Finite Difference Flow Optimization for RL Post-Training of Text-to-Image Models

David McAllister[1⋆], Miika Aittala[2], Tero Karras[2], Janne Hellsten[2], Angjoo Kanazawa[1], Timo Aila[2], and Samuli Laine[2]

[1] UC Berkeley [2] NVIDIA

Abstract. Reinforcement learning (RL) has become a standard technique for post-training diffusion-based image synthesis models, as it enables learning from reward signals to explicitly improve desirable aspects such as image quality and prompt alignment. In this paper, we propose an online RL variant that reduces the variance in the model updates by sampling paired trajectories and pulling the flow velocity in the direction of the more favorable image. Unlike existing methods that treat each sampling step as a separate policy action, we consider the entire sampling process as a single action. We experiment with both high-quality vision language models and off-the-shelf quality metrics for rewards, and evaluate the outputs using a broad set of metrics. Our method converges faster and yields higher output quality and prompt alignment than previous approaches.

1 Introduction

Generation of synthetic images based on text prompts has become a ubiquitous task for deep learning models. The dominant paradigm today is using diffusion models [18,50,51,53,54] that generate images by reversing a stochastic corruption process. These models learn a prompt-conditional probability flow that maps a random sample of noise into an image on the data manifold. Flow matching [2,10,17,30,33] is a popular parameterization for this flow [1].

As with large language models, the training of image synthesis models is commonly divided into pre-training and post-training stages that have different goals [41]. Pre-training is performed on large-scale, weakly curated data. The goal is to inject as much real-world knowledge into the model as possible with relatively little concern for image or prompt quality. In the typically much shorter post-training phase, the model's sample distribution is then concentrated on regions representing desirable outputs based on an external reward metric or a carefully curated fine-tuning set.

The post-training of image generators typically has a wide variety of vague goals, such as making the images more beautiful, have nicer composition, more expressive lighting, etc. This "true reward" cannot be expressed as a loss function, and thus cannot be directly optimized. What can be done instead is to define a

⋆ Work done during an internship at NVIDIA.

set of quantifiable *proxy rewards* that hopefully address many of the desirable aspects. Reinforcement learning (RL) is a general method that can, in principle, optimize any such proxy reward. Common proxy rewards include learned models trained to mimic human preferences [25, 59] along with more targeted goals, such as the clarity of text rendering.

The pre-training objective of diffusion models is minimized by a unique flow field determined by the training dataset. As such, the training constantly supervises that the generated distribution stays in alignment. Unfortunately, this is no longer true during post-training, and aspects that are irrelevant to the specified reward are left to drift freely; for example, a reward targeting accurate text rendering can disregard or even harm the overall image quality. While this "reward hacking" [3] is in the nature of RL — the policy is only designed to optimize the provided reward — the RL algorithm should not exacerbate the problem by, e.g., actively drifting in random directions along the underspecified dimensions. Such bad behavior can be partially mitigated by, e.g., specifying multiple rewards, by limiting the amount of allowed change via KL regularization, or by mixing in some pre-training objective.

Existing RL post-training methods for diffusion models typically recast stochastic sampling as a Markov decision process (MDP), i.e., a sequence of policy-guided actions with Gaussian transitions [5, 15]. DDPO [5] adopts the PPO [49] algorithm for reward optimization under the MDP formulation. Flow-GRPO [32] and DanceGRPO [64] apply the same idea in the context of flow matching. In this approach, the proposed updates to the flow are random perturbations that are independent between the sampling steps. These perturbations are reinforced if their overall effect on the sampling trajectory lands on a higher-than-average reward.

The noise in these updates is a serious weakness from the viewpoint just laid out. While the aggregate update will improve the reward, only a small fraction of its magnitude contributes to this, while the rest is reward-neutral noise that pushes the flow around in random directions. This poses several problems. The speed of progress per update is significantly limited, as a large part of any individual update does not contribute to the goal. The noise also causes the unrelated dimensions to drift freely, e.g., cycling through random image styles if they are not constrained by any reward. Finally, the drift also slowly accumulates into detrimental side effects, essentially setting a cap on how much fine-tuning can be done in total. As an example of this, we show that Flow-GRPO starts to introduce artifacts into the generated images upon extended training.

The goal of our approach is to improve the signal-to-noise ratio of the flow updates. Our method uses similar random perturbations to discover high-reward images, but decouples the resulting update from this random walk. Specifically, our method generates two nearby images and uses their difference as an approximate gradient. This image difference is weighted by the respective reward difference, so that it is guaranteed to point from the lower-reward image to the higher-reward one. We then *uniformly* update the flow directions along the generating trajectory towards this direction, relying on the de-facto "non-rotational" behavior specific

to diffusion flows [23]. Each sampling step thus receives an update that directly benefits the reward. This is in contrast to the MDP formulation where, roughly speaking, close to half of the individual flow updates may be detrimental to the reward.

Compared to Flow-GRPO, our method converges significantly faster, to higher rewards, and with fewer reward hacking artifacts. It can be used as a drop-in replacement for SOTA RL algorithms in post-training of diffusion models. The simplicity and data efficiency of our approach also enables training on *on-policy* human preference feedback. These results position our formulation as a promising alternative for RL post-training.

Our implementation and trained models are available at `https://github.com/NVlabs/finite-difference-flow-optimization`

2 Previous Work

Many methods have been proposed for post-training neural networks, and we will briefly review the most closely related ones here. For a broader context, Liu et al. [31] present a survey on image generator post-training, constructing a taxonomy for a wide range of literature on related algorithms and evaluation. Also, Uehara et al. [55] present a more RL-focused survey that examines the connections between different post-training methods in detail.

Supervised fine-tuning A common approach is to simply continue training using a smaller fine-tuning dataset. Direct preference optimization (DPO) [46], on the other hand, relies on annotated human preference pairs. It is simple to implement, building on supervised training without needing an online reward model. However, it is limited to cases where annotated preference data is available. Diffusion DPO [56] applies the method to image generators.

Differentiable rewards If the reward function is differentiable with respect to the generated image, it is possible to backpropagate the reward gradients to the generator. Commonly used approaches in this category include ReFL [62], DRaFT [9], and Deep Reward Supervision (DRS) [60]. These methods differ mainly in where exactly the supervision happens inside the generator — ReFL supervises on a random step, DRaFT on a sequence of last steps, and DRS on a sparse subset of steps. Dual-process image generation [36] propagates gradients through a VLM reward to adjust image generator weights at inference time.

Domingo-Enrich et al. [11] cast reward fine-tuning as a stochastic optimal control (SOC) problem, which is strongly connected to RL [28]. They formulate a noise schedule that is free of value function bias that typical post-training objectives suffer from. They also present adjoint matching, a general method for solving SOC problems that removes high-variance importance weights from the regression objective.

Non-differentiable rewards In many practically relevant scenarios, the reward function is not differentiable, most notably when humans provide direct reward signals. Although this is an obvious application of RL, a number of RL-adjacent

supervised learning approaches have been proposed as well. Off-policy sample-evaluate-update loops are used in language modeling [40, 65] and control settings [43,47]. In image generation, several authors apply these techniques through reward-weighted regression [5, 14, 27] as well as rejection sampling [12].

True RL methods, applicable to any reward, include the aforementioned MDP-based methods DDPO [5], Flow-GRPO [32] and DanceGRPO [64], with the last two demonstrating current state-of-the-art results. Other works perform on-policy RL with the standard denoising objective [37, 63], though they rely on strong regularization for stability in image generation settings. Some works also apply variants of value-based RL [58] to diffusion models [21, 42, 44].

GFlowNets [4, 34, 66] also rely on the MDP formulation but propose a novel training objective that is closely related to path consistency learning [39, 55]. The aim is to tilt the distribution proportionally to the reward, the same goal as KL-regularized RL [28]. However, performance vs. mainstream SOTA (Flow-GRPO, DanceGRPO) is unknown.

Inference-time optimization Orthogonal to model post-training, inference-time tweaks to the sampling process can provide improvements to output quality at the cost of reducing diversity. These techniques include, e.g., classifier-free guidance (CFG) [19], gradient-guided diffusion [16], and loss-guided diffusion [52].

3 Background

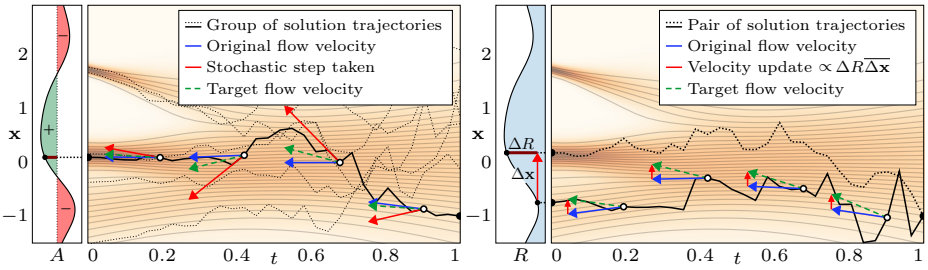
Flow matching [2,10,17,30,33] is a parameterization of a probability flow diffusion process [18, 50, 51, 53, 54]. It is popular for its intuitive formulation and ease of implementation, as well as good practical performance stemming from favorable built-in design choices.

Flow matching works by drawing an initial random noise image $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_N)$ of N pixels, and numerically solving an ordinary differential equation (ODE)

$$\mathrm{d}\mathbf{x}(t) = v_\theta(\mathbf{x}; t, \mathbf{c})\mathrm{d}t \tag{1}$$

backward in time from $t = 1$ to $t = 0$ with initial condition $\mathbf{x}(1) = \boldsymbol{\epsilon}$. Here, $v_\theta(\cdot)$ is the *velocity function*, implemented as a neural network with weights θ, that points towards the average of noise-free images consistent with the noisy image $\mathbf{x}(t)$ and time parameter t, conditioned on prompt embedding $\mathbf{c}$ for conditional generation. Following the ODE pushes the image towards the evolving denoising estimate under the training image dataset, gradually reducing the noise level and revealing a clean image $\mathbf{x}(0)$ from this distribution.

In practice, the ODE is solved by evaluating $\mathbf{x}(t)$ at discrete time steps $t_0 = 1, t_1, \ldots, t_{T-1}, t_T = 0$, where each step yields an intermediate image $\mathbf{x}_i := \mathbf{x}(t_i)$ that is slightly less noisy than the previous one. Using the simple Euler solver scheme, the step from time t_i to t_{i+1} is given by $\mathbf{x}_{i+1} = \mathbf{x}_i + (t_{i+1} - t_i)v_\theta(\mathbf{x}_i; t_i, \mathbf{c})$, so that after T steps the initial noise image at t_0 is transformed into the generated image at t_T. We refer to a sequence of images $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_T$ as a sampling trajectory.



(a) Flow-GRPO [32] (b) Ours

Fig. 1: Illustration of differences between denoising MDPs, including Flow-GRPO [32], and our method. **(a)** For each prompt, Flow-GRPO samples a group of trajectories using stochastic Euler–Maruyama sampling. The relative advantage A of each sample is the reward normalized with the group statistics. For each time step, the flow velocity is optimized towards the stochastic perturbation the trajectory took, or away from it if the sample's advantage is negative. **(b)** In our method, we sample a pair of trajectories and compute the difference $\Delta\mathbf{x}$ between the output images. The flow velocity is optimized towards normalized $\overline{\Delta\mathbf{x}}$ scaled by the reward delta ΔR. Note that both Flow-GRPO and our method apply updates to all sampled trajectories, whereas only one is highlighted here for clarity.

3.1 Reward Maximization

Given a reward function $R(\mathbf{x})$ that maps images to scalar reward values, our goal is to fine-tune the pre-trained velocity function v_θ to maximize the expected reward over draws from the deterministically sampled generative model:

$$\arg\max_\theta \mathbb{E}_{\mathbf{c}\sim C,\,\mathbf{x}_0\sim\mathcal{N}(\mathbf{0},\mathbf{I})} R\big(f_\theta(\mathbf{x}_0;\mathbf{c})\big), \tag{2}$$

where C is the distribution of fine-tuning prompt embeddings, and $f_\theta(\cdot)$ represents the process of drawing a sample starting from an initial noise $\mathbf{x}_0$ using velocity function v_θ. Geometrically, the goal is to redirect the flow velocities such that a larger mass of noises map to high-reward regions of the image space. Generally, the reward may be non-differentiable or highly discontinuous.

3.2 Markov Decision Process Formulation

As an alternative to ODE-based deterministic sampling, diffusion models also allow for SDE-based *stochastic* sampling under the same flow. Stochastic sampling injects fresh Gaussian noise at each time step to randomize the trajectory.

MDP based approaches [5, 15, 32, 64] take advantage of stochasticity and cast each stochastic step as a Gaussian-distributed action distribution whose mean is defined by the flow velocity. Beneficial actions are reinforced by pulling the velocity toward the random steps of high-reward trajectories.

Fig. 1a illustrates the group-relative approach used by Flow-GRPO [32] and DanceGRPO [64] algorithms. They adopt the denoising MDP for flow schedules and estimate the advantage of each trajectory by normalizing its corresponding

reward within a group. The advantages determine the weight of the corresponding updates to the flow. While these updates point towards more favorable images on average, a significant proportion of them point opposite of reward increase, contributing to undesirable variance.

4 Our Method

We adopt an approximate on-policy approach of alternating between (1) generating a set of trajectory rollouts from the current model, and (2) training the network velocity predictions along these trajectories, so as to redirect them towards collecting higher reward at their endpoints. To obtain a training signal without direct access to gradient of the reward, we propose to roll out *pairs* of images with variations in detail and to reinforce the probability of the more favorable image among the two.

Specifically, we generate a pair of rollouts $\mathbf{x}_i$ and $\hat{\mathbf{x}}_i$ from a shared initial noise $\mathbf{x}_0 = \hat{\mathbf{x}}_0 = \boldsymbol{\epsilon}$, but apply a modest amount of stochasticity that perturbs them along the sampling trajectory. This induces random differences in generated image details, whereby one of the output images $\{\mathbf{x}_T, \hat{\mathbf{x}}_T\}$ usually collects a higher reward than the other. Then, the difference vector $\Delta\mathbf{x} = \hat{\mathbf{x}}_T - \mathbf{x}_T$ weighted by the reward difference $\Delta R = R(\hat{\mathbf{x}}_T) - R(\mathbf{x}_T)$ will point towards the higher-reward image. We train the velocities at all time steps along both trajectories to bend towards $\Delta R \Delta\mathbf{x}$. The approach is illustrated in Fig. 1b.

4.1 Analysis

The weighted difference $\Delta R \Delta\mathbf{x}$ points towards desirable changes in the generated image at $t = 0$, whereas our updates divert the flow toward this direction at intermediate steps. Our method thus relies on the expectation that this induces a similar change in the image.

To justify this informally, we note that the denoising process essentially fades out noise to reveal a hidden signal in a coarse-to-fine fashion. Then, adding "signal", i.e., the reinforced image difference, at an intermediate noisy image will approximately pass it to the generated image. This is a core underlying assumption in a broad range of methods (e.g., [7, 8, 24, 38]) that perform coarse edits on a partially noised image (say, adding a blob of green pixels), and rely on the remainder of the flow to flesh out the detail while maintining the broad direction of the edit (say, into a tree).

More formally, our update aims to satisfy $\nabla R(\mathbf{x}_T)^\top \mathbf{J}_i(\mathbf{x}_i)\,[\Delta R \Delta\mathbf{x}] \geq 0$ for each step i, where $\mathbf{J}_i(\mathbf{x}_i)$ is the Jacobian matrix of the mapping induced by the flow from step i onward. While it *is* possible to construct counter-examples that violate this condition, there are significant arguments for it holding at least to the extent required in practical image generation. For example, it can be shown that under certain assumptions the condition holds on expectation if $\mathbf{J}_i$ is positive semi-definite (see Appendix C), which would be true for an optimal transport mapping [48]. The apparent similarity of diffusion flows and optimal transport

Algorithm 1 Stochastic flow matching sampler

procedure StochasticFlowSampler($\theta, \boldsymbol{\epsilon}, \mathbf{c}, \gamma_{0:T}$)
  $\mathbf{x}_0 \leftarrow \boldsymbol{\epsilon}$ ▷ Start from given noise
  for each $i \in \{0, \dots, T-1\}$ **do** ▷ Take T sampling steps
    $\tilde{t}_{i+1} \leftarrow t_{i+1}/(1 - \gamma_i t_{i+1} + \gamma_i)$ ▷ Overshoot
    $\mathbf{v}_i \leftarrow v_\theta(\mathbf{x}_i; t_i, \mathbf{c})$ ▷ Evaluate velocity
    $\tilde{\mathbf{x}}_{i+1} \leftarrow \mathbf{x}_i + (\tilde{t}_{i+1} - t_i)\mathbf{v}_i$ ▷ Step from t_i to $\tilde{t}_{i+1}$
    sample $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ▷ Draw fresh noise
    $\mathbf{x}_{i+1} \leftarrow \dfrac{\tilde{\mathbf{x}}_{i+1} + \tilde{t}_{i+1}\sqrt{\gamma_i^2 + 2\gamma_i} \cdot \boldsymbol{\epsilon}_i}{\gamma_i \tilde{t}_{i+1} + 1}$ ▷ Add noise to land at t_{i+1}, correct scale
  return $\mathbf{x}_{0:T}, \mathbf{v}_{0:T-1}$

mappings, while not theoretically exact [26], has inspired studies where a strong numerical similarity is nevertheless found [23].

4.2 Stochastic Sampling

Like Flow-GRPO, our approach requires a means to generate nearby random image variants. As discussed by Karras et al. [22], an Euler–Maruyama sampler (used in, e.g., Flow-GRPO) suffers from two related numerical problems when applied to flows. The velocity network is called with a slightly inconsistent time conditioning and noise amount at each step, as the impact of the noise injection sub-step is not accounted for. This is exacerbated by oversized noise injections that occur at some steps, because the amount of noise added is not proportioned to the existing noise level in the sample.

In Algorithm 1 we adapt the key idea from the EDM stochastic sampler [22] to introduce stochasticity into the flow-matching solution trajectories. When stepping from t_i to t_{i+1}, we take the ODE direction but overshoot the target time to a lower noise level, and then compensate by adding fresh random noise to land at t_{i+1} (with appropriate scale corrections, see Appendix B.1 for details).

The parameters γ_i specify the strength of stochastic randomization at each solver step. Roughly, γ_i expresses the fraction of noise in $\mathbf{x}_i$ that is re-randomized on that step, with value $\gamma_i = 0$ corresponding to deterministic flow matching. Intuitively, re-randomizing at an intermediate noise level replaces the to-be-generated finer image detail that is still encoded by the noise with a different random realization, while keeping the already resolved coarser detail intact.

4.3 Implementation Details

We present the full method pseudocode in Appendix B.

Stochasticity strength Where not specified otherwise, we use a uniform stochasticity schedule of $\gamma_i = 0.0025$ for all timesteps. This induces variations in semantic detail of an image pair (e.g., moving or swapping out parts), but mostly retains their overall layout and content. We experimented with more complex schedules but found no consistent benefit to them. See Appendix B.2 for details.

Normalization Neural network training benefits from approximately uniform gradient magnitude among training samples. The random stochastic perturbations result in different magnitudes for $\Delta\mathbf{x}$ between rollouts, and we can also expect ΔR to be approximately (to the first order) proportional to $\Delta\mathbf{x}$. The raw training signal $\Delta R\Delta\mathbf{x}$ thus counts the norm $\|\Delta\mathbf{x}\|_{\mathrm{RMS}}$ twice into its magnitude. We cancel this effect by using normalized $\overline{\Delta\mathbf{x}} = \Delta\mathbf{x}/(\|\Delta\mathbf{x}\|_{\mathrm{RMS}}^2 + 10^{-6})$ as the actual training signal.

Batching We train the model for up to 1000 epochs. For each epoch, we generate 432 pairs of rollouts with randomly drawn prompts and store all $T = 40$ steps of their trajectories, resulting in the same memory consumption as Flow-GRPO. The resulting $432\times2\times40$ samples are then randomly split into 4 training batches, totaling 8640 samples/batch. The weighted image differences are backpropagated through the velocity function and accumulated, until they are updated into model weights θ after each batch using AdamW [35].

On-policy optimization Making several training steps on a frozen set of rollouts throughout the epoch is sample-efficient, as computing fresh rollouts frequently would be expensive. However, it runs the risk of training on stale data, as the parameters θ may have changed significantly since the set of rollouts was refreshed. Following the usual RL policy optimization recipe, we apply clipping to downweight updates to velocities that have moved too far from their original position since the last refresh. We use Simple Policy Optimization (SPO) [61] clipping that is similar to the widely used Proximal Policy Optimization (PPO-Clip) [49].

5 Results

We will now evaluate our proposed RL algorithm experimentally to compare its performance against Flow-GRPO [32], a current state-of-the-art method. We start by considering the effectiveness of our method in increasing the proxy reward as quickly as possible, using rewards of different complexity to highlight differences between the algorithms. In these tests, we are mainly interested in convergence speed, highest obtainable reward, and potential algorithm-dependent artifacts.

We then employ external control metrics to analyze how the rewards, RL algorithms, and CFG strength affect image quality, prompt alignment, and diversity. We ablate the major design choices of our algorithm as well as the key differences to Flow-GRPO. Finally, we demonstrate on-policy human-in-the-loop training with human feedback as the live reward. Additional results, metrics, and details for all experiments are provided in Appendix A.

5.1 Experiment Setup

We use the official implementation of Flow-GRPO [32]. We explored its hyperparameters to a considerable extent and concluded that the official defaults are close to optimal in all cases. We thus use them with the following exceptions. First, we

disable exponential moving averaging of model weights as it was strictly harmful. Second, unless stated otherwise, we disable KL regularization [32] and CFG [19] because they can mask the differences between RL algorithms and ideally would not be needed. Third, we use 40 sampling steps during both training and inference. The role of CFG is explored in Section 5.3, and reducing the number of sampling steps during training is evaluated in Section 5.5.

We perform all experiments using Stable Diffusion 3.5 Medium [13], extended with low-rank adaptation layers (LoRA) [20] and update only these layers during post-training, in line with Flow-GRPO. We generate all images at 512×512 resolution. The number of model and reward evaluations per epoch is the same for Flow-GRPO and our method.

We use two primary reward functions, PickScore [45] and a novel VLM reward, that focus on predicting human preferences and prompt alignment, respectively. Both are evaluated using the Pick-a-Pic [25] training prompts. Our use of a VLM reward is motivated by the observation made by Lin et al. [29] that VLMs are surprisingly effective in estimating things like prompt alignment compared to explicit human preference models, and their performance can be expected to further improve as new models are released. To this end, we adopt a scheme similar to VQAScore [29] using Qwen2.5-VL-7B-Instruct [57]. We feed in the generated image along with a text query *"Does this image match the caption "$\cdots$"? Answer Yes or No."*, where $\cdots$ corresponds to the prompt that was used to generate the image. We then run the VLM to predict the logits for the next token and calculate the reward as $100 \cdot \text{sigmoid}(\text{logits[Yes]} - \text{logits[No]})$. We also consider the weighted combination of these two rewards, defined as PickScore + VLM alignment / 10.

For evaluation, we use the reward itself as well as a set of external control metrics, including OneIG-Bench [6] and HPSv2 [59]. We calculate these after-the-fact based on checkpoints exported during training, so that the evaluation protocol remains independent of the post-training algorithm.

5.2 Reward Optimization

Fig. 2 compares our method with Flow-GRPO when optimizing three different rewards: (a) PickScore, (b) VLM-based prompt alignment, and (c) their weighted combination. In this test, an epoch is equally expensive for both methods. PickScore is the easiest to optimize and both methods work well, although our method reaches a slightly higher reward much faster. The result achieved by Flow-GRPO matches the original paper closely.

The VLM-based alignment reward is more difficult to optimize; Flow-GRPO starts to struggle and the gap to our method widens significantly. The combined reward paints a similar picture with our method converging significantly faster to a higher reward. More complicated rewards consisting of multiple goals can be beneficial as they can reduce the aspects the RL optimization is blind to.

Fig. 3 shows representative image progressions for the combined reward. The images are drawn from training snapshots corresponding to the dots annotated in Fig. 2 and demonstrate that higher rewards correspond to visually better results,

as expected. While subjective comparison is inherently difficult, we feel that our method stacks well against Flow-GRPO in reward-matched pairwise comparison. Furthermore, there is a clear style drift in the Flow-GRPO images — this was a consistent behavior with all the prompts we experimented with. We attribute this style drift to Flow-GRPO's more noisy flow updates that lead to a greater amount of random mutation of the flow until a given reward is reached.

We also note that Flow-GRPO eventually starts introducing grid-like reward hacking artifacts that fade in and out during training. Fig. 4 shows handpicked examples where this effect is severe. The fact that these artifacts again disappear over time implies that the combined reward likely does penalize them, but not very strongly. We have never seen these artifacts in our method, even after equally long training.

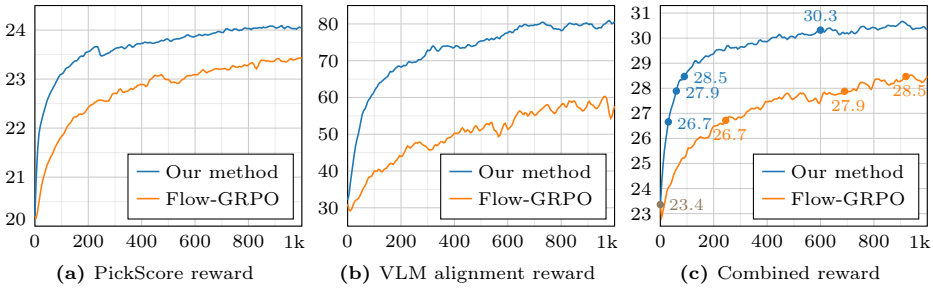
5.3 Control Metrics

Let us now further analyze the post-trained models to see how the choice of reward affects alignment with prompts, alignment with human preferences, and diversity. To estimate these, we use OneIG-Bench [6] (prompt alignment, diversity) and HPSv2 [59] (human preference) as independent control metrics.

Fig. 5a shows that the prompt alignment control metric is poorly optimized by the PickScore reward alone, as could be expected. Our VLM reward, which specifically targets this aspect, does a much better job, but the combined reward works even more reliably. Same conclusions apply to both our algorithm and Flow-GRPO, and the benefits of our method that were observed with the raw reward (Fig. 2b) also hold for the control metric. Fig. 5b shows that human preferences are well optimized by PickScore that focuses on this aspect. VLM reward does not work particularly well here, but again the combined reward comes out on top using our algorithm. Fig. 5c shows that all rewards reduce diversity, with PickScore causing the fastest decline and VLM reward the slowest. Considering that our algorithm optimizes the rewards much faster than Flow-GRPO, diversity loss is approximately equally fast for the two algorithms on an iso-reward basis. These observations position the combined reward as a clear choice over the more commonly used PickScore.

Fig. 6 uses the same control metrics to assess the effects of CFG. In these tests, we enable CFG for the base model and redo the entire RL post-training and evaluation using the CFG-adjusted velocity predictions in place of the original ones. We can see that increasing the guidance strength improves alignment with prompts and human preferences, at the cost of further diversity loss. This is not surprising, and the observations apply equally to our method and Flow-GRPO.

Fig. 7 provides visual examples of this tradeoff. Before post-training, the image quality of the base model is poor without CFG (Fig. 7a). Enabling CFG in that setting (Fig. 7b) improves prompt alignment and image quality dramatically while reducing diversity (see Fig. 6abc, epoch 0). With post-training, it is debatable whether CFG is useful, especially given its high inference cost. Subjective inspection of the result images suggests that post-training without CFG improves image fidelity and detail significantly (Fig. 7c), and the use of CFG

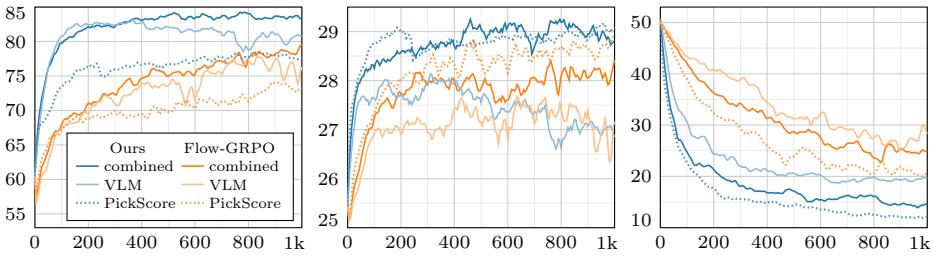


(a) PickScore reward **(b)** VLM alignment reward **(c)** Combined reward

Fig. 2: Comparison between our method and Flow-GRPO using different reward functions. Each plot shows the reward as a function of RL epoch. KL regularization and CFG are disabled. Example images for the highlighted checkpoints of combined reward are shown visually in Fig. 3. **(a)** PickScore reward. Flow-GRPO reaches 23.43 after 1000 epochs (2000 training steps). **(b)** Our VLM alignment reward: *Does this image match the caption "⋯ "? Answer Yes or No.* **(c)** Combined reward: PickScore + VLM alignment / 10.

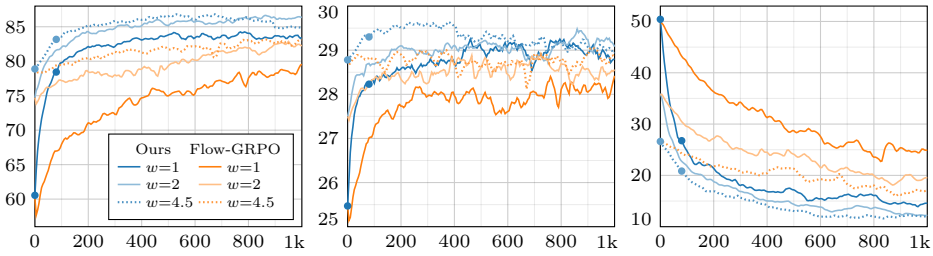
Fig. 3: Visual results for the combined reward using the prompt *"A girl with pigtails is holding a giant sunflower."* The epoch numbers (bottom-right corners) correspond to the dots in Fig. 2c, approximately matching the reward between the methods. The last column: Flow-GRPO is unable to increase the reward enough to match our method.

Fig. 4: When trained long enough, Flow-GRPO starts exhibiting grid-like artifacts that come and go, but are very disturbing in some epochs (here 830). Our results do not show these at any stage; here epoch 70 serves as an iso-reward comparison. Left: *"A green tractor plowing a field at sunrise."* Right: *"A quaint cottage nestled in a vibrant flower-filled meadow."* Combined reward, no CFG or KL regularization.



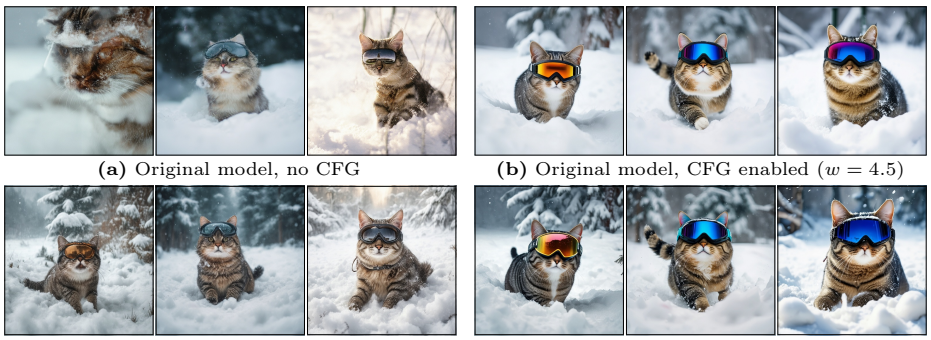
(a) OneIG-Bench prompt align. **(b)** HPSv2 **(c)** OneIG-Bench diversity

Fig. 5: Control metrics for our three rewards (combined, VLM, PickScore) as a function of RL epoch. **(a)** Independent assessment of prompt alignment. **(b)** Independent assessment of human preferences. **(c)** Independent assessment of result diversity.



(a) OneIG-Bench prompt align. **(b)** HPSv2 **(c)** OneIG-Bench diversity

Fig. 6: Control metrics without CFG ($w = 1$), and with medium ($w = 2$) and high ($w = 4.5$) guidance strength as a function of RL epoch using the combined reward. CFG drastically changes the results before post-training (epoch 0) but the gap narrows over time. Results for the highlighted checkpoints are shown visually in Figure 7.



(a) Original model, no CFG **(b)** Original model, CFG enabled ($w = 4.5$)

(c) 80 epochs of post-training, no CFG **(d)** 80 epochs of post-training, CFG enabled

Fig. 7: Visual results for prompt alignment vs. diversity: *"A cat wearing ski goggles is exploring in the snow."* **(a)** Without CFG, the original model has high diversity but poor quality and alignment. **(b)** Enabling CFG flips all these axes, leading to a distinctive low-detail look. **(c)** Our RL post-training achieves similar diversity and alignment with more detail. **(d)** Enabling CFG simplifies images and reduces diversity slightly. The used snapshots align with the dots in Fig. 6.

Ablations	OneIG align	
	200 ep	max
A Flow-GRPO	71.06	79.56
B + Our stochastic sampling	76.74	82.94
C + Finite difference flow opt	**82.14**	83.07
D + Start from the same noise	81.11	83.95
E + Cover more prompts (Ours)	82.13	**84.25**
E Our method	82.13	**84.25**
F + One trajectory is deterministic	79.76	84.02
G + PPO clipping instead of SPO	81.99	83.91
H + Do not normalize diff vector	79.74	80.78
I + Compute true reward gradient	81.47	83.73
J + Backprop through SDE steps	66.69	68.78

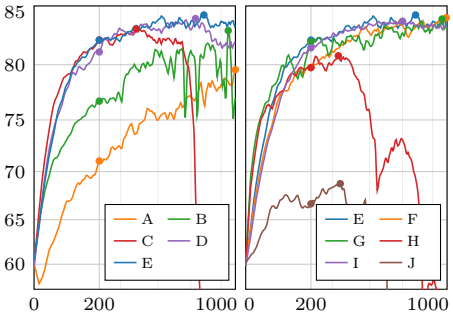


Fig. 8: OneIG-Bench alignment score for ablations of our method using the combined reward. The plots correspond to the top and bottom section of the table, respectively. We report the alignment score at 200 epochs as well as the highest achieved score, annotated in the plots.

in this setting mainly compromises diversity and introduces the characteristic, high-contrast look (Fig. 7d).

5.4 Ablations

Figure 8 presents a number of ablation experiments, starting with the Flow-GRPO baseline (row A). The amount of stochasticity is tuned separately for each ablation variant. Switching to our stochastic sampler in Algorithm 1 (row B) improves convergence considerably. Replacing the Flow-GRPO policy gradients with our finite difference flow optimization (row C), while keeping the number of unique prompts per epoch unchanged, results in another major boost. We then start all trajectories in each group from the same initial noise (row D), which improves stability and improves the results further. Finally, we can increase the number of unique prompts considered in each epoch by virtue of needing only two rollouts per prompt, which leads to our full method (row E).

The bottom section of the table evaluates some of our specific design choices. Instead of using stochastic sampling for both trajectories (row E), we can equally well use deterministic sampling for one of them (row F), which may be desirable in order to reduce the discrepancy between rollout generation and inference-time sampling. Using PPO [49] (row G) in place of SPO [61] (row E) yields similar results, although only when using the interval stochasticity schedule (Appendix B.2) — this was the only experiment where a constant γ_i schedule was clearly inferior to a more complex schedule. It is beneficial to normalize $\Delta\mathbf{x}$ as described in Section 4.3; disabling this (row H) is highly detrimental to stability. Replacing $\Delta R\Delta\mathbf{x}$ with the actual reward gradient obtained by backpropagating through the reward model (row I) does not improve the results, implying that our method is a good fit regardless of whether the reward is differentiable or not. Further backpropagating this reward gradient through the sampling trajectory to determine the update for each sampling step (row J) is significantly worse than our approach of bending all sampling steps towards $\Delta R\Delta\mathbf{x}$.

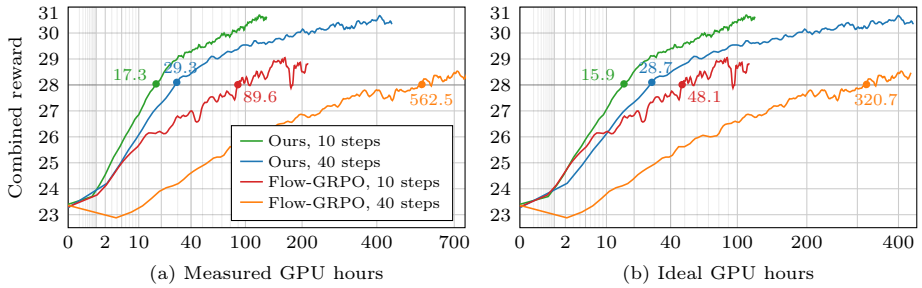


Fig. 9: Wall-clock time comparison (NVIDIA H200 hours). **(a)** Actual elapsed time. **(b)** Assuming ideal, zero-overhead implementation of both methods. Note the different horizontal scales.

Fig. 10: Qualitative progress across 50 post-training epochs with human rewards. Left: *"A small baby bird on a piece of metal."* Right: *"A cat floating inside the international space station."*

5.5 Wall-Clock Performance

Fig. 9a plots convergence as a function of GPU hours, comparing our method and Flow-GRPO in baseline (40 sampling steps) and fast (10 steps) configurations. As can be seen, our method converges significantly faster, crossing the highlighted combined reward level 19× faster in the baseline configuration and 5× faster in the fast configuration. Because the official Flow-GRPO implementation suffers from significant implementation overheads, we also estimated zero-overhead convergence curves (Fig. 9b) to confirm that our wall-clock advantage is not merely a result of a more careful implementation.

5.6 Direct Human Reward Function

As with other RL methods, our results depend heavily on the quality of proxy rewards. In principle, the proxies can be eliminated in favor of direct feedback from a human in the training loop, assuming the optimization is sample efficient enough to converge in reasonable amount of time. This leads to a method that uses human preferences on *on-policy data*, instead of fine-tuning using a pre-collected dataset [56].

To prototype this design, we ran a closed-loop generate-label-train procedure in which one of the authors provided live pairwise feedback on on-policy generations over ∼4 hours (3,200 image pairs generated using HPDv2 [59] prompts). We used neither reward models nor offline preference data. Fig. 10 shows the model's

samples across 50 rounds of reward collection and training (epochs). This was possible because our method combines three key properties: high sample efficiency (4h would have been nowhere near enough for FlowGRPO-like methods), pairwise comparison (update direction from a single human-provided preference), and support for non-differentiable rewards (human preferences).

6 Discussion and Future Work

Our method (FDFO) presents a step forward in RL post-training of diffusion models. It is a direct replacement to Flow-GRPO with faster convergence and higher-quality results. By leaving the multi-step MDP formulation, we shorten the reward attribution horizon and expand the algorithmic design space.

Our evaluation is focused on comparing RL algorithms with as few confounding factors as possible, and thus we disabled KL regularization in our main experiments. However, it remains compatible with our algorithm, as demonstrated in further experiments in Appendix A. KL regularization is typically used to better retain variation in the results, but the best way to do this in general remains an open problem. Finding a way to formulate a VLM-based reward component that targets diversity directly could be an interesting avenue for future work.

Acknowledgments We extend our thanks to Tero Kuosmanen and Samuel Klenberg for maintaining our compute infrastructure.

References

1. Albergo, M., Boffi, N., Vanden-Eijnden, E.: Stochastic interpolants: A unifying framework for flows and diffusions. JMLR **26**(209) (2025)
2. Albergo, M.S., Vanden-Eijnden, E.: Building normalizing flows with stochastic interpolants. In: Proc. ICLR (2023)
3. Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., Mané, D.: Concrete problems in AI safety. CoRR **abs/1606.06565** (2016)
4. Bengio, E., Jain, M., Korablyov, M., Precup, D., Bengio, Y.: Flow network based generative models for non-iterative diverse candidate generation. In: Proc. NeurIPS (2021)
5. Black, K., Janner, M., Du, Y., Kostrikov, I., Levine, S.: Training diffusion models with reinforcement learning. In: Proc. ICLR (2024)
6. Chang, J., Fang, Y., Xing, P., Wu, S., Cheng, W., Wang, R., Zeng, X., Yu, G., Chen, H.B.: OneIG-Bench: Omni-dimensional nuanced evaluation for image generation. In: Proc. NeurIPS (2025)
7. Chen, S.X., Vaxman, Y., Ben Baruch, E., Asulin, D., Moreshet, A., Lien, K.C., Sra, M., Sen, P.: TiNO-Edit: Timestep and noise optimization for robust diffusion-based image editing. In: Proc. CVPR (2024)
8. Choi, J., Kim, S., Jeong, Y., Gwon, Y., Yoon, S.: ILVR: Conditioning method for denoising diffusion probabilistic models. In: Proc. ICCV (2021)
9. Clark, K., Vicol, P., Swersky, K., Fleet, D.J.: Directly fine-tuning diffusion models on differentiable rewards (DRaFT). In: Proc. ICLR (2024)

10. Delbracio, M., Milanfar, P.: Inversion by direct iteration: An alternative to denoising diffusion for image restoration. TMLR (2023)
11. Domingo-Enrich, C., Drozdzal, M., Karrer, B., Chen, R.T.Q.: Adjoint matching: Fine-tuning flow and diffusion generative models with memoryless stochastic optimal control. In: Proc. ICLR (2025)
12. Dong, H., Xiong, W., Goyal, D., Zhang, Y., Chow, W., Pan, R., Diao, S., Zhang, J., Shum, K., Zhang, T.: RAFT: Reward ranked finetuning for generative foundation model alignment. TMLR (2023)
13. Esser, P., Kulal, S., Blattmann, A., Entezari, R., Müller, J., Saini, H., Levi, Y., Lorenz, D., Sauer, A., Boesel, F., Podell, D., Dockhorn, T., English, Z., Rombach, R.: Scaling rectified flow transformers for high-resolution image synthesis. In: Proc. ICML (2024), `https://huggingface.co/stabilityai/stable-diffusion-3.5-medium` (accessed 6/23/2026)
14. Fan, J., Shen, S., Cheng, C., Chen, Y., Liang, C., Liu, G.: Online reward-weighted fine-tuning of flow matching with Wasserstein regularization. In: Proc. ICLR (2025)
15. Fan, Y., Watkins, O., Du, Y., Liu, H., Ryu, M., Boutilier, C., Abbeel, P., Ghavamzadeh, M., Lee, K., Lee, K.: DPOK: Reinforcement learning for fine-tuning text-to-image diffusion models. In: Proc. NeurIPS (2023)
16. Guo, Y., Yuan, H., Yang, Y., Chen, M., Wang, M.: Gradient guidance for diffusion models: An optimization perspective. In: Proc. NeurIPS (2024)
17. Heitz, E., Belcour, L., Chambon, T.: Iterative α-(de)blending: A minimalist deterministic diffusion model. In: Proc. SIGGRAPH (2023)
18. Ho, J., Jain, A., Abbeel, P.: Denoising diffusion probabilistic models. In: Proc. NeurIPS (2020)
19. Ho, J., Salimans, T.: Classifier-free diffusion guidance. In: Proc. NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications (2021)
20. Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W.: LoRA: Low-rank adaptation of large language models. In: Proc. ICLR (2022)
21. Jia, Z., Nan, Y., Zhao, H., Liu, G.: Reward fine-tuning two-step diffusion models via learning differentiable latent-space surrogate reward. In: Proc. CVPR (2025)
22. Karras, T., Aittala, M., Aila, T., Laine, S.: Elucidating the design space of diffusion-based generative models. In: Proc. NeurIPS (2022)
23. Khrulkov, V., Ryzhakov, G., Chertkov, A., Oseledets, I.: Understanding DDPM latent codes through optimal transport. In: Proc. ICLR (2023)
24. Kim, G., Kwon, T., Ye, J.C.: DiffusionCLIP: Text-guided diffusion models for robust image manipulation. In: Proc. CVPR (2022)
25. Kirstain, Y., Polyak, A., Singer, U., Matiana, S., Penna, J., Levy, O.: Pick-a-Pic: An open dataset of user preferences for text-to-image generation. In: Proc. NeurIPS (2023), `https://huggingface.co/yuvalkirstain/PickScore_v1` (accessed 6/23/2026)
26. Lavenant, H., Santambrogio, F.: The flow map of the Fokker–Planck equation does not provide optimal transport. Appl. Math. Lett. **133** (2022)
27. Lee, K., Liu, H., Ryu, M., Watkins, O., Du, Y., Boutilier, C., Abbeel, P., Ghavamzadeh, M., Gu, S.: Aligning text-to-image models using human feedback. CoRR **abs/2302.12192** (2023)
28. Levine, S.: Reinforcement learning and control as probabilistic inference: Tutorial and review. CoRR **abs/1805.00909** (2018)
29. Lin, Z., Pathak, D., Li, B., Li, J., Xia, X., Neubig, G., Zhang, P., Ramanan, D.: Evaluating text-to-visual generation with image-to-text generation. In: Proc. ECCV (2024)

30. Lipman, Y., Chen, R.T.Q., Ben-Hamu, H., Nickel, M., Le, M.: Flow matching for generative modeling. In: Proc. ICLR (2023)
31. Liu, B., Shao, S., Li, B., Bai, L., Xu, Z., Xiong, H., Kwok, J.T., Helal, S., Xie, Z.: Alignment of diffusion models: Fundamentals, challenges, and future. ACM Comput. Surv. (2026)
32. Liu, J., Liu, G., Liang, J., Li, Y., Liu, J., Wang, X., Wan, P., Zhang, D., Ouyang, W.: Flow-GRPO: Training flow matching models via online RL. In: Proc. NeurIPS (2025), `https://github.com/yifan123/flow_grpo` (accessed 6/23/2026)
33. Liu, X., Gong, C., Liu, Q.: Flow straight and fast: Learning to generate and transfer data with rectified flow. In: Proc. ICLR (2023)
34. Liu, Z., Xiao, T.Z., Liu, W., Bengio, Y., Zhang, D.: Efficient diversity-preserving diffusion alignment via gradient-informed GFlowNets. In: Proc. ICLR (2025)
35. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: Proc. ICLR (2019)
36. Luo, G., Granskog, J., Holynski, A., Darrell, T.: Dual-process image generation. In: Proc. ICCV (2025)
37. McAllister, D., Ge, S., Yi, B., Kim, C.M., Weber, E., Choi, H., Feng, H., Kanazawa, A.: Flow matching policy gradients. In: Proc. ICLR (2026)
38. Meng, C., He, Y., Song, Y., Song, J., Wu, J., Zhu, J.Y., Ermon, S.: SDEdit: Guided image synthesis and editing with stochastic differential equations. In: Proc. ICLR (2022)
39. Nachum, O., Norouzi, M., Xu, K., Schuurmans, D.: Bridging the gap between value and policy based reinforcement learning. In: Proc. NeurIPS (2017)
40. Nakano, R., Hilton, J., Balaji, S., Wu, J., Ouyang, L., Kim, C., Hesse, C., Jain, S., Kosaraju, V., Saunders, W., Jiang, X., Cobbe, K., Eloundou, T., Krueger, G., Button, K., Knight, M., Chess, B., Schulman, J.: WebGPT: Browser-assisted question-answering with human feedback. CoRR **abs/2112.09332** (2021)
41. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C.L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., Lowe, R.: Training language models to follow instructions with human feedback. In: Proc. NeurIPS (2022)
42. Park, S., Li, Q., Levine, S.: Flow Q-learning. In: Proc. ICML (2025)
43. Peters, J., Schaal, S.: Reinforcement learning by reward-weighted regression for operational space control. In: Proc. ICML (2007)
44. Psenka, M., Escontrela, A., Abbeel, P., Ma, Y.: Learning a diffusion model policy from rewards via Q-score matching. In: Proc. ICML (2024)
45. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., Sutskever, I.: Learning transferable visual models from natural language supervision. In: Proc. ICML (2021), `https://huggingface.co/laion/CLIP-ViT-H-14-laion2B-s32B-b79K` and `https://huggingface.co/openai/clip-vit-large-patch14` (both accessed 6/23/2026)
46. Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C.D., Finn, C.: Direct preference optimization: Your language model is secretly a reward model. In: Proc. NeurIPS (2023)
47. Rubinstein, R.: The cross-entropy method for combinatorial and continuous optimization. Method. Comput. Appl. Prob. **1**(2) (1999)
48. Santambrogio, F.: Optimal Transport for Applied Mathematicians: Calculus of Variations, PDEs, and Modeling, Progress in Nonlinear Differential Equations and Their Applications, vol. 87. Birkhäuser (2015)

49. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR **abs/1707.06347** (2017)
50. Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., Ganguli, S.: Deep unsupervised learning using nonequilibrium thermodynamics. In: Proc. ICML (2015)
51. Song, J., Meng, C., Ermon, S.: Denoising diffusion implicit models. In: Proc. ICLR (2021)
52. Song, J., Zhang, Q., Yin, H., Mardani, M., Liu, M.Y., Kautz, J., Chen, Y., Vahdat, A.: Loss-guided diffusion models for plug-and-play controllable generation. In: Proc. ICML (2023)
53. Song, Y., Ermon, S.: Generative modeling by estimating gradients of the data distribution. In: Proc. NeurIPS (2019)
54. Song, Y., Sohl-Dickstein, J., Kingma, D.P., Kumar, A., Ermon, S., Poole, B.: Score-based generative modeling through stochastic differential equations. In: Proc. ICLR (2021)
55. Uehara, M., Zhao, Y., Biancalani, T., Levine, S.: Understanding reinforcement learning-based fine-tuning of diffusion models: A tutorial and review. CoRR **abs/2407.13734** (2024)
56. Wallace, B., Dang, M., Rafailov, R., Zhou, L., Lou, A., Purushwalkam, S., Ermon, S., Xiong, C., Joty, S., Naik, N.: Diffusion model alignment using direct preference optimization. In: Proc. CVPR (2024)
57. Wang, P., Bai, S., Tan, S., Wang, S., Fan, Z., Bai, J., Chen, K., Liu, X., Wang, J., Ge, W., Fan, Y., Dang, K., Du, M., Ren, X., Men, R., Liu, D., Zhou, C., Zhou, J., Lin, J.: Qwen2-VL: Enhancing vision-language model's perception of the world at any resolution. CoRR **abs/2409.12191** (2024), `https://huggingface.co/Qwen/Qwen2.5-VL-7B-Instruct` (accessed 6/23/2026)
58. Watkins, C.J.C.H., Dayan, P.: Q-learning. Mach. Learn. **8**(3) (1992)
59. Wu, X., Hao, Y., Sun, K., Chen, Y., Zhu, F., Zhao, R., Li, H.: Human preference score v2: A solid benchmark for evaluating human preferences of text-to-image synthesis. CoRR **abs/2306.09341** (2023)
60. Wu, X., Hao, Y., Zhang, M., Sun, K., Huang, Z., Song, G., Liu, Y., Li, H.: Deep reward supervisions for tuning text-to-image diffusion models. In: Proc. ECCV (2024)
61. Xie, Z., Zhang, Q., Yang, F., Hutter, M., Xu, R.: Simple policy optimization. In: Proc. ICML (2025)
62. Xu, J., Liu, X., Wu, Y., Tong, Y., Li, Q., Ding, M., Tang, J., Dong, Y.: ImageReward: Learning and evaluating human preferences for text-to-image generation. In: Proc. NeurIPS (2023)
63. Xue, S., Ge, C., Zhang, S., Li, Y., Ma, Z.M.: Advantage weighted matching: Aligning RL with pretraining in diffusion models. CoRR **abs/2509.25050** (2025)
64. Xue, Z., Wu, J., Gao, Y., Kong, F., Zhu, L., Chen, M., Liu, Z., Liu, W., Guo, Q., Huang, W., Luo, P.: DanceGRPO: Unleashing GRPO on visual generation. CoRR **abs/2505.07818** (2025)
65. Zelikman, E., Wu, Y., Mu, J., Goodman, N.D.: STaR: Bootstrapping reasoning with reasoning. In: Proc. NeurIPS (2022)
66. Zhang, D., Zhang, Y., Gu, J., Zhang, R., Susskind, J.M., Jaitly, N., Zhai, S.: Improving GFlowNets for text-to-image diffusion alignment. TMLR (2025)