flux)	Learning rate	0.4	0.5	0.5	0.5	0.5	0.4	0.5	0.5	0.4
	Optimization steps	1	1	1	1	1	1	1	1	1
	Inference steps	30	30	30	30	30	30	30	30	30
	Full source steps	5	5	0	2	5	3	5	0	5
	Edit guidance scale	4.5	4.5	4.5	4.5	7.5	10.0	4.5	0.0	10.0

Table 7. Hyperparameter examples for which various editing tasks can be performed (following Algorithm 2). Notably, the **FlowChef** (Flux) variant can be further optimized for task-specific settings that will follow Algorithm 1 with a careful selection of hyperparameters.

Method	NFEs	CG Scale	FID (↓)	VRAM (↓)	Time (↓)
DDIM	50	-	5.39	3.67	14.22
MPGD	50	1	4.24	6.56	25.01
MPGD	50	10	5.46	6.56	25.01
Ours$_{\text{w/ skip grad}}$	50	1	**19.28**	6.56	24.95
RFPP (2-flow)	2	-	4.56	3.29	0.28
RFPP (2-flow)	15	-	4.29	3.36	2.75
Ours$_{\text{w/ backpropagation}}$	15	5	**2.77**	17.98	12.79
Ours$_{\text{w/ skip grad}}$	15	50	3.13	**6.64**	**5.85**

Table 8. Performance of Various guided sampling methods on ImageNet64x64 with 32 batch size inference on A6000 GPU.

Method	CLIP-I (↑)	CLIP-T (↑)	VRAM	Time
FreeDoM	**0.5343**	0.2541	17GB	80 sec
MPGD	0.5285	**0.2616**	**16GB**	20 sec
RetifID	0.4583	0.1702	18GB	30 sec
D-Flow	0.4851	0.2591	23GB	5 sec
FlowChef$_{\text{(10 NFEs)}}$	0.5044	**0.2655**		**2 sec**
FlowChef$_{\text{(30 NFEs)}}$	0.5301	0.2600	**14GB**	**7 sec**
FlowChef$_{\text{(30 NFEs × 2)}}$	**0.5531**	0.2478		12 sec

Table 9. Comparison of Various Classifier Guided Style Transfer.

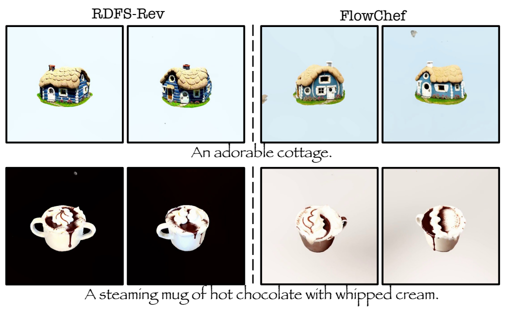


Figure 7. Extending **FlowChef** to 3D multiview synthesis.

16. Extended Results

Classifier Guidance. To validate our findings from Proposition 4.1, we conduct a toy study comparing classifier guidance on two ODE sampling methods using pretrained IDDPM and Rectified Flow++ (RF++) models on the ImageNet 64x64. As reported in Table 8, skipping the gradient in DDIM-based sampling increases the FID score, indicating significant $\epsilon(t)$. Conversely, RF++ converges well and improves the FID score. These empirical evidences further bolster our hypothesis that Rectified Flow models observe smooth vector field with the help of Proposition 4.1. Although backpropagating through the ODESolver further improves performance, it incurs higher computational costs as highlighted.

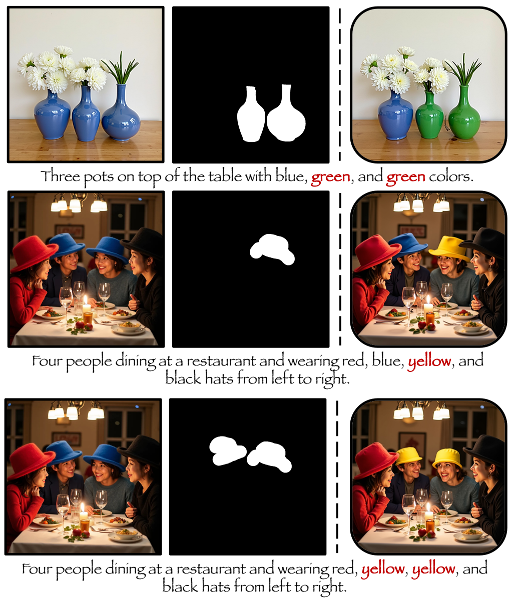

Three pots on top of the table with blue, green, and green colors.
Four people dining at a restaurant and wearing red, blue, yellow, and black hats from left to right.
Four people dining at a restaurant and wearing red, yellow, yellow, and black hats from left to right.

Figure 8. **FlowChef** (Flux) multi object editing examples.

Style Transfer. We conducted classifier-guided style transfer experiments using 100 randomly selected style reference images paired with 100 random prompts. The objective was to generate stylistic images that align visually with the reference style while adhering to the prompt. A pretrained CLIP model was used for evaluation, and we report both CLIP-T and CLIP-S scores [42]. For baseline comparisons, we included diffusion-based methods FreeDoM and MPGD and flow-based methods D-Flow and RectifID, which were extended for this task. The back-

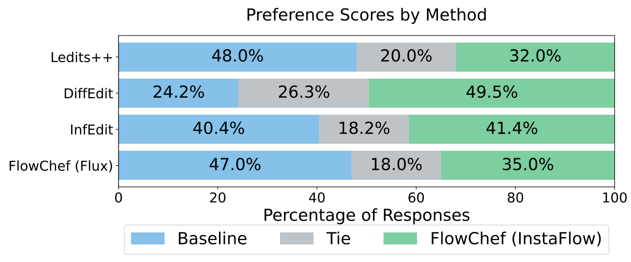


Figure 9. Human preference analysis for image editing.

bone was fixed to Stable Diffusion v1.5 (SDv1.5), with **FlowChef** evaluated in its InstaFlow variant to ensure a consistent comparison. Both quantitative and qualitative results are presented in Table 9, demonstrating the effectiveness of **FlowChef** in this setup.

Multiobject editing & 3D generations. To highlight the versatility and effectiveness of **FlowChef**, we extended our method to tackle multi-object image editing and 3D multiview generation. Figure 8 demonstrates **FlowChef** (Flux) performing complex multi-object edits, such as simultaneously modifying two pots and hats. Notably, this capability relies on the base model's ability to understand textual instructions effectively. **FlowChef** leverages this strength of Flux, achieving edits without requiring inversion, a significant advantage over traditional methods. In Figure 7, we explore **FlowChef**'s multiview synthesis capability, inspired by Score Distillation Sampling (SDS) [41]. By incorporating the core idea of **FlowChef** for model steering into recent work on RFDS [62], we evaluate its effectiveness for 3D view generation. While **FlowChef** does not improve inference efficiency or reduce cost compared to RFDS-Rev [62], it demonstrates competitive performance in generating high-quality multiview outputs. These results underline the adaptability of **FlowChef**, showcasing its potential for advanced generative tasks such as multi-object editing and 3D synthesis, while maintaining the state-of-the-art quality expected from RFMs.

Human Evaluations (Image Editing). A human preference evaluation on randomly selected 100 PIE-Bench edits (see Figure 9) shows **FlowChef** (InstaFlow) outperforming DiffEdit and competing with InfEdit. Although Ledits++ scored highest, it requires inversion, resulting in higher VRAM and time requirements. Importantly, FlowChef on Flux achieves performance comparable to Ledits++ without inversion. Comparisons with RF-Inversion show that FlowChef reduces time by almost 50% without needing inversion and achieves competitive performance.

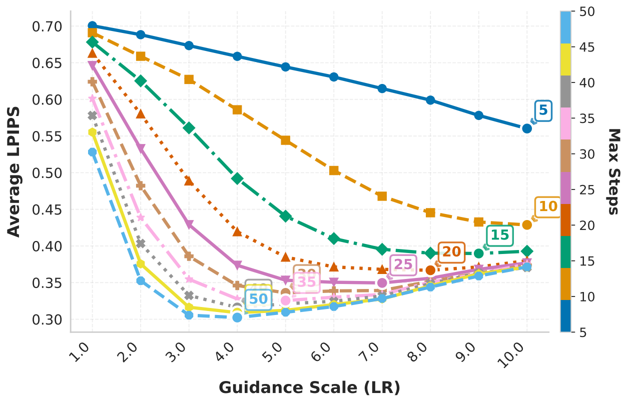


Figure 10. Hyperparameter sensitivity analysis.

17. Hyper-parameter Study

Figures 11, 12, and 13 present an analysis of the impact of various hyperparameters on steering the InstaFlow model using **FlowChef**. Figure 11 demonstrates that a lower learning rate combined with a single optimization step is insufficient to effectively steer the model. Optimal performance is achieved with a learning rate of 0.1. Additionally, Figure 12 shows that lower learning rates necessitate more optimization steps to achieve convergence. Finally, Figure 13 illustrates how the denoising trajectory can be controlled by adjusting the learning rate and optimization steps, enabling recovery of the target sample with the desired accuracy. This control is particularly critical for image editing tasks, where striking the right balance between preserving the reference sample and applying the editing prompt is essential. Table 7 further highlights optimal hyperparameter settings for image editing tasks, providing valuable guidance for achieving high-quality edits. This study underscores the flexibility of **FlowChef** in adapting to diverse use cases by tuning these parameters effectively.

Additionally, we plot the hyperparameter sensitivity for **FlowChef** (InstaFlow)'s convergence rate w.r.t. guidance scale and steps on 100 AFHQ-Cat images on latent inverse problem (see Figure 10). We can observe that lower steps require higher guidance scale and more steps require lower guidance scale. Importantly, **FlowChef** observes a smooth convergence that clearly indicates the robustness.

18. Qualitative Results

Figure 15 showcases additional qualitative examples of image editing tasks. For tasks such as changing materials or removing objects, **FlowChef** outperforms the baselines significantly. However, some limitations are noted: while **FlowChef** (InstaFlow) struggles to replace a cat with a tiger, InfEdit handles this task effectively, and Ledits++ exhibits difficulties. On the other hand, **FlowChef** (Flux)

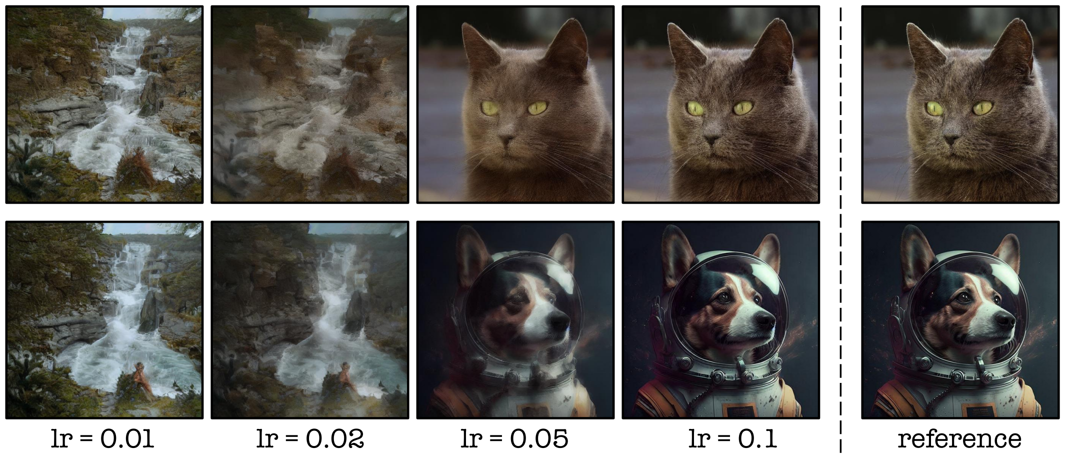


Figure 11. Effect of **`FlowChef`** learning rate with fixed 20 max steps and one optimization step on InstaFlow.

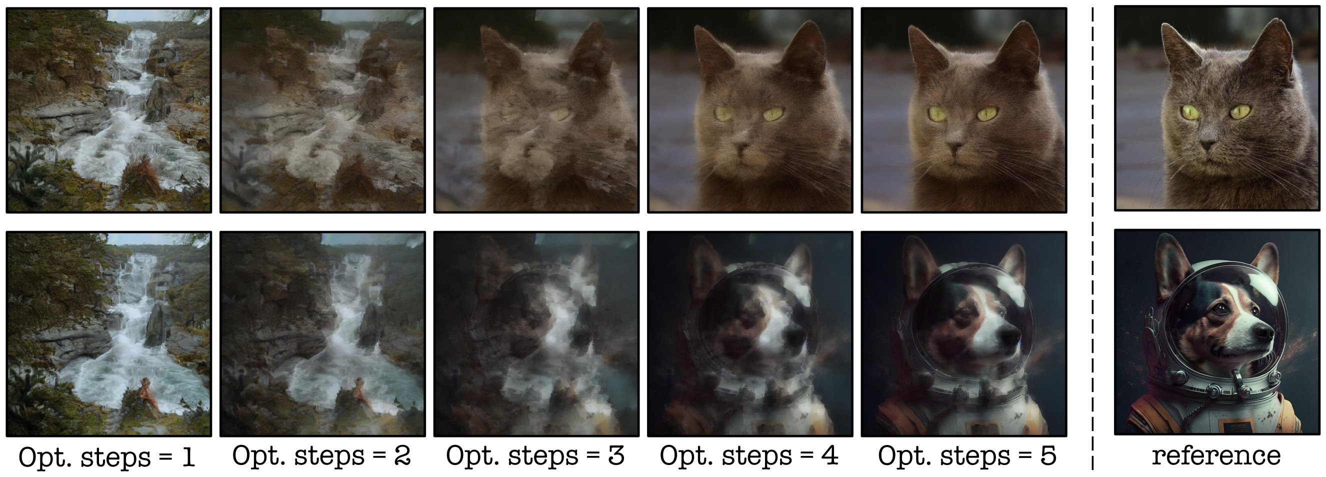


Figure 12. Effect of **`FlowChef`** optimization steps with fixed 20 max steps and 0.02 learning rate on InstaFlow.

achieves superior results, though it replaces a dog with a tiger instead of a lion in one instance. In the final example, both Ledits++ and **`FlowChef`** successfully edit long hair into short hair. Importantly, the results in Figure 15 are presented without cherry-picking, using consistent hyperparameters for both baselines and **`FlowChef`**. Variability in outcomes may still arise due to random seeds and fine-tuned hyperparameter selection.

Figures 16, 17, 18, 19, 20, and 21 provide pixel-level qualitative results for various inverse problems, spanning inpainting, deblurring, and super-resolution tasks under both easy and hard scenarios. Readers are encouraged to zoom in to inspect these comparisons more closely. For each task, we randomly selected 10 CelebA examples and evaluated various baselines. Across all difficulty levels, FreeDoM, DPS, and PnPFlow demonstrate better performance than D-Flow and OT-ODE. However, **`FlowChef` consistently outperforms all baselines**, producing sharp and visually appealing results where other methods either fail outright or introduce excessive smoothness. Hard scenarios pose challenges for all methods, but **`FlowChef`** notably improves performance even under these conditions. While **`FlowChef`** shows promise, future work is needed to address potential adversarial effects and further enhance robustness.

19. Limitations & Future Work

Limitations. While **`FlowChef`** represents a significant leap in steering RFMs for controlled generation, it shares some limitations with its baseline counterparts. Hyperparameter tuning remains a challenge, particularly due to differences in trajectory behavior. For instance, while InstaFlow trajectories are relatively linear, Flux.1[Dev] trajectories exhibit non-linearity, necessitating careful tuning.

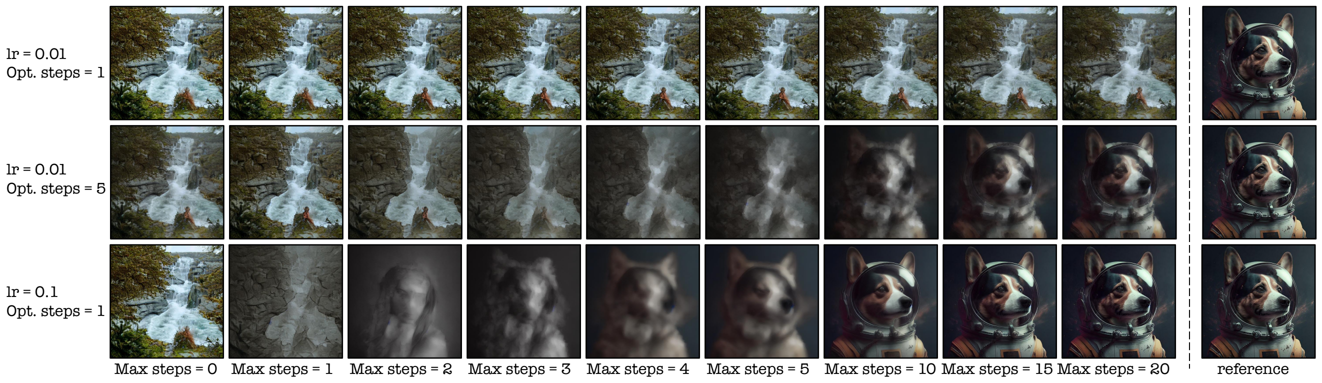
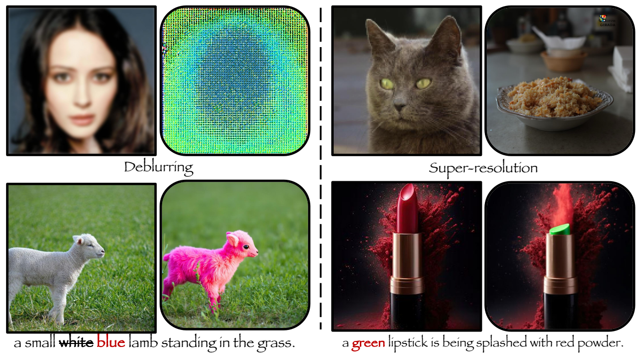


Figure 13. Effect of various **FlowChef**'s steering parameters with increasing maximum optimization steps on InstaFlow.



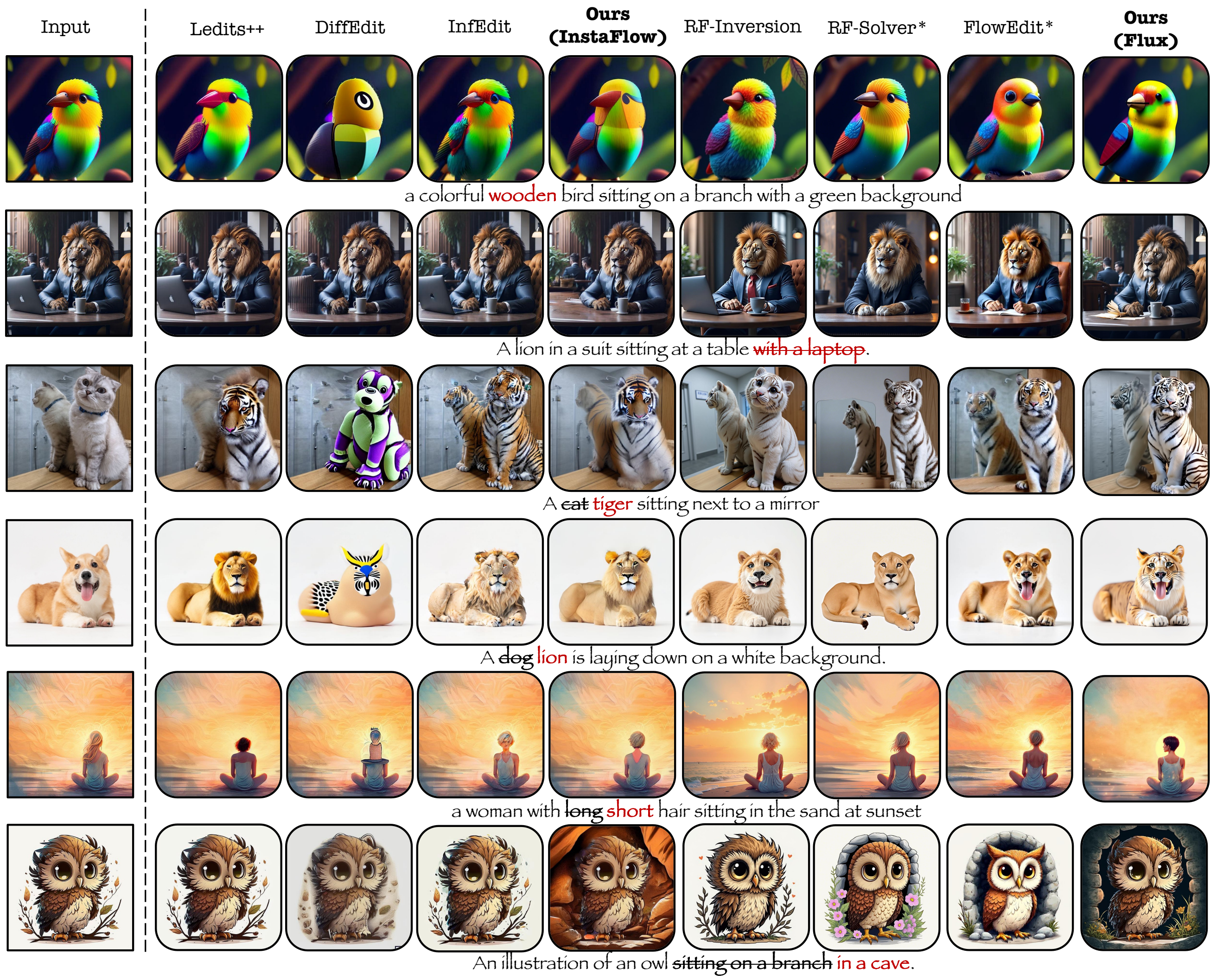
Deblurring
Super-resolution
a small ~~white~~ blue lamb standing in the grass.
a green lipstick is being splashed with red powder.

Figure 14. **FlowChef** (Flux) model failures on inverse problems and image editing.

As shown in Figure 8, **FlowChef** (Flux) faces difficulties in deblurring and super-resolution tasks, which we attribute to the pixel-space loss and non-linear behavior of the VAE model. Importantly, these limitations occur about 20%-25% of cases and can often be resolved by simply adjusting the random seed. Furthermore, due to Flux's lack of true classifier-free guidance (CFG), Algorithm 3 occasionally fails to perfectly execute color changes, sometimes producing the unaltered target image without reflecting the edit (see Figure 8). Despite these minor limitations, **FlowChef** still delivers state-of-the-art performance, making these challenges opportunities for further refinement rather than fundamental drawbacks.

Future Work. **FlowChef** opens a promising avenue for steering RFMs effortlessly with guaranteed convergence for controlled image generation. While this work extensively evaluates **FlowChef** on image generative models, future research should focus on expanding its capabilities to video and 3D generative models, areas that remain largely unexplored. Additionally, the current implementation assumes the availability of human-annotated masks for image editing. Automating this step with advanced attention mechanisms could make **FlowChef** a fully automated image editing framework. We encourage the research community to build upon this foundation to enhance its accessibility and functionality.

Ethical Concerns. As with all generative models, ethical concerns such as safety, misuse, and copyright issues apply to **FlowChef** [23, 24]. By enabling controlled generation with state-of-the-art RFMs, **FlowChef** can be leveraged for beneficial and harmful purposes. To mitigate these risks, future efforts should focus on solutions such as image watermarking, content moderation, and unlearning harmful behaviors. While these issues are not unique to **FlowChef**, addressing them will be key to ensuring its responsible use.



Input
Ledits++
DiffEdit
InfEdit
Ours (InstaFlow)
RF-Inversion
RF-Solver*
FlowEdit*
Ours (Flux)
a colorful wooden bird sitting on a branch with a green background
A lion in a suit sitting at a table ~~with a laptop~~.
A ~~cat~~ tiger sitting next to a mirror
A ~~dog~~ lion is laying down on a white background.
a woman with ~~long~~ short hair sitting in the sand at sunset
An illustration of an owl ~~sitting on a branch~~ in a cave.

Figure 15. **Qualitative results on image editing.** Additional qualitative comparisons of **`FlowChef`** with the baselines.

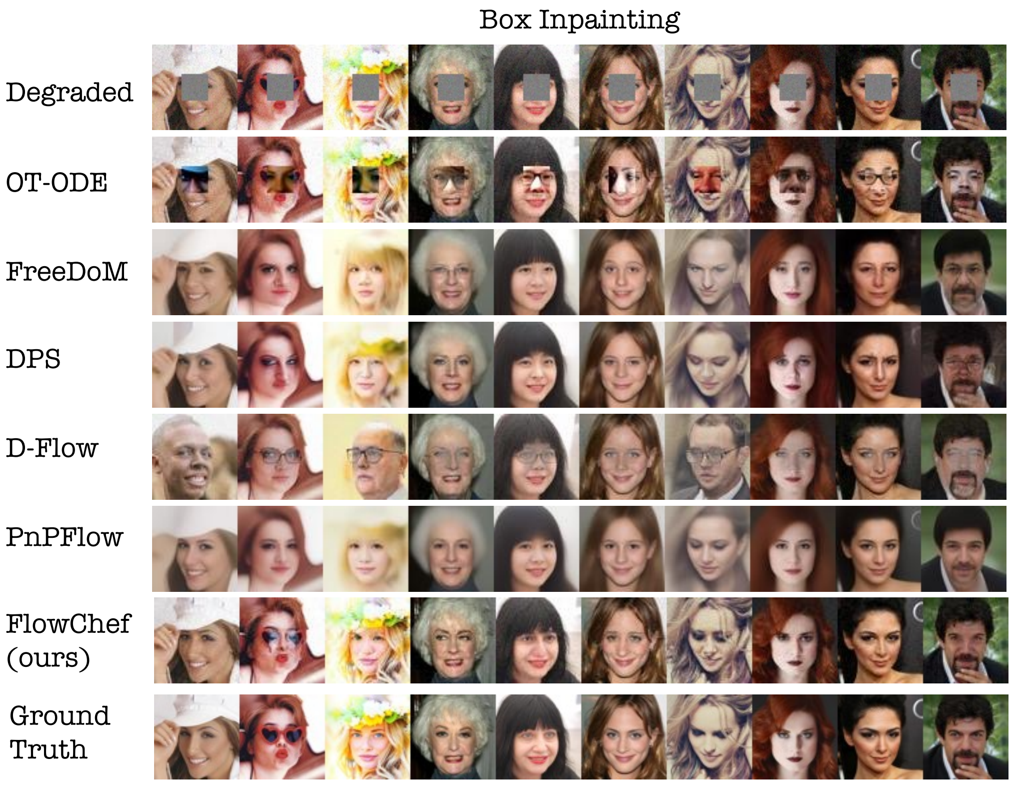


Figure 16. Qualitative examples of various methods for easy box inpainting task on RF++.

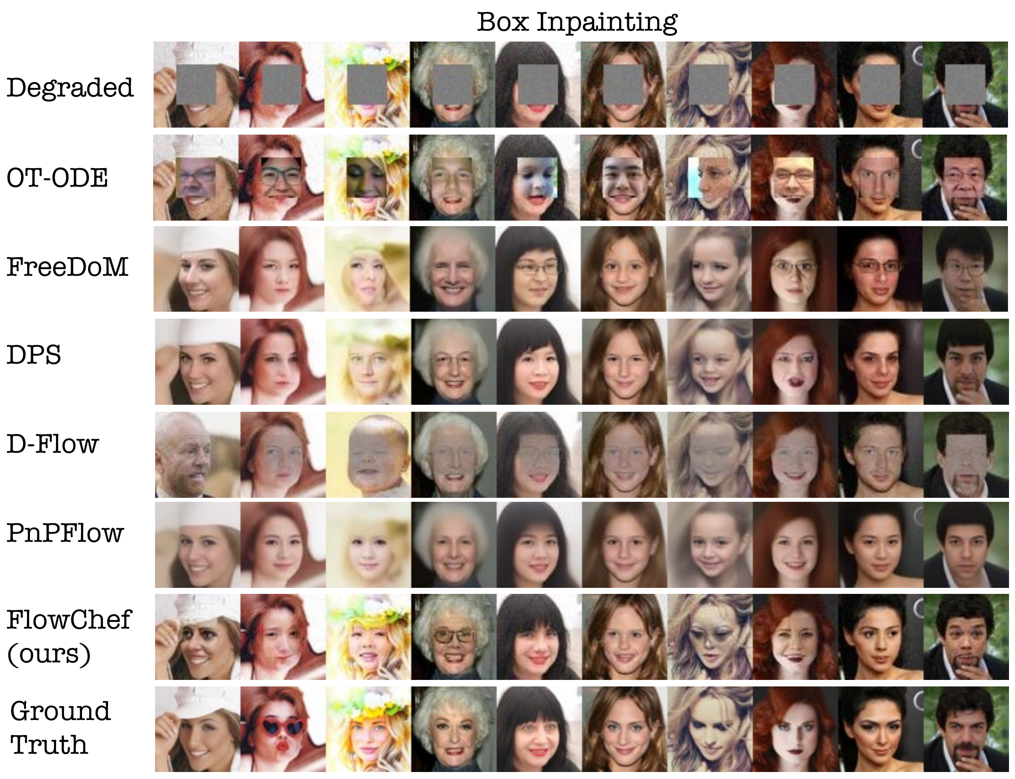


Figure 17. Qualitative examples of various methods for hard box inpainting task on RF++.

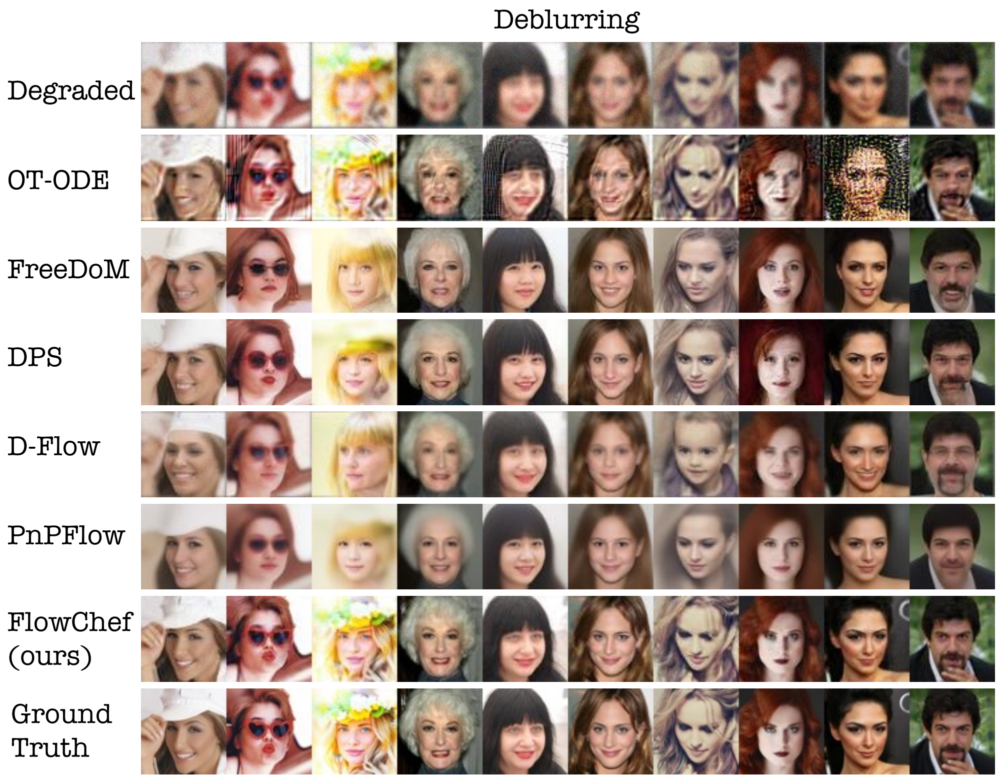


Figure 18. Qualitative examples of various methods for an easy deblurring task on RF++.

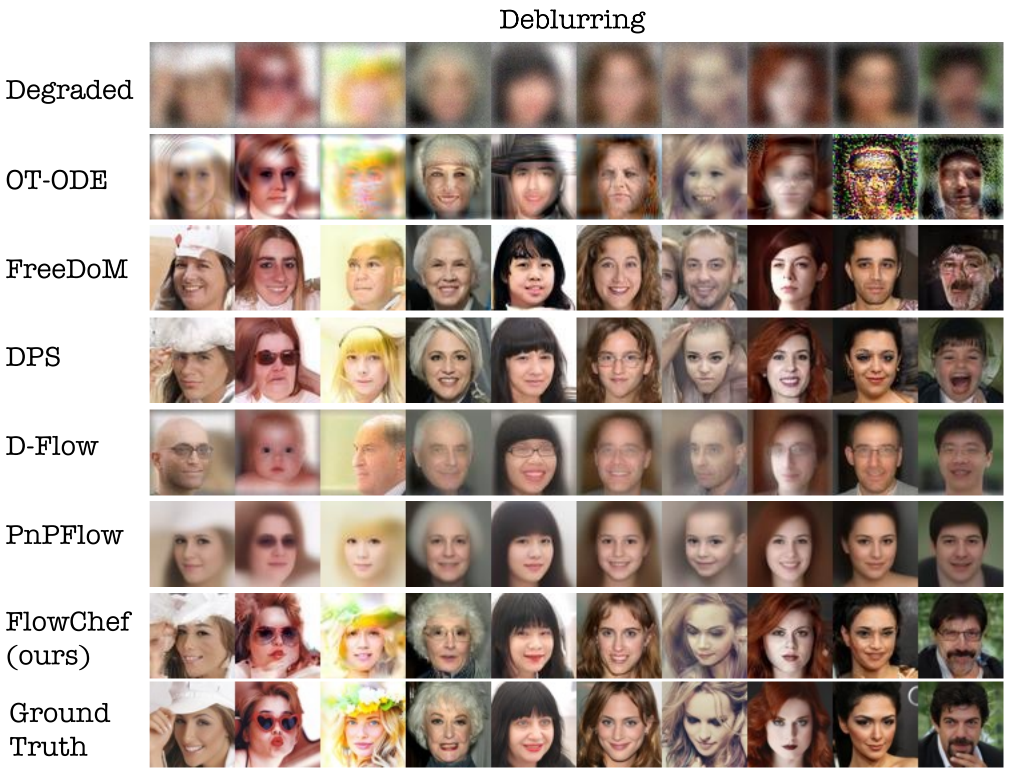


Figure 19. Qualitative examples of various methods for the hard deblurring task on RF++.

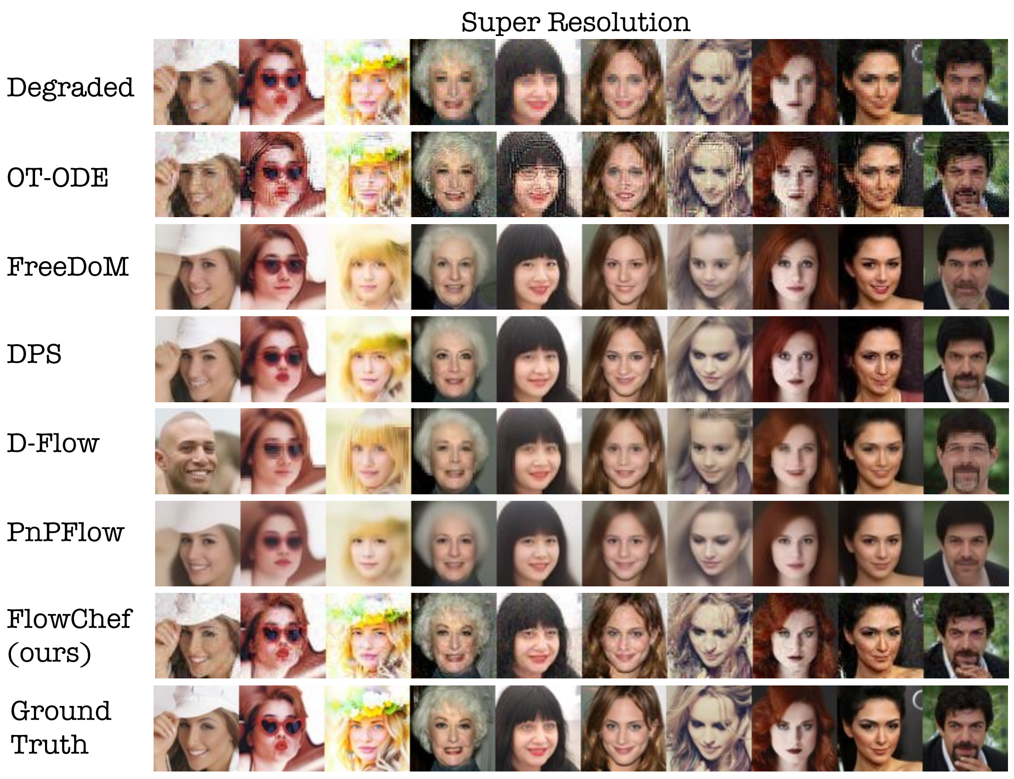


Figure 20. Qualitative examples of various methods for an easy super-resolution task on RF++.

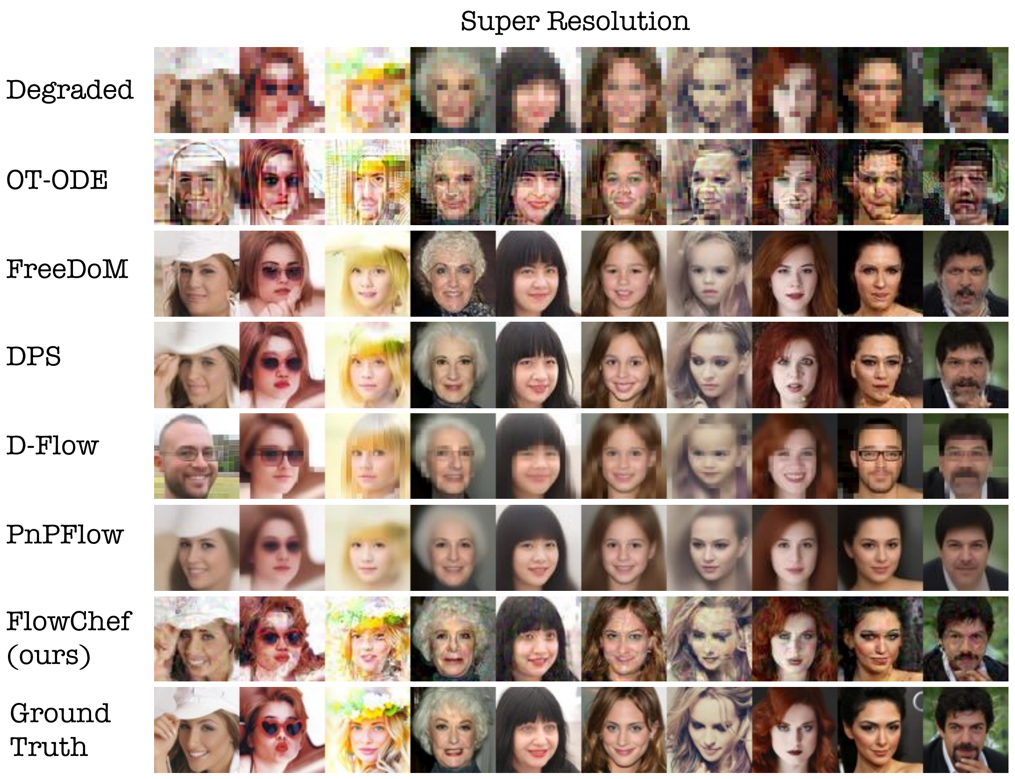


Figure 21. Qualitative examples of various methods for the hard super-resolution task on RF++.