Stitched Value Model for Diffusion Alignment

Hyojun Go[1], **Hyungjin Chung** , **Prune Truong**[2], **Goutam Bhat**[2], **Li Mi**[1], **Zhaochong An**[3], **Zixiang Zhao**[1], **Dominik Narnhofer**[1], **Serge Belongie**[3], **Federico Tombari**[2] **and Konrad Schindler**[1]
[1]ETH Zurich, [2]Google, [3]University of Copenhagen

For practical use, diffusion- or flow-based generative models must be *aligned* with task-specific rewards, such as prompt fidelity or aesthetic preference. That alignment is challenging because the reward is defined for clean output images, but the alignment procedure requires value function estimates at noisy intermediate latents. Existing methods resort to Tweedie-style or Monte Carlo approximations, trading off estimator bias against computational cost: Tweedie estimates are efficient but biased, while Monte Carlo estimates are more accurate but require expensive rollouts. A natural alternative would be a learned value function, but it remains an open question how to effectively train a strong and general value model specifically for noisy latents. Here, we propose StitchVM (Stitched Value Model), a model stitching framework that efficiently transfers reward models pretrained for clean images to the noisy latent regime. StitchVM starts from an existing, truncated pixel-space reward model and attaches a frozen diffusion backbone to it as its head. From the pixel-space model, the resulting hybrid retains a carefully pretrained, robust reward capability; from the diffusion backbone, it inherits its native ability to handle noisy latents. The stitching procedure is exceptionally lightweight, e.g., stitching and finetuning CLIP ViT-L and SD 3.5 Medium takes only ≈10 GPU-hours. By lifting powerful pixel-space reward models to latent space, StitchVM opens up a new style of diffusion alignment: instead of rough, yet costly per-sample approximation of the value function, the correct function for the actual, noisy latents is constructed once and then amortized over many samples and iterations. We show that this approach yields improvements across a broad range of downstream steering and post-training methods: DPS becomes 3.2× faster while halving peak GPU memory, and DiffusionNFT becomes 2.3× faster.

1. Introduction

Diffusion [1–4] and flow-based [5–7] denoising models have enabled remarkable success in generative image modelling, including image [8–10], video [11–14], and 3D generation [15–17]. Still, the pretraining objective of these models captures the training data distribution, and in practice, task-specific adaptation is often required, e.g. to ensure fidelity to a user prompt [18] or to match human aesthetic preferences [19, 20]. This customization is achieved through *alignment*, which aims to adapt the pretrained diffusion or flow model according to a specific reward.

Most existing alignment methods, whether applied at training time [21–30] or at inference time [31–45], share a common requirement: they must repeatedly assess noisy latents $\mathbf{z}_t$ along the denoising trajectory to determine how promising they are. This information is captured by a *value function* [40, 46], which measures the *expected* reward of clean samples induced by $\mathbf{z}_t$.

Directly evaluating the value function is difficult, in large part because the reward is normally defined for clean images $\mathbf{x}_0$ [20, 47–50]. Therefore, existing methods must resort to workarounds: **(1) Tweedie approximation**, which first estimates the posterior mean of the clean sample induced by $\mathbf{z}_t$, then computes the reward for that proxy [31, 32, 51]; or **(2) Monte Carlo (MC) approximation**, which rolls out multiple denoising trajectories from $\mathbf{z}_t$ and averages the reward for each resulting clean sample [40, 46]. Both approaches have significant drawbacks: the Tweedie approximation can be substantially biased in the high-noise regime [52], moreover it requires an extra denoiser

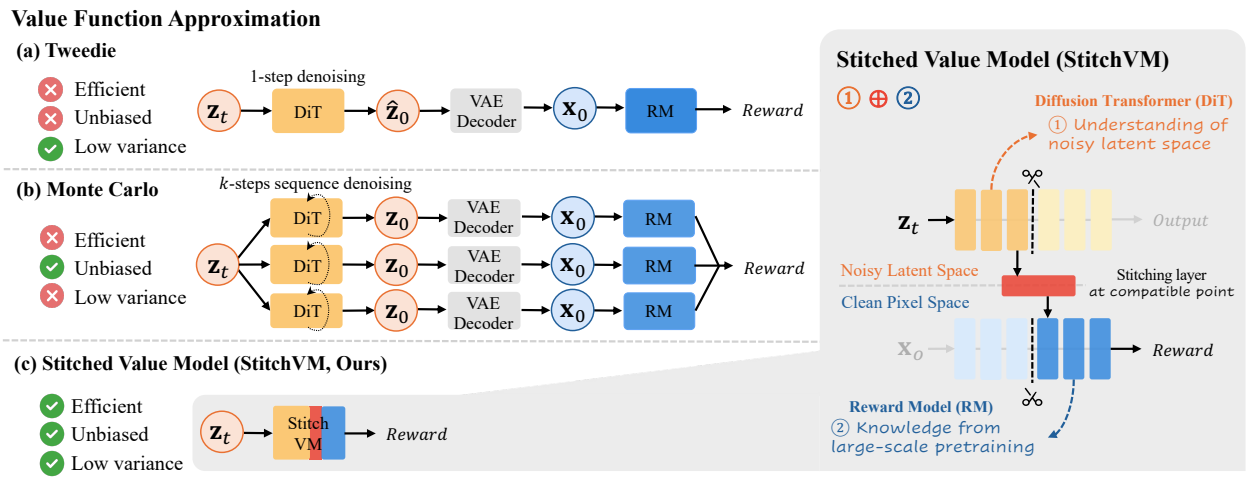


Figure 1 | **StitchVM overview. Left:** Unlike Tweedie (a), which requires a denoiser and VAE decoder evaluations, and is biased in high noise, and MC (b), which requires N-denoising rollouts, StitchVM (c) directly evaluates the value function on noisy latent. **Right:** StitchVM stitches a diffusion backbone head to a reward model tail, turning the reward model into a value model.

evaluation and VAE decoding; MC incurs high, often prohibitive, cost for the rollouts.

An alternative to these workarounds is to directly learn a value function for noisy latents [40, 53–56]. Once trained, such value models can be incorporated into both training-time and inference-time methods, improving alignment along both axes. In terms of accuracy, they avoid the bias of Tweedie and the inherent variance of stochastic MC rollouts [56]; in terms of efficiency, they eliminate both the extra denoiser and decoder evaluations required by Tweedie and the costly rollouts of MC [27, 55].

Despite these clear advantages, only few works have explored direct training of a value model. This is because substantial amounts of data and compute would be required to train a value function for noisy latents that could rival the performance and generality of contemporary pixel-space reward models [20, 47–50]. Beyond the prohibitive upfront cost, such an approach is fundamentally unsustainable: for each new diffusion backbone or improved reward model, one would have to repeat the full large-scale training. Therefore, existing works train at much smaller scales, either reusing diffusion features [54, 55, 57, 58] or initializing with pretrained reward models [19, 56, 59, 60] to reduce cost. Unfortunately, this leads to inferior accuracy and generalization compared to foundation-scale reward models defined in pixel space [20, 47–50]. Consequently, the trend has been to fall back to Tweedie or MC approximations, whereas direct value models were largely sidelined.

Here, we propose **StitchVM** (**Stitched Value Model**), a framework that transfers the capabilities of pretrained reward models into the noisy latent regime with only a small finetuning cost. Building on model stitching [61–65], our approach combines a truncated frozen diffusion backbone as "head"—natively able to handle noisy latents [57, 66]—with a sliced pretrained reward model as "tail", via a lightweight stitching layer (Fig. 1). The key is to identify a stitch point where the *representations are compatible*. One way to ensure that is to find layers where the diffusion features of the head can (almost) be mapped to the reward features of the tail with a linear transformation. Since the mapping can be fitted in closed form and the remaining representation gap is small, a short finetuning is sufficient to close the gap without harming the predictive skill of the reward model. In this way, the stitched model inherits the capability to predict the reward, but is able to operate directly on noisy latents and thus to serve as a value model.

StitchVM is remarkably effective with a range of different diffusion backbones (SD 3.5 Medium [67, 68], SD 3.5 Large [67, 68], FLUX [69]) and reward models (DFN-CLIP [70], CLIP [49], Aesthetic

Score Predictor [71], HPSv2 [20]). With only a few unlabeled images and lightweight finetuning, the stitched models retain the benchmark performance of the underlying clean reward models while directly ingesting noisy latents. Notably, transferring ViT-L/14@336px CLIP into an SD 3.5 Medium value function takes only ≈10 hours on a single GH200 GPU.

We test the stitched value models with various alignment methods. In case of inference-time alignment, the low-cost estimator for the value function lets each particle in FK steering [38] pick the best of several local proposals at each step, making it more efficient than standard particle scaling. Alternatively, it can replace the long gradient paths of DPS [31] with direct gradients from the value model, making the method 3.2× faster and halving peak GPU memory, while at the same time improving quality. For training-time alignment, our stitched value models enable training at intermediate noisy latents and avoid full rollouts: DiffusionNFT [30] becomes 2.3× faster, while direct reward finetuning [21, 24] becomes 1.3× faster, and more effective (e.g., +30% GenEval) through supervision at high-noise steps.

2. Related Work

Alignment methods and value function. Most existing inference-time alignment methods evaluate the value function on noisy latents *indirectly*, through approximations, in order to leverage pixel-level rewards. The *Tweedie approximation* [31, 32, 51] forms the basis of many guidance and sequential Monte Carlo methods [33–39, 41, 42, 72, 73], where the estimated clean sample is used either to compute guidance gradients or to weight particles. The *Monte Carlo approximation* [40, 46] instead evaluates the value function by averaging rewards over multiple denoising rollouts, as in SVDD [40] and search-based methods such as DSearch [43]. Training-time alignment follows a similar paradigm. Direct reward finetuning [21, 22, 74] propagates terminal rewards through denoising trajectories, while PPO-style methods [28, 29, 75–77] optimize policy objectives over sampled trajectories.

To avoid these approximations, several works propose to learn the value function directly for the noisy latent. These models have been used to improve credit assignment in PPO-style post-training [56, 59], provide reward feedback at high-noise timesteps in direct reward finetuning [55], and reduce rollout cost in search-based inference [60]. However, these value models are typically trained with a narrow preference corpus or with task-specific labels, giving reliable signals only in a narrow domain. We provide a broader discussion about alignment methods and the value function in Appendix B.

Training value models and noisy latent reward models. A primary concern when learning a value model, or more broadly a noisy latent reward model, has been to avoid impractical large-scale training. These efforts fall into two categories. *(1) Diffusion-feature predictors* attach prediction heads [54, 55, 57, 58] or LLM interfaces [78] to diffusion features. While naturally noise-aware [57, 66], their prediction heads are typically trained on narrow preference data and lack the broad generalization of foundational reward models. *(2) Adaptation of pretrained reward models* takes one of two routes. The first applies Tweedie-style one-step prediction [19] on top of clean-image reward models, but inherits Tweedie's bias. The second learns projections from noisy latents to the input space of a pretrained reward model [56, 59, 60, 79], introducing a distribution shift that small-data adaptation cannot fully bridge. Consequently, no practically tractable scheme based on noisy latents has yet been able to match the broad zero-shot capability of pretrained pixel-space reward models.

Model stitching. Originally introduced to study neural representations [61], model stitching re-composes the early layers of one neural network and the later layers of another one into a new network, usually with the help of an additional stitching layer. Beyond revealing similarities between representations that metrics such as CKA may miss [62, 64], it has been shown that even networks

with different architectures can often be stitched into hybrid models with minimal degradation [80], enabling applications such as resource-constrained model reassembly [63] and variable-scale network construction [65]. Recent work has begun to apply stitching to generative models: VIST3A [15] and VGGRPO [81] stitch 3D reconstruction networks [82, 83] onto *clean* latents. We extend this idea to the *noisy latent* regime and show that pretrained reward models can also be stitched directly to intermediate states of the denoising process.

3. Preliminary

3.1. Diffusion and Flow-based Models

Let $\mathbf{x}_0 \in \mathbb{R}^n \sim p_{\text{data}}$ denote a clean data sample. We consider the flow matching (FM) framework [5–7] in latent space [84], where $\mathbf{z}_0 = \mathcal{E}(\mathbf{x}_0) \in \mathbb{R}^d$ is the clean latent and $\mathcal{E}$ is the encoder of the latent diffusion model. Throughout this work, $t = 0$ corresponds to the clean latent distribution p_0 (with $\mathbf{z}_0 \sim p_0$) and $t = 1$ corresponds to the reference Gaussian $p_1 = \mathcal{N}(0, I_d)$ (with $\mathbf{z}_1 = \boldsymbol{\epsilon} \sim p_1$); note that this is the reverse of the convention in [5]. We define a Gaussian conditional probability path:

$$\mathbf{z}_t = \alpha_t \mathbf{z}_0 + \sigma_t \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, I_d) \Leftrightarrow p_t(\mathbf{z}_t|\mathbf{z}_0) = \mathcal{N}(\mathbf{z}_t; \alpha_t \mathbf{z}_0, \sigma_t^2 I_d), \tag{1}$$

where for FM, $\alpha_t = 1 - t, \sigma_t = t$. This induces a marginal probability path $p_t(\mathbf{z}_t)$ which interpolates between p_0 and p_1. FM models learn the marginal velocity field $u_t(\mathbf{z}_t) = \int u_t(\mathbf{z}_t|\mathbf{z}_0) p_{0|t}(\mathbf{z}_0|\mathbf{z}_t)\, d\mathbf{z}_0$, where $u_t(\mathbf{z}_t|\mathbf{z}_0) = \boldsymbol{\epsilon} - \mathbf{z}_0$ is the conditional velocity. To sample, one can resort to ODE $\frac{d}{dt}\mathbf{z}_t = u_t(\mathbf{z}_t)$, SDE [3], or discrete transition kernels [1, 85]. See Appendix A.1 for further discussion.

3.2. Alignment as reward tilting

Pretraining aims to model the data distribution. In many applications, however, we do not simply want likely samples—we seek samples that also score highly under some reward function[1] $r(\mathbf{x}_0) : \mathbb{R}^n \mapsto \mathbb{R}$ that encodes task-specific notions of sample quality, including prompt alignment [49], aesthetics [71], human preference [20, 47], and physical consistency [31, 46, 86]. A standard way to formalize alignment is through the reward-tilted target distribution [46]

$$p^\star(\mathbf{x}_0) = \frac{1}{Z_{\mathbf{x}}} p(\mathbf{x}_0) \exp(r(\mathbf{x}_0)) \quad \text{or, equivalently} \quad p^\star(\mathbf{z}_0) = \frac{1}{Z_{\mathbf{z}}} p(\mathbf{z}_0) \exp\left(r\left(\mathcal{D}(\mathbf{z}_0)\right)\right), \tag{2}$$

where $p(\mathbf{x}_0), p(\mathbf{z}_0)$ are the base prior distributions from pretraining, $\mathcal{D}$ is the decoder, and $Z_{\mathbf{x}}, Z_{\mathbf{z}}$ are the partition functions. While the reward is normally defined after decoding with $\mathcal{D}$, we often omit it and simply denote $r(\mathbf{z}_0)$ for simplicity. Although the generation corresponds to a trajectory through time, starting at $t = 1$, the reward is only defined at the terminal $t = 0$. Therefore, it is useful to define the *soft value function*:

$$V_t(\mathbf{z}_t) := \log \mathbb{E}[\exp(r(\mathbf{z}_0)) \mid \mathbf{z}_t], \tag{3}$$

where the expectation is over $\mathbf{z}_0 \sim p_{0|t}(\mathbf{z}_0 \mid \mathbf{z}_t)$. Value functions can be used in both inference-time steering and post-training, as discussed next.

Inference with gradient guidance. One can show (see Appendix A.2) that by modifying the velocity:

$$u_t^r(\mathbf{z}_t) = u_t(\mathbf{z}_t) + c_t \nabla_{\mathbf{z}_t} V_t(\mathbf{z}_t), \tag{4}$$

with c_t some constant, one can sample from the tilted distribution in Eq. (2). As $V_t(\mathbf{z}_t)$ is intractable, widely used gradient guidance methods [31, 34, 72] leverage Tweedie approximation, *i.e.*, $V_t(\mathbf{z}_t) \approx r(\mathbb{E}[\mathbf{z}_0|\mathbf{z}_t])$, which incurs a bias known as the Jensen gap [31].

[1] In practice, the reward function may include further inputs such as a prompt, we omit them for brevity.

Inference with particle sampling. Methods based on sequential Monte Carlo and search-based methods [39–45] evaluate the approximated value function of each particle and probabilistically decide whether to keep the particle or not. Several works again resort to the Tweedie approximation [38, 41, 42]. Others [40] approximate the value function with Monte Carlo (MC) samples, *i.e.*, $V_t(\mathbf{z}_t) \approx \log\left(\frac{1}{N}\sum_{i=1}^{N}\exp\left(r(\mathbf{z}_{0,i})\right)\right), \mathbf{z}_{0,i} \sim p_{0|t}(\mathbf{z}_0|\mathbf{z}_t)$. The former approaches again introduce bias, whereas the MC sampling leads to *high variance*, and requires a lot of compute.

Training with reinforcement learning (RL). The KL-regularized RL objective

$$\arg\max_{\theta} \mathbb{E}_{\mathbf{z}_0\sim p_\theta}[r(\mathbf{z}_0)] - D_{KL}(p_\theta||p) \tag{5}$$

yields the tilted distribution in Eq. (2). Diffusion sampling can be regarded as a Markov Decision Process [28]. Existing works that leverage RL post-training for diffusion models aim to optimize for Eq. (5) or variants of it [22, 30, 76]. Similar to inference-time methods, RL post-training also requires evaluation of the value function, which is normally approximated through MC roll-outs that tend to be unstable and incur high variance. See Appendix A.3 for a discussion.

4. Methodology

In this section, we present **StitchVM**, a stitching-based framework that inherits the strong capability of pretrained reward models in the noisy latent regime at small finetuning cost (Section 4.1). We then show how StitchVM improves inference-time (Section 4.2) and training-time (Section 4.3) alignment.

4.1. StitchVM

Diffusion backbones natively process noisy latents and extract useful features from them [57, 66]; while pretrained reward models, trained at foundation model scale, output precise, task-relevant rewards for a broad range of clean images. StitchVM combines the two through a lightweight stitching layer that aligns the diffusion features with the reward model's feature space. Specifically, given stitching indices (i, j), the stitched value model is defined as:

$$V_\omega^{(i,j)}(\mathbf{z}_t) = r_\phi^{\geq j}\left(s_\psi\left(u_\theta^{\leq i}(\mathbf{z}_t)\right)\right), \tag{6}$$

where $u_\theta^{\leq i}$ and $r_\phi^{\geq j}$ denote the diffusion backbone truncated at layer i and the reward model starting from layer j, respectively, and s_ψ is the stitching layer.

Stage 1: Selecting the stitching interface. A key decision is at which indices (i, j) to stitch, *i.e.*, where to hand over from the diffusion model to the reward model. To identify an interface with compatible representations, we exhaustively search a set of candidate indices. Given a clean image $\mathbf{x}_0$ and its latent $\mathbf{z}_0 = \mathcal{E}(\mathbf{x}_0)$, we sample $\mathbf{z}_t \sim p_t(\mathbf{z}_t|\mathbf{z}_0)$ using Eq. (1) and extract paired features $u_\theta^{\leq i}(\mathbf{z}_t)$ and $r_\phi^{\leq j-1}(\mathbf{z}_0)$. For each candidate pair (i, j), we fit a linear mapping W by feature matching:

$$W_{i,j}^\star = \arg\min_W \mathbb{E}_{\mathbf{z}_0,t,\epsilon}\left[\left\|Wu_\theta^{\leq i}(\mathbf{z}_t) - r_\phi^{\leq j-1}(\mathbf{z}_0)\right\|_2^2\right]. \tag{7}$$

The optimization can be solved in closed form, making it practical to evaluate many candidate pairs. We then select the pair $(i^\star, j^\star)$ with the lowest feature-matching loss.

Stage 2: Finetuning StitchVM. Since $(i^\star, j^\star)$ are already chosen such that the representations are maximally compatible, the diffusion features (after linear transformation) lie close to the features expected by $r_\phi^{\geq j}$, leaving only a small mismatch. A short finetuning of the stitching layer s_ψ and the

truncated reward model $r_\phi^{\geq j}$ suffices to compensate that mismatch, without degrading the reward model's performance.

We finetune the stitched model using unlabeled clean images $\mathbf{z}_0$. For each $\mathbf{z}_0$, we sample a noisy latent $\mathbf{z}_t \sim p_t(\mathbf{z}_t|\mathbf{z}_0)$ from the forward process and use the score $r_\phi(\mathbf{z}_0)$ of the original reward model as the supervision target:

$$\mathcal{L}_{\text{value}}(\omega) = \mathbb{E}_{\mathbf{z}_0,\, t,\, \epsilon}\left[\left\| V_\omega^{(i^\star,j^\star)}(\mathbf{z}_t) - r_\phi(\mathbf{z}_0)\right\|_2^2\right]. \tag{8}$$

One can show that the minimizer of Eq. (8) satisfies $V_{\omega^\star}^{(i^\star,j^\star)}(\mathbf{z}_t) = \mathbb{E}[r_\phi(\mathbf{z}_0) \mid \mathbf{z}_t]$, *i.e.*, the value function. It is worth mentioning some design choices regarding Eq. (8). First, while on-policy regression is possible [40], we opt for an off-policy objective to further save compute[2]. Second, we choose to regress the standard value function rather than the soft value in Eq. (3), as this resulted in more stable training. One can show that, in terms of the reward scale, the two match to leading order. See Appendix A.4 for the proofs and further discussion.

Further details for the stitching architecture are given in Appendix C.1.

4.2. Inference-Time Alignment with StitchVM

StitchVM eliminates the additional denoiser and decoder evaluations required by the Tweedie approximation (Fig. 1). We show that, due to this saving, it improves Diffusion Posterior Sampling (DPS) [31] in both quality and speed, and Feynman-Kac (FK) steering [38] in quality.

DPS. DPS modifies the denoising velocity using the gradient of the value function $\nabla_{\mathbf{z}_t} V_t(\mathbf{z}_t)$, as in Eq. (4). Since the true value function is intractable, previous DPS schemes rely on the Tweedie approximation to estimate this gradient. As a result, the guidance signal inherits the bias of Tweedie and requires a long backpropagation chain through the reward model, the VAE decoder, and the denoiser. We instead use the gradient of StitchVM, $\nabla_{\mathbf{z}_t} V_\omega^{(i^\star,j^\star)}(\mathbf{z}_t)$, computed directly in the noisy latent space. This yields more accurate guidance, especially in high-noise regions while avoiding long backpropagation chains, making DPS both more effective and more efficient. Note the similarity to classifier guidance [87].

FK steering. Given particles $\{\mathbf{z}_{t_k}^n\}_{n=1}^N$, FK steering draws proposals $\bar{\mathbf{z}}_{t_{k-1}}^n \sim p_{t_{k-1}|t_k}(\mathbf{z}_{t_{k-1}}^n \mid \mathbf{z}_{t_k}^n)$ and computes potentials using a value function, e.g., $G(\mathbf{z}_{t_k}^n, \bar{\mathbf{z}}_{t_{k-1}}^n) = \exp(V_t(\bar{\mathbf{z}}_{t_{k-1}}^n) - V_t(\mathbf{z}_{t_k}^n))$. The next particles are then obtained by resampling according to:

$$a_{t_{k-1}}^n \sim \text{Multinomial}(G(\mathbf{z}_{t_k}^1, \bar{\mathbf{z}}_{t_{k-1}}^1), \ldots, G(\mathbf{z}_{t_k}^N, \bar{\mathbf{z}}_{t_{k-1}}^N)), \quad \mathbf{z}_{t_{k-1}}^n = \bar{\mathbf{z}}_{t_{k-1}}^{a_{t_{k-1}}^n}. \tag{9}$$

In text-to-image generation, FK steering estimates the value function via the Tweedie approximation, requiring a denoiser evaluation and VAE decoding for each particle.

In contrast, StitchVM evaluates the value function with a single forward pass of $V_\omega^{(i^\star,j^\star)}$, avoiding both full denoiser inference and VAE decoding. Value function estimates become substantially cheaper, allowing us to increase the number of proposals per particle without significantly inflating compute. This exposes a new scaling axis: rather than increasing the number of particles N, we can increase the number of local proposals M per particle and select among them using the cheap StitchVM score. Concretely, at a designated set of steps, each particle spawns M proposals $\mathbf{z}_{t_{k-1}}^{n,m} \sim p_{t_{k-1}|t_k}(\mathbf{z}_{t_{k-1}}^n \mid \mathbf{z}_{t_k}^n)$ for $m = 1, \ldots, M$. We then select the best proposal under StitchVM, $\bar{\mathbf{z}}_{t_{k-1}}^n = \mathbf{z}_{t_{k-1}}^{n,m_n^\star}$ with $m_n^\star = \arg\max_m V_\omega^{(i^\star,j^\star)}(\mathbf{z}_{t_{k-1}}^{n,m})$. For further details, see Appendix C.2.

[2] Off-policy and on-policy objectives have the same minimizer when the diffusion policy is exact.

4.3. Training-Time Alignment with StitchVM

Training-time alignment methods require full denoising rollouts to evaluate the reward at the final, clean image. Our StitchVM instead enables rollouts to stop at intermediate, noisy latents while still providing supervision through direct evaluation of the value function. We show how this improves and accelerates direct reward finetuning [21, 24], and also accelerates DiffusionNFT [30].

AlignProp & DRaFT. To maximize the objective in Eq. (5), direct reward finetuning methods like AlignProp [21] and DRaFT [24] roll out the full denoising trajectory to obtain $\mathbf{x}_0$ and backpropagate the reward gradient through the chain. In practice, this backpropagation is memory-intensive and prone to unstable or exploding gradients, so existing methods often restrict it to the final, low-noise steps. Our StitchVM avoids both issues: at each training iteration, we randomly sample a stopping timestep τ, halt the denoising at $\mathbf{z}_\tau$, and backpropagate using the StitchVM prediction $V_\omega^{(i^\star,j^\star)}(\mathbf{z}_\tau)$ in place of the terminal reward. This delivers effective supervision even in high-noise regions while avoiding long backpropagation chains. For further details, see Appendix C.3.

DiffusionNFT. DiffusionNFT [30] performs online RL on the forward process via flow matching, using terminal rewards from complete generations to define positive and negative samples, and thus an implicit direction for improving the policy. With StitchVM, we instead stop the generation early at an intermediate, noisy latent $\mathbf{z}_\tau$, evaluate its value function as $V_\omega^{(i^\star,j^\star)}(\mathbf{z}_\tau)$, and use that value in place of the terminal reward. This allows us to keep the original reward-weighted forward-process regression of DiffusionNFT, while moving supervision from clean, terminal outputs to intermediate, noisy latents. This provides supervision without having to generate all the way to a clean sample, substantially improving training efficiency. For further details, see Appendix C.4.

5. Experiments

Here, we show that StitchVM transfers pixel-space reward models into value models on noisy latents while inheriting the reward model's capability (Section 5.1). We then demonstrate that our StitchVM improves both inference-time (Section 5.2) and training-time (Section 5.3) alignment methods.

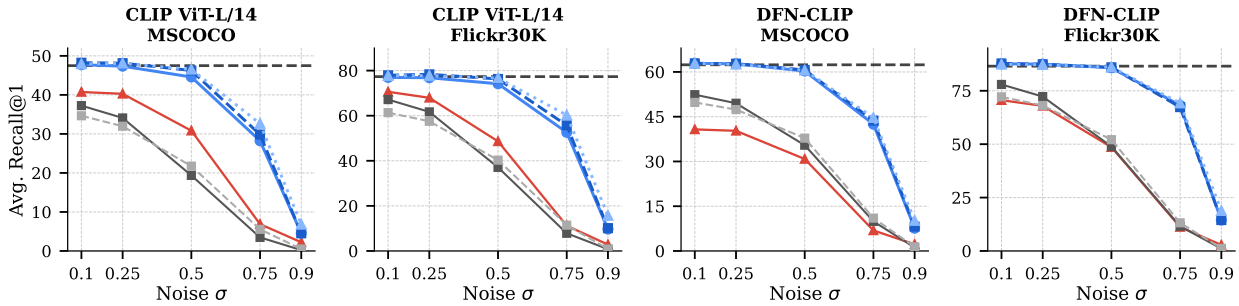
5.1. Main Results: StitchVM Performance

We evaluate the proposed StitchVM across three diffusion backbones—SD 3.5 Medium, SD 3.5 Large [67, 68], and FLUX.1-dev [69]—and four reward models: OpenAI CLIP (ViT-L/14, 336px) [49], DFN-CLIP (ViT-H/14, 378px) [70], HPSv2 [20], and the Aesthetic predictor [71]. All StitchVMs are trained for 5 epochs on unlabeled images from AVA [88] and HPDv2 [20]. We evaluate each model on noisy latents drawn from the flow matching forward process at $\sigma \in \{0.1, 0.25, 0.5, 0.75, 0.9\}$. For CLIP-based models, we report zero-shot cross-modal retrieval on MSCOCO [89] and Flickr30K [90]; for HPSv2, preference accuracy on ImageReward [47] and HPDv2 [20]; and for the Aesthetic predictor, SRCC on the AVA test split following [91].

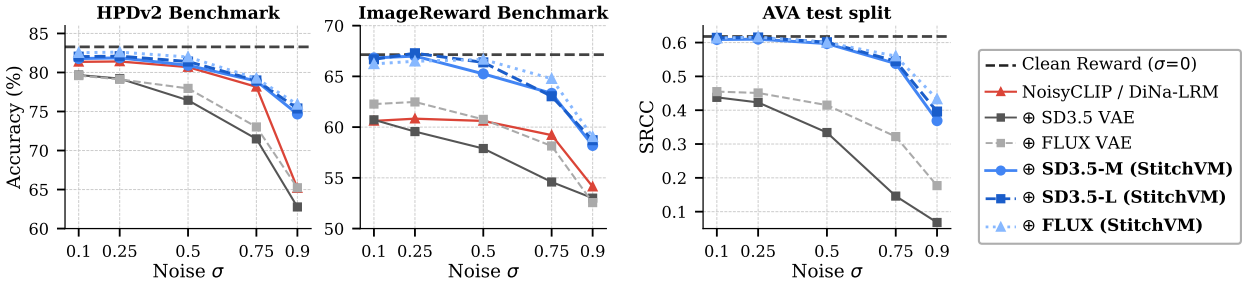
We compare against three baselines. First, we adapt VIST3A [15], which stitches to a VAE decoder rather than the diffusion backbone. Second, for CLIP-based reward models, we reimplement and train NoisyCLIP [79] at scale on LAION-400M [92]. Third, for HPSv2, we compare with DiNa-LRM [54], a diffusion-feature reward model trained on HPDv3 preference data [50]. Additional details are provided in Appendix D.1.

We report the main results in Fig. 2 (full table in Appendix E.1) and organize the findings as follows.

(1) Our StitchVM retains reward model capability on noisy latents. At low noise ($\sigma \leq 0.5$),



(a) Zero-shot image-text retrieval (Avg. Recall@1) on MSCOCO and Flickr30K.



(b) Preference accuracy on HPDv2 and ImageReward. (c) Aesthetic SRCC on AVA.

Figure 2 | **Results of StitchVM on latents with different noise levels.** ⊕ denotes stitching of a reward model with a pretrained diffusion module (VAE encoder or DiT).

StitchVM closely matches the performance of the original clean reward models across CLIP retrieval, HPSv2 preference prediction, and aesthetic prediction. As the noise level increases, StitchVM exhibits a gradual performance decline and remains substantially more robust than the baselines. Thus, StitchVM effectively converts existing pixel-space reward models into noisy latent value models, preserving their original capability while gaining robustness to intermediate noisy latents.

(2) Diffusion features are critical for robust transfer to noisy latents. StitchVM substantially outperforms the VAE stitching baseline across all noise levels, with the gap becoming especially large at high noise, where VAE stitching collapses. This comparison highlights the role of diffusion features: both methods stitch a pretrained reward model into the latent regime, but VAE stitching relies only on clean latent space, whereas StitchVM uses diffusion features that are trained to process noisy latents. This supports our strategy of stitching diffusion models with reward models.

(3) StitchVM outperforms noisy latent retraining and preference-data training. StitchVM outperforms NoisyCLIP [79] for CLIP-based reward models, despite NoisyCLIP being trained at a much larger scale on LAION-400M [92]. This shows that transferring a pretrained reward model through stitching is more effective than retraining the model on noisy latents from scratch. For HPSv2, StitchVM also outperforms DiNa-LRM [54] on both HPDv2 and ImageReward, despite using only unlabeled images rather than larger HPDv3 preference dataset [50]. Together, these comparisons show that StitchVM achieves robust noisy latent value function prediction by transferring the capability of pretrained reward models, without large-scale retraining or preference-label supervision.

5.2. Results on Inference-Time Methods

We evaluate StitchVM-enhanced DPS and FK steering (Section 4.2) with HPSv2, the Aesthetic predictor, and CLIP-based reward models. We measure ImageReward, Aesthetic Score, HPSv2, and PickScore [93] on generation from DrawBench [9] prompts, and additionally report GenEval [18] for

Table 1 | **Results of DPS with StitchVM on DrawBench.** ImgRwd: ImageReward, Aes: Aesthetic, Pick: PickScore, Mem: peak GPU memory (GB), Time: seconds per sample.

	SD3.5-Medium							SD3.5-Large						
Method	ImgRwd	Aes	HPSv2	Pick	CLIP	Mem ↓	Time ↓	ImgRwd	Aes	HPSv2	Pick	CLIP	Mem ↓	Time ↓
Flow baseline	0.95	5.31	0.283	22.56	28.86	—	—	1.07	5.44	0.294	22.82	29.07	—	—
HPSv2 Reward														
DPS	0.96	5.27	0.335	23.02	28.94	56.4	52.8	1.01	5.38	0.344	**23.31**	28.68	89.3	84.6
DPS + StitchVM	**1.22**	**5.43**	**0.348**	**23.03**	**28.95**	**26.0**	**16.5**	**1.20**	**5.45**	**0.356**	23.13	**28.97**	**43.2**	**36.3**
Aesthetic Reward														
DPS	0.82	5.73	0.280	**22.47**	28.16	54.3	54.2	0.91	5.71	0.280	22.41	27.95	87.3	84.8
DPS + StitchVM	**0.98**	**5.87**	**0.282**	22.44	**28.43**	**23.4**	**14.7**	**1.02**	**5.76**	**0.294**	**22.79**	**28.80**	**40.7**	**36.7**
CLIP Reward														
DPS	0.50	4.80	0.246	21.54	33.01	54.6	52.2	0.72	5.04	0.263	22.03	32.55	87.5	83.1
DPS + StitchVM	**0.68**	**4.82**	**0.249**	**21.63**	**33.12**	**23.9**	**14.6**	**1.05**	**5.20**	**0.279**	**22.44**	**32.95**	**41.5**	**36.3**

Table 2 | **Results of FK steering with StitchVM on DrawBench and GenEval.** We set $N = 4$ (number of particles). ImgRwd: ImageReward, Aes: Aesthetic, Pick: PickScore.

	SD3.5-Medium					SD3.5-Large					FLUX				
Method	ImgRwd	Aes	HPSv2	Pick	GenEval	ImgRwd	Aes	HPSv2	Pick	GenEval	ImgRwd	Aes	HPSv2	Pick	GenEval
Flow baseline	0.88	5.34	0.282	22.55	0.62	1.01	5.51	0.293	22.86	0.65	1.06	5.80	0.301	22.92	0.62
HPSv2 Reward															
BoN	0.91	5.26	0.284	22.28	0.63	**1.24**	5.41	0.308	22.96	0.68	1.15	**5.78**	0.314	23.13	0.65
FKS	0.93	5.26	0.283	22.24	0.62	1.22	5.39	0.308	22.96	0.68	1.17	5.76	0.314	23.10	0.66
FKS + StitchVM	**1.10**	**5.41**	**0.303**	**22.86**	**0.69**	1.20	**5.52**	**0.310**	**23.11**	**0.70**	**1.18**	5.74	**0.318**	**23.22**	**0.68**
Aesthetic Reward															
BoN	0.73	5.47	0.270	22.12	0.56	**1.06**	5.62	0.295	22.76	0.66	**1.07**	5.98	**0.301**	22.86	0.61
FKS	0.65	5.46	0.268	22.00	0.55	1.02	5.59	0.293	22.72	0.63	0.99	5.94	0.299	22.82	0.60
FKS + StitchVM	**0.99**	**5.58**	**0.289**	**22.65**	**0.65**	1.03	**5.68**	**0.298**	**22.87**	**0.68**	1.01	**6.00**	**0.301**	**22.88**	**0.64**
CLIP Reward															
BoN	0.73	5.13	0.266	22.00	0.58	1.16	5.32	0.294	22.75	0.67	1.09	5.73	0.301	22.94	0.63
FKS	0.79	5.11	0.267	22.03	0.59	1.18	5.30	0.295	22.79	0.68	1.08	5.71	0.302	22.96	0.64
FKS + StitchVM	**0.96**	**5.26**	**0.282**	**22.53**	**0.68**	**1.20**	**5.40**	**0.298**	**22.96**	**0.71**	**1.11**	**5.75**	**0.303**	**23.00**	**0.65**

FK steering. Additional details are provided in Appendix D.2.

(1) Our StitchVM makes DPS both more effective and more efficient. Table 1 shows that replacing the Tweedie-approximated DPS gradient with the StitchVM gradient improves DPS across nearly all reward–metric pairs on both SD 3.5 Medium and SD 3.5 Large, with only minor exceptions on PickScore. At the same time, StitchVM substantially reduces inference cost: peak GPU memory drops by about 50% (*e.g.*, 56.4 → 26.0 GB on SD 3.5 Medium), and sampling becomes up to 3.2× faster (52.8 → 16.5 s/sample). These gains come from the same source: StitchVM provides direct noisy latent gradients, avoiding both Tweedie approximation bias in high-noise regions and backpropagation through the denoiser, VAE, and pixel-space reward model.

(2) StitchVM improves FK steering. Table 2 shows that FK steering with StitchVM outperforms both standard FK steering (FKS) and Best-of-N (BoN) on most metrics, whereas FKS often fails to improve over BoN. The gains are especially large on SD 3.5 Medium: under HPSv2 reward, FK steering with StitchVM improves ImageReward from 0.93 (FKS) and 0.91 (BoN) to 1.10, and GenEval from 0.62 to 0.69. These gains come from StitchVM's low-cost value function evaluation: each evaluation requires only partial DiT inference up to the stitching layer plus the stitched reward model, and this partial DiT computation is shared with the next denoising step, so the marginal cost of additional proposals

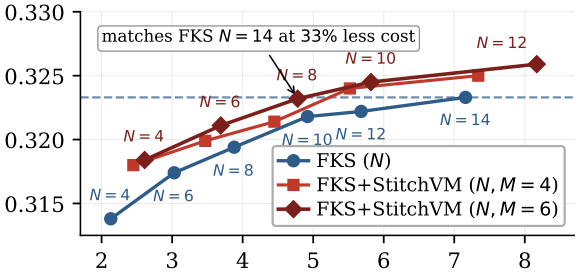


Figure 3 | **HPSv2 reward (target) vs. GPU-hours over 200 prompts on FK steering.**

Table 3 | **Training-time alignment results, with joint DFN-CLIP + HPSv2 as the training reward.**

Method	GPU-h ↓	HPSv2	DFN	ImgRwd	Pick	GenEval
Flow-GRPO-fast	122.2	0.348	0.408	1.44	22.77	0.65
DRaFT-1	128.1	0.308	0.379	0.89	22.65	0.53
DRaFT-3	128.0	0.329	0.392	1.36	21.36	0.66
DRaFT-1 + StitchVM	**94.8**	**0.348**	0.418	**1.47**	23.06	0.69
DRaFT-3 + StitchVM	100.3	0.347	**0.420**	**1.47**	**23.13**	**0.71**
DiffusionNFT	191.5	**0.347**	0.413	**1.50**	22.98	0.67
DiffusionNFT + StitchVM	**84.7**	**0.347**	**0.414**	**1.50**	**23.06**	**0.68**

is small. In contrast, achieving the same effect of M under the Tweedie approximation would require a full denoiser inference and VAE decoding for each proposal.

(3) M-scaling vs. N-scaling on FK steering. This low marginal cost opens a second scaling axis beyond the standard particle count N: each particle spawns M candidates and selects the best under StitchVM. In Fig. 3, we evaluate FK steering and its variant with StitchVM on FLUX with HPSv2 target reward by varying N and M. Across the compute range, FK steering with StitchVM lies above the standard N-scaling curve: for example, (N=8, M=6) matches standard FKS at N=14 with 33% lower cost. This shows that increasing local proposals M is a more computationally efficient axis than increasing N alone.

5.3. Results on Training-Time Methods

We finetune SD3.5 Medium at 512×512 resolution using DFN-CLIP and HPSv2 as training rewards. We compare DiffusionNFT and DRaFT-K, which backpropagates through the final K denoising steps [22], against their StitchVM-based variants with $K \in \{1, 3\}$. We also include Flow-GRPO-Fast [76] as a baseline. As metrics, we report GenEval and ImageReward, PickScore, HPSv2, and DFN-CLIP scores on DrawBench. Additional details are provided in Appendix D.3.

Results. Table 3 shows that StitchVM accelerates both DRaFT and DiffusionNFT, while additionally improving DRaFT's generation quality. By stopping rollouts at intermediate noisy latents and evaluating the value function directly, StitchVM reduces GPU-hours by 22–26% for DRaFT and by over 55% for DiffusionNFT. For DRaFT, this also improves quality: standard DRaFT restricts backpropagation to low-noise steps to avoid unstable gradients, whereas StitchVM provides direct value function supervision at intermediate latents, including high-noise regions. We further plot training curves in Appendix E.2. As shown in the results, StitchVM variants reach higher scores across all metrics with less compute.

5.4. Analysis

Training cost of StitchVM. Table 4 reports the total computational cost of StitchVM on SD 3.5 Medium with different reward models, measured on GH200 GPUs and including both the search for the stitching layer and the subsequent finetuning stage. Each StitchVM takes only ≈10 GPU-hours at 512×512 resolution, or 24–32 GPU-hours at 1024×1024. The timings underline that the lightweight, one-time transfer procedure of StitchLM is a lot more efficient than large-scale retraining of a reward model.

Table 4 | **StitchVM training cost (GPU-h).**

	512×512			1024×1024		
Reward	Train	Search	Total	Train	Search	Total
Aesthetic	6.4	0.6	7.0	23.1	1.1	24.2
CLIP	9.3	0.7	10.0	23.5	1.0	24.5
HPSv2	9.4	0.8	10.2	31.0	1.3	32.3

Additional results. In Appendix E, we further analyze the stitching layer search (Appendix E.3), demonstrate that a smaller StitchVM can guide a larger generator (Appendix E.4), and ablate when to stop rollouts for DiffusionNFT with StitchVM (Appendix E.5).

6. Conclusion

We have presented **StitchVM**, a recipe for transferring high-end pixel-space reward models into value functions for the noisy latents of a diffusion or flow model. The key idea is to stitch a frozen diffusion backbone to the pretrained reward model. To that end we find the most compatible layers in the two models, which by construction already includes fitting the stitching layer. After that, a light, self-supervised finetuning stage is sufficient to close the remaining representation gap. The resulting StitchVMs closely match the underlying original reward models but can be evaluated at noisy latent precursors of the generated pixel images. We have applied the stitched value functions both to inference-time alignment methods (FK steering and DPS) and to training-time alignment (direct reward finetuning and DiffusionNFT). Across both settings, replacing expensive per-sample approximations with direct evaluation of the StitchVM value function improves efficiency while maintaining or even improving alignment quality. More broadly speaking, StitchVM provides a generic template for combining a latent diffusion backbone with a pixel-space feedforward model without sacrificing the extensive pretraining of either. We believe that this practice may have important applications beyond diffusion model alignment.

References

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.

[2] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*, 2015.

[3] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *ICLR*, 2021.

[4] Hyojun Go, Kim Kim, Yunsung Lee, Seunghyun Lee, Shinhyeok Oh, Hyeongdon Moon, and Seungtaek Choi. Addressing negative transfer in diffusion models. In *NeurIPS*, volume 36, pages 27199–27222, 2023.

[5] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *ICLR*, 2023.

[6] Michael Samuel Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. In *ICLR*, 2023.

[7] Xingchao Liu, Chengyue Gong, and qiang liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *ICLR*, 2023.

[8] Black Forest Labs, Stephen Batifol, Andreas Blattmann, Frederic Boesel, Saksham Consul, Cyril Diagne, Tim Dockhorn, Jack English, Zion English, Patrick Esser, et al. FLUX. 1 Kontext: Flow matching for in-context image generation and editing in latent space. *arXiv preprint*, 2025.

[9] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. In *NeurIPS*, volume 35, 2022.

[10] Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Sheng-ming Yin, Shuai Bai, Xiao Xu, Yilei Chen, et al. Qwen-image technical report. *arXiv preprint*, 2025.

[11] Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint*, 2025.

[12] Thaddäus Wiedemer, Yuxuan Li, Paul Vicol, Shixiang Shane Gu, Nick Matarese, Kevin Swersky, Been Kim, Priyank Jaini, and Robert Geirhos. Video models are zero-shot learners and reasoners. *arXiv preprint*, 2025.

[13] Zhaochong An, Zirui Li, Mingqiao Ye, Feng Qiao, Jiaang Li, Zongwei Wu, Vishal Thengane, Chengzu Li, Lei Li, Luc Van Gool, et al. Video understanding: From geometry and semantics to unified models. *Machine Intelligence Research*, 2026.

[14] Zhaochong An, Menglin Jia, Haonan Qiu, Zijian Zhou, Xiaoke Huang, Zhiheng Liu, Weiming Ren, Kumara Kahatapitiya, Ding Liu, Sen He, et al. OneStory: Coherent multi-shot video generation with adaptive memory. *CVPR*, 2026.

[15] Hyojun Go, Dominik Narnhofer, Goutam Bhat, Prune Truong, Federico Tombari, and Konrad Schindler. Text-to-3D by stitching a multi-view reconstruction network to a video generator. In *ICLR*, 2026.

[16] Hyojun Go, Byeongjun Park, Jiho Jang, Jin-Young Kim, Soonwoo Kwon, and Changick Kim. Splatflow: Multi-view rectified flow model for 3d gaussian splatting synthesis. In *CVPR*, 2025.

[17] Hyojun Go, Byeongjun Park, Hyelin Nam, Byung-Hoon Kim, Hyungjin Chung, and Changick Kim. Videorfsplat: Direct scene-level text-to-3d gaussian splatting generation with flexible pose and multi-view joint modeling. In *ICCV*, 2025.

[18] Dhruba Ghosh, Hannaneh Hajishirzi, and Ludwig Schmidt. Geneval: An object-focused framework for evaluating text-to-image alignment. In *NeurIPS*, 2023.

[19] Zhanhao Liang, Yuhui Yuan, Shuyang Gu, Bohan Chen, Tiankai Hang, Mingxi Cheng, Ji Li, and Liang Zheng. Aesthetic post-training diffusion models from generic preferences with step-by-step preference optimization. In *CVPR*, 2025.

[20] Xiaoshi Wu, Yiming Hao, Keqiang Sun, Yixiong Chen, Feng Zhu, Rui Zhao, and Hongsheng Li. Human preference score v2: A solid benchmark for evaluating human preferences of text-to-image synthesis. *arXiv preprint*, 2023.

[21] Mihir Prabhudesai, Anirudh Goyal, Deepak Pathak, and Katerina Fragkiadaki. Aligning text-to-image diffusion models with reward backpropagation. *arXiv preprint*, 2023.

[22] Kevin Clark, Paul Vicol, Kevin Swersky, and David J. Fleet. Directly fine-tuning diffusion models on differentiable rewards. In *ICLR*, 2024.

[23] Kimin Lee, Hao Liu, Moonkyung Ryu, Olivia Watkins, Yuqing Du, Craig Boutilier, Pieter Abbeel, Mohammad Ghavamzadeh, and Shixiang Shane Gu. Aligning text-to-image models using human feedback. *arXiv preprint*, 2023.

[24] Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, Kashun Shum, and Tong Zhang. RAFT: Reward ranked finetuning for generative foundation model alignment. *Transactions on Machine Learning Research*, 2023.

[25] Bram Wallace, Meihua Dang, Rafael Rafailov, Linqi Zhou, Aaron Lou, Senthil Purushwalkam, Stefano Ermon, Caiming Xiong, Shafiq Joty, and Nikhil Naik. Diffusion model alignment using direct preference optimization. In *CVPR*, 2024.

[26] Kai Yang, Jian Tao, Jiafei Lyu, Chunjiang Ge, Jiaxin Chen, Weihan Shen, Xiaolong Zhu, and Xiu Li. Using human feedback to fine-tune diffusion models without any reward model. In *CVPR*, 2024.

[27] Jie Liu, Gongye Liu, Jiajun Liang, Ziyang Yuan, Xiaokun Liu, Mingwu Zheng, Xiele Wu, Qiulin Wang, Menghan Xia, Xintao Wang, Xiaohong Liu, Fei Yang, Pengfei Wan, Di ZHANG, Kun Gai, Yujiu Yang, and Wanli Ouyang. Improving video generation with human feedback. In *NeurIPS*, 2026.

[28] Kevin Black, Michael Janner, Yilun Du, Ilya Kostrikov, and Sergey Levine. Training diffusion models with reinforcement learning. In *ICLR*, 2024.

[29] Ying Fan, Olivia Watkins, Yuqing Du, Hao Liu, Moonkyung Ryu, Craig Boutilier, Pieter Abbeel, Mohammad Ghavamzadeh, Kangwook Lee, and Kimin Lee. Reinforcement learning for fine-tuning text-to-image diffusion models. In *NeurIPS*, 2023.

[30] Kaiwen Zheng, Huayu Chen, Haotian Ye, Haoxiang Wang, Qinsheng Zhang, Kai Jiang, Hang Su, Stefano Ermon, Jun Zhu, and Ming-Yu Liu. DiffusionNFT: Online diffusion reinforcement with forward process. In *ICLR*, 2026.

[31] Hyungjin Chung, Jeongsol Kim, Michael Thompson Mccann, Marc Louis Klasky, and Jong Chul Ye. Diffusion posterior sampling for general noisy inverse problems. In *ICLR*, 2023.

[32] Jiaming Song, Qinsheng Zhang, Hongxu Yin, Morteza Mardani, Ming-Yu Liu, Jan Kautz, Yongxin Chen, and Arash Vahdat. Loss-guided diffusion models for plug-and-play controllable generation. In *ICML*, 2023.

[33] Haotian Ye, Haowei Lin, Jiaqi Han, Minkai Xu, Sheng Liu, Yitao Liang, Jianzhu Ma, James Zou, and Stefano Ermon. TFG: Unified training-free guidance for diffusion models. In *NeurIPS*, 2024.

[34] Jiwen Yu, Yinhuai Wang, Chen Zhao, Bernard Ghanem, and Jian Zhang. FreeDoM: Training-free energy-guided conditional diffusion model. In *ICCV*, 2023.

[35] Yutong He, Naoki Murata, Chieh-Hsin Lai, Yuhta Takida, Toshimitsu Uesaka, Dongjun Kim, Wei-Hsiang Liao, Yuki Mitsufuji, J Zico Kolter, Ruslan Salakhutdinov, and Stefano Ermon. Manifold preserving guided diffusion. In *ICLR*, 2024.

[36] Jeongsol Kim, Bryan Sangwoo Kim, and Jong Chul Ye. FlowDPS: Flow-driven posterior sampling for inverse problems. In *ICCV*, 2025.

[37] Jiaming Song, Arash Vahdat, Morteza Mardani, and Jan Kautz. Pseudoinverse-guided diffusion models for inverse problems. In *ICLR*, 2023.

[38] Raghav Singhal, Zachary Horvitz, Ryan Teehan, Mengye Ren, Zhou Yu, Kathleen McKeown, and Rajesh Ranganath. A general framework for inference-time scaling and steering of diffusion models. In *ICML*, 2025.

[39] Jaihoon Kim, TaeHoon Yoon, Jisung Hwang, and Minhyuk Sung. Inference-time scaling for flow models via stochastic generation and rollover budget forcing. In *NeurIPS*, 2026.

[40] Xiner Li, Yulai Zhao, Chenyu Wang, Gabriele Scalia, Gokcen Eraslan, Surag Nair, Tommaso Biancalani, Aviv Regev, Sergey Levine, and Masatoshi Uehara. Derivative-free guidance in continuous and discrete diffusion models with soft value-based decoding. *arXiv preprint*, 2024.

[41] Luhuan Wu, Brian L. Trippe, Christian A Naesseth, John Patrick Cunningham, and David Blei. Practical and asymptotically exact conditional sampling in diffusion models. In *NeurIPS*, 2023.

[42] Sunwoo Kim, Minkyu Kim, and Dongmin Park. Test-time alignment of diffusion models without reward over-optimization. In *ICLR*, 2025.

[43] Xiner Li, Masatoshi Uehara, Xingyu Su, Gabriele Scalia, Tommaso Biancalani, Aviv Regev, Sergey Levine, and Shuiwang Ji. Dynamic search for inference-time alignment in diffusion models. *arXiv preprint*, 2025.

[44] XiangCheng Zhang, Haowei Lin, Haotian Ye, James Zou, Jianzhu Ma, Yitao Liang, and Yilun Du. Inference-time scaling of diffusion models through classical search. In *NeurIPS 2025 Workshop on Structured Probabilistic Inference & Generative Modeling*, 2025.

[45] Marta Skreta, Tara Akhound-Sadegh, Viktor Ohanesian, Roberto Bondesan, Alan Aspuru-Guzik, Arnaud Doucet, Rob Brekelmans, Alexander Tong, and Kirill Neklyudov. Feynman-Kac correctors in diffusion: Annealing, guidance, and product of experts. In *ICML*, 2025.

[46] Masatoshi Uehara, Yulai Zhao, Chenyu Wang, Xiner Li, Aviv Regev, Sergey Levine, and Tommaso Biancalani. Inference-time alignment in diffusion models with reward-guided generation: Tutorial and review. *arXiv preprint*, 2025.

[47] Jiazheng Xu, Xiao Liu, Yuchen Wu, Yuxuan Tong, Qinkai Li, Ming Ding, Jie Tang, and Yuxiao Dong. ImageReward: Learning and evaluating human preferences for text-to-image generation. *arXiv preprint*, 2023.

[48] Yibin Wang, Yuhang Zang, Hao Li, Cheng Jin, and Jiaqi Wang. Unified reward model for multimodal understanding and generation. *arXiv preprint*, 2025.

[49] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, 2021.

[50] Yuhang Ma, Xiaoshi Wu, Keqiang Sun, and Hongsheng Li. HPSv3: Towards wide-spectrum human preference score. In *ICCV*, 2025.

[51] Bradley Efron. Tweedie's formula and selection bias. *Journal of the American Statistical Association*, 106 (496):1602–1614, 2011.

[52] Yaxuan Zhu, Zehao Dou, Haoxin Zheng, Yasi Zhang, Ying Nian Wu, and Ruiqi Gao. Think twice before you act: Improving inverse problem solving with MCMC. *arXiv preprint*, 2024.

[53] Fengyuan Dai, Zifeng Zhuang, Yufei Huang, Siteng Huang, Bangyan Liao, Donglin Wang, and Fajie Yuan. VARD: Efficient and dense fine-tuning for diffusion models with value-based RL. *arXiv preprint*, 2025.

[54] Gongye Liu, Bo Yang, Yida Zhi, Zhizhou Zhong, Lei Ke, Didan Deng, Han Gao, Yongxiang Huang, Kaihao Zhang, Hongbo Fu, et al. Beyond VLM-based rewards: Diffusion-native latent reward modeling. *arXiv preprint*, 2026.

[55] Xiaoyue Mi, Wenqing Yu, Jiesong Lian, Shibo Jie, Ruizhe Zhong, Zijun Liu, Guozhen Zhang, Zixiang Zhou, Zhiyong Xu, Yuan Zhou, et al. Video generation models are good latent reward models. *arXiv preprint*, 2025.

[56] Mykola Vysotskyi, Zahar Kohut, Mariia Shpir, Taras Rumezhak, and Volodymyr Karpiv. Critic-guided reinforcement unlearning in text-to-image diffusion. *arXiv preprint*, 2026.

[57] Xiaole Xian, Xilin He, Wenting Chen, Wenshuang Liu, wenqi mu, Yancheng He, Liang Li, Yi Zhang, and Xiangyu Yue. Consistent noisy latent rewards for trajectory preference optimization in diffusion models. In *ICLR*, 2026.

[58] Tao Zhang, Cheng Da, Kun Ding, Huan Yang, kun jin, Yan Li, Tingting Gao, Di Zhang, Shiming Xiang, and Chunhong Pan. Diffusion model as a noise-aware latent reward model for step-level preference optimization. In *NeurIPS*, 2026.

[59] Ziyi Zhang, Sen Zhang, Yibing Zhan, Yong Luo, Yonggang Wen, and Dacheng Tao. Confronting reward overoptimization for diffusion models: A perspective of inductive and primacy biases. In *ICML*, 2024.

[60] Zengqun Zhao, Ziquan Liu, Yu Cao, Shaogang Gong, Zhensong Zhang, Jifei Song, Jiankang Deng, and Ioannis Patras. LatSearch: Latent reward-guided search for faster inference-time scaling in video diffusion. *arXiv preprint*, 2026.

[61] Karel Lenc and Andrea Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *CVPR*, 2015.

[62] Adrián Csiszárik, Péter Kőrösi-Szabó, Akos Matszangosz, Gergely Papp, and Dániel Varga. Similarity and matching of neural network representations. In *NeurIPS*, 2021.

[63] Xingyi Yang, Daquan Zhou, Songhua Liu, Jingwen Ye, and Xinchao Wang. Deep model reassembly. In *NeurIPS*, 2022.

[64] Yamini Bansal, Preetum Nakkiran, and Boaz Barak. Revisiting model stitching to compare neural representations. In *NeurIPS*, 2021.

[65] Zizheng Pan, Jianfei Cai, and Bohan Zhuang. Stitchable neural networks. In *CVPR*, 2023.

[66] Kyungmin Lee, Sihyun Yu, and Jinwoo Shin. Decoupled meanflow: Turning flow models into flow maps for accelerated sampling. *arXiv preprint*, 2025.

[67] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *ICML*, 2024.

[68] Stability AI. Stable Diffusion 3.5. https://github.com/Stability-AI/sd3.5, 2024. Official inference repository for Stable Diffusion 3.5 Large, Large Turbo, and Medium. Accessed: 2026-04-27.

[69] Black Forest Labs. FLUX.1 [dev]. https://github.com/black-forest-labs/flux, 2024. Official inference repository for FLUX.1 open-weight models. Accessed: 2026-04-27.

[70] Alex Fang, Albin Madappally Jose, Amit Jain, Ludwig Schmidt, Alexander T Toshev, and Vaishaal Shankar. Data filtering networks. In *ICLR*, 2024.

[71] Christoph Schuhmann. CLIP+MLP aesthetic score predictor. https://github.com/christophschuhmann/improved-aesthetic-predictor, 2022. Train, use, and visualize an aesthetic score predictor based on a neural network taking CLIP embeddings as input. Accessed: 2026-04-27.

[72] Arpit Bansal, Hong-Min Chu, Avi Schwarzschild, Soumyadip Sengupta, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Universal guidance for diffusion models. In *CVPR*, 2023.

[73] Xu Han, Caihua Shan, Yifei Shen, Can Xu, Han Yang, Xiang Li, and Dongsheng Li. Training-free multi-objective diffusion model for 3d molecule generation. In *ICLR*, 2024.

[74] Xiaoshi Wu, Yiming Hao, Manyuan Zhang, Keqiang Sun, Zhaoyang Huang, Guanglu Song, Yu Liu, and Hongsheng Li. Deep reward supervisions for tuning text-to-image diffusion models. In *ECCV*, 2024.

[75] Zichen Miao, Jiang Wang, Ze Wang, Zhengyuan Yang, Lijuan Wang, Qiang Qiu, and Zicheng Liu. Training diffusion models towards diverse image generation with reinforcement learning. In *CVPR*, 2024.

[76] Jie Liu, Gongye Liu, Jiajun Liang, Yangguang Li, Jiaheng Liu, Xintao Wang, Pengfei Wan, Di ZHANG, and Wanli Ouyang. Flow-GRPO: Training flow matching models via online RL. In *NeurIPS*, 2026.

[77] Zeyue Xue, Jie Wu, Yu Gao, Fangyuan Kong, Lingting Zhu, Mengzhao Chen, Zhiheng Liu, Wei Liu, Qiushan Guo, Weilin Huang, et al. DanceGRPO: Unleashing GRPO on visual generation. *arXiv preprint*, 2025.

[78] Davide Bucciarelli, Evelyn Turri, Lorenzo Baraldi, Marcella Cornia, and Rita Cucchiara. Tiny inference-time scaling with latent verifiers. *arXiv preprint*, 2026.

[79] Vasco Ramos, Regev Cohen, Idan Szpektor, and Joao Magalhaes. Beyond the Noise: Aligning prompts with latent representations in diffusion models. *arXiv preprint*, 2025.

[80] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *ICML*, 2019.

[81] Zhaochong An, Orest Kupyn, Théo Uscidda, Andrea Colaco, Karan Ahuja, Serge Belongie, Mar Gonzalez-Franco, and Marta Tintore Gazulla. VGGRPO: Towards world-consistent video generation with 4d latent reward. *arXiv preprint*, 2026.

[82] Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. VGGT: Visual geometry grounded transformer. In *CVPR*, 2025.

[83] Lihan Jiang, Yucheng Mao, Linning Xu, Tao Lu, Kerui Ren, Yichen Jin, Xudong Xu, Mulin Yu, Jiangmiao Pang, Feng Zhao, et al. AnySplat: Feed-forward 3d gaussian splatting from unconstrained views. *ACM Transactions on Graphics*, 44(6):1–16, 2025.

[84] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.

[85] Peter Holderrieth, Uriel Singer, Tommi Jaakkola, Ricky TQ Chen, Yaron Lipman, and Brian Karrer. GLASS Flows: Transition sampling for alignment of flow and diffusion models. *arXiv preprint*, 2025.

[86] Byeongjun Park, Hyojun Go, Hyelin Nam, Byung-Hoon Kim, Hyungjin Chung, and Changick Kim. Steerx: Creating any camera-free 3d and 4d scenes with geometric steering. In *ICCV*, 2025.

[87] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat GANs on image synthesis. In *NeurIPS*, 2021.

[88] Naila Murray, Luca Marchesotti, and Florent Perronnin. AVA: A large-scale database for aesthetic visual analysis. In *CVPR*, 2012.

[89] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014.

[90] Peter Young, Alice Lai, Micah Hodosh, and Julia Hockenmaier. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *Transactions of the Association for Computational Linguistics*, 2:67–78, 2014.

[91] Simon Hentschel, Konstantin Kobs, and Andreas Hotho. CLIP knows image aesthetics. *Frontiers in Artificial Intelligence*, 5:976235, 2022.

[92] Christoph Schuhmann, Richard Vencu, Romain Beaumont, Robert Kaczmarczyk, Clayton Mullis, Aarush Katta, Theo Coombes, Jenia Jitsev, and Aran Komatsuzaki. LAION-400M: Open dataset of CLIP-filtered 400 million image-text pairs. *arXiv preprint*, 2021.

[93] Yuval Kirstain, Adam Polyak, Uriel Singer, Shahbuland Matiana, Joe Penna, and Omer Levy. Pick-a-Pic: An open dataset of user preferences for text-to-image generation. In *NeurIPS*, 2023.

[94] Zheng Ding and Weirui Ye. TreeGRPO: Tree-advantage GRPO for online RL post-training of diffusion models. In *ICLR*, 2026.

[95] Yuming Li, Yikai Wang, Yuying zhu, Zhongyu Zhao, Ming Lu, Qi She, and Shanghang Zhang. Branch-GRPO: Stable and efficient GRPO with structured branching in diffusion models. In *ICLR*, 2026.

[96] Junzhe Li, Yutao Cui, Tao Huang, Yinping Ma, Chun Fan, Yiming Cheng, Miles Yang, Zhao Zhong, and Liefeng Bo. MixGRPO: Unlocking flow-based GRPO efficiency with mixed ODE-SDE. *arXiv preprint*, 2025.

[97] Jing Wang, Jiajun Liang, Jie Liu, Henglin Liu, Gongye Liu, Jun Zheng, Wanyuan Pang, Ao Ma, Zhenyu Xie, Xintao Wang, et al. GRPO-Guard: Mitigating implicit over-optimization in flow matching via regulated clipping. *arXiv preprint*, 2025.

[98] Ali Taghibakhshi, Sahil Jain, Gerald Shen, Nima Tajbakhsh, and Arash Vahdat. Enhance text-to-image fine-tuning with DRaFT+, now part of NVIDIA NeMo. https://developer.nvidia.com/blog/enhance-text-to-image-fine-tuning-with-draft-now-part-of-nvidia-nemo/, 2024. NVIDIA Technical Blog. Accessed: 2026-04-29.

[99] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

[100] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint*, 2022.

[101] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *ICLR*, 2022.

[102] Hyojun Go, Yunsung Lee, Jin-Young Kim, Seunghyun Lee, Myeongho Jeong, Hyun Seung Lee, and Seungtaek Choi. Towards practical plug-and-play diffusion models. In *CVPR*, 2023.

[103] Byeongjun Park, Hyojun Go, Jin-Young Kim, Sangmin Woo, Seokil Ham, and Changick Kim. Switch diffusion transformer: Synergizing denoising tasks with sparse mixture-of-experts. In *ECCV*, 2024.

[104] Yunsung Lee, JinYoung Kim, Hyojun Go, Myeongho Jeong, Shinhyeok Oh, and Seungtaek Choi. Multi-architecture multi-expert diffusion models. In *AAAI*, volume 38, 2024.

[105] Byeongjun Park, Sangmin Woo, Hyojun Go, Jin-Young Kim, and Changick Kim. Denoising task routing for diffusion models. In *ICLR*, 2024.

[106] Seokil Ham, Sangmin Woo, Jin-Young Kim, Hyojun Go, Byeongjun Park, and Changick Kim. Diffusion model patching via mixture-of-prompts. In *AAAI*, volume 39, 2025.

A. Proofs

We start by reviewing the sampling process of the flow-based models. A standard approach is to resort to ODE sampling:

$$\mathbf{z}_1 \sim p_1, \quad \frac{d}{dt}\mathbf{z}_t = u_t(\mathbf{z}_t), \quad t : 1 \to 0 \Rightarrow \mathbf{z}_t \sim p_t. \tag{10}$$

One can also resort to SDE sampling [3]:

$$\mathbf{z}_1 \sim p_1, \quad d\mathbf{z}_t = \left[u_t(\mathbf{z}_t) - \frac{\nu_t^2}{2}\nabla_{\mathbf{z}_t}\log p_t(\mathbf{z}_t)\right]dt + \nu_t\, dW_t, \quad t : 1 \to 0, \tag{11}$$

where dW_t is the standard Wiener process, and $\nabla_{\mathbf{z}_t}\log p_t(\mathbf{z}_t)$ is the score function, which can be obtained by simply reparametrizing u_t. The diffusion coefficient ν_t is a free parameter controlling the amount of injected noise; for FM with $\alpha_t = 1 - t, \sigma_t = t$, a natural choice that preserves the marginals of the probability flow ODE is $\nu_t^2 = \frac{4t}{1-t}$, giving drift coefficient $\frac{2t}{1-t}$ on the score. See Section A.1 for the full derivation. In discrete time, for schedules $1 = t_K > t_{K-1} > \cdots > t_0 = 0$, there exist other ways to sample from the data distribution with discrete transition kernels [1, 85], i.e.

$$p_0(\mathbf{z}_0) = \int p_1(\mathbf{z}_{t_K})\left[\prod_{k=1}^{K} p_{t_{k-1}|t_k}(\mathbf{z}_{t_{k-1}}|\mathbf{z}_{t_k})\right] d\mathbf{z}_{t_{1:K}}, \tag{12}$$

where $p_{t_{k-1}|t_k}(\mathbf{z}_{t_{k-1}}|\mathbf{z}_{t_k})$ is the discrete transition kernel.

A.1. Reparametrization in Diffusion and Flow-based Models

Here, we provide a brief clarification on why the reparametrization of the velocity field in FMs can retrieve the score function $\nabla_{\mathbf{z}_t}\log p_t(\mathbf{z}_t)$ and the posterior mean $\mathbb{E}[\mathbf{z}_0|\mathbf{z}_t]$. For general α_t and σ_t, the conditional vector field is given as [5]

$$u_t(\mathbf{z}_t|\mathbf{z}_0) = \frac{\dot\sigma_t}{\sigma_t}\mathbf{z}_t + \left(\dot\alpha_t - \alpha_t\frac{\dot\sigma_t}{\sigma_t}\right)\mathbf{z}_0. \tag{13}$$

Denoiser as velocity field reparametrization. We have that

$$\begin{aligned} u_t(\mathbf{z}_t) &= \int u_t(\mathbf{z}_t|\mathbf{z}_0)\, p_{0|t}(\mathbf{z}_0|\mathbf{z}_t)\, d\mathbf{z}_0 && (14)\\ &= \frac{\dot\sigma_t}{\sigma_t}\mathbf{z}_t + \left(\dot\alpha_t - \alpha_t\frac{\dot\sigma_t}{\sigma_t}\right)\int \mathbf{z}_0\, p_{0|t}(\mathbf{z}_0|\mathbf{z}_t)\, d\mathbf{z}_0 && (15)\\ &= \frac{\dot\sigma_t}{\sigma_t}\mathbf{z}_t + \left(\dot\alpha_t - \alpha_t\frac{\dot\sigma_t}{\sigma_t}\right) D_t(\mathbf{z}_t), && (16)\end{aligned}$$

where we denoted the posterior mean as the *denoiser* $D_t(\mathbf{z}_t) := \mathbb{E}[\mathbf{z}_0|\mathbf{z}_t]$. Rearranging,

$$D_t(\mathbf{z}_t) = \frac{1}{\dot\alpha_t\sigma_t - \alpha_t\dot\sigma_t}\left(\sigma_t u_t(\mathbf{z}_t) - \dot\sigma_t\mathbf{z}_t\right). \tag{17}$$

Score function as velocity field reparametrization. Tweedie's formula establishes the relation between the posterior mean and the score function

$$D_t(\mathbf{z}_t) = \mathbb{E}[\mathbf{z}_0|\mathbf{z}_t] = \frac{1}{\alpha_t}\left(\mathbf{z}_t + \sigma_t^2\nabla_{\mathbf{z}_t}\log p_t(\mathbf{z}_t)\right). \tag{18}$$

Plugging Eq. (18) into Eq. (17) and rearranging yields

$$u_t(\mathbf{z}_t) = \frac{\dot{\alpha}_t}{\alpha_t}\mathbf{z}_t + \frac{\tilde{\nu}_t^2}{2}\nabla_{\mathbf{z}_t}\log p_t(\mathbf{z}_t), \tag{19}$$

where we defined

$$\frac{\tilde{\nu}_t^2}{2} := \frac{\sigma_t(\dot{\alpha}_t\sigma_t - \alpha_t\dot{\sigma}_t)}{\alpha_t} = \sigma_t^2\frac{d}{dt}\log\frac{\alpha_t}{\sigma_t}. \tag{20}$$

We use $\tilde{\nu}_t$ (rather than ν_t) here to distinguish this velocity–score coupling from the SDE diffusion coefficient ν_t in the main-text sampling SDE (11). These are different quantities: for the FM schedule below, $\tilde{\nu}_t^2 < 0$ while $\nu_t^2 > 0$ is a free design parameter of the sampler.

Specialization to the FM schedule. We now specialize the above to the FM schedule used in the main text, $\alpha_t = 1 - t$ and $\sigma_t = t$, for which $\dot{\alpha}_t = -1$, $\dot{\sigma}_t = 1$, and

$$\dot{\alpha}_t\sigma_t - \alpha_t\dot{\sigma}_t = -t - (1 - t) = -1. \tag{21}$$

The denoiser (17) simplifies to

$$D_t(\mathbf{z}_t) = \mathbf{z}_t - t\,u_t(\mathbf{z}_t), \tag{22}$$

and the velocity–score coupling (20) becomes

$$\frac{\tilde{\nu}_t^2}{2} = -\frac{t}{1-t}, \tag{23}$$

so (19) reads

$$u_t(\mathbf{z}_t) = -\frac{1}{1-t}\mathbf{z}_t - \frac{t}{1-t}\nabla_{\mathbf{z}_t}\log p_t(\mathbf{z}_t). \tag{24}$$

Equivalently, the score can be recovered from u_t and $\mathbf{z}_t$ as

$$\nabla_{\mathbf{z}_t}\log p_t(\mathbf{z}_t) = -\frac{(1-t)\,u_t(\mathbf{z}_t) + \mathbf{z}_t}{t}. \tag{25}$$

A.2. From score reparametrization to gradient guidance

Step 1: Sampling SDE in terms of the score. The sampling SDE (11) in the main text,

$$d\mathbf{z}_t = \left[u_t(\mathbf{z}_t) - \frac{\nu_t^2}{2}\nabla_{\mathbf{z}_t}\log p_t(\mathbf{z}_t)\right]dt + \nu_t\,dW_t, \quad t : 1 \to 0, \tag{26}$$

is written in terms of the velocity u_t. Substituting the velocity–score reparametrization (19) eliminates u_t and gives a form in which the drift depends only on $\mathbf{z}_t$ and $\nabla_{\mathbf{z}_t}\log p_t(\mathbf{z}_t)$:

$$d\mathbf{z}_t = \left[\frac{\dot{\alpha}_t}{\alpha_t}\mathbf{z}_t + \frac{\tilde{\nu}_t^2}{2}\nabla_{\mathbf{z}_t}\log p_t(\mathbf{z}_t) - \frac{\nu_t^2}{2}\nabla_{\mathbf{z}_t}\log p_t(\mathbf{z}_t)\right]dt + \nu_t\,dW_t \tag{27}$$

$$= \left[\frac{\dot{\alpha}_t}{\alpha_t}\mathbf{z}_t + \frac{\tilde{\nu}_t^2 - \nu_t^2}{2}\nabla_{\mathbf{z}_t}\log p_t(\mathbf{z}_t)\right]dt + \nu_t\,dW_t. \tag{28}$$

For the FM schedule with the main-text choice $\nu_t^2 = 4t/(1-t)$, combined with (23),

$$\frac{\tilde{\nu}_t^2 - \nu_t^2}{2} = -\frac{t}{1-t} - \frac{2t}{1-t} = -\frac{3t}{1-t}, \tag{29}$$

so (28) becomes

$$d\mathbf{z}_t = \left[-\frac{1}{1-t}\mathbf{z}_t - \frac{3t}{1-t}\nabla_{\mathbf{z}_t}\log p_t(\mathbf{z}_t)\right]dt + \nu_t\,dW_t. \tag{30}$$

Step 2: Score of the reward-tilted marginal. To sample from the tilted target (2), we need the score of the tilted marginal $p_t^\star$ at every intermediate t. Marginalizing the same forward kernel $p_t(\mathbf{z}_t|\mathbf{z}_0)$ against $p_0^\star$,

$$p_t^\star(\mathbf{z}_t) = \int p_t(\mathbf{z}_t|\mathbf{z}_0)\, p_0^\star(\mathbf{z}_0)\, d\mathbf{z}_0 \tag{31}$$

$$= \frac{1}{Z_\mathbf{z}} \int p_t(\mathbf{z}_t|\mathbf{z}_0)\, p_0(\mathbf{z}_0) \exp(r(\mathbf{z}_0))\, d\mathbf{z}_0 \tag{32}$$

$$= \frac{p_t(\mathbf{z}_t)}{Z_\mathbf{z}} \int p_{0|t}(\mathbf{z}_0|\mathbf{z}_t) \exp(r(\mathbf{z}_0))\, d\mathbf{z}_0 \tag{33}$$

$$= \frac{p_t(\mathbf{z}_t)}{Z_\mathbf{z}} \exp(V_t(\mathbf{z}_t)), \tag{34}$$

where the last line uses the definition of the value function (3). Taking the logarithm and then the gradient in $\mathbf{z}_t$, the $\mathbf{z}_t$-independent factor $Z_\mathbf{z}$ drops out, leaving

$$\boxed{\nabla_{\mathbf{z}_t} \log p_t^\star(\mathbf{z}_t) = \nabla_{\mathbf{z}_t} \log p_t(\mathbf{z}_t) + \nabla_{\mathbf{z}_t} V_t(\mathbf{z}_t)} \tag{35}$$

The score of the tilted marginal is the pretrained score shifted by the gradient of the value function.

Step 3: Gradient guidance as sampling from $p^\star$. Sampling from $p^\star$ uses the same SDE (28) but with the tilted score $\nabla \log p_t^\star$ in place of $\nabla \log p_t$. Substituting (35),

$$d\mathbf{z}_t = \left[\frac{\dot\alpha_t}{\alpha_t}\mathbf{z}_t + \frac{\tilde\nu_t^2 - \nu_t^2}{2}\nabla_{\mathbf{z}_t} \log p_t^\star(\mathbf{z}_t)\right] dt + \nu_t\, dW_t \tag{36}$$

$$= \left[\frac{\dot\alpha_t}{\alpha_t}\mathbf{z}_t + \frac{\tilde\nu_t^2 - \nu_t^2}{2}\left(\nabla_{\mathbf{z}_t} \log p_t(\mathbf{z}_t) + \nabla_{\mathbf{z}_t} V_t(\mathbf{z}_t)\right)\right] dt + \nu_t\, dW_t \tag{37}$$

$$= \underbrace{\left[\frac{\dot\alpha_t}{\alpha_t}\mathbf{z}_t + \frac{\tilde\nu_t^2 - \nu_t^2}{2}\nabla_{\mathbf{z}_t} \log p_t(\mathbf{z}_t)\right] dt + \nu_t\, dW_t}_{\text{pretrained sampling SDE (28)}} + \underbrace{\frac{\tilde\nu_t^2 - \nu_t^2}{2}\nabla_{\mathbf{z}_t} V_t(\mathbf{z}_t)\, dt}_{\text{gradient guidance correction}}. \tag{38}$$

Translating the score-only form back to the velocity form by inverting the substitution in Step 1 (i.e. using $\frac{\dot\alpha_t}{\alpha_t}\mathbf{z}_t = u_t(\mathbf{z}_t) - \frac{\tilde\nu_t^2}{2}\nabla_{\mathbf{z}_t} \log p_t$), we obtain the equivalent expression

$$d\mathbf{z}_t = \left[u_t(\mathbf{z}_t) - \frac{\nu_t^2}{2}\nabla_{\mathbf{z}_t} \log p_t(\mathbf{z}_t) + \frac{\tilde\nu_t^2 - \nu_t^2}{2}\nabla_{\mathbf{z}_t} V_t(\mathbf{z}_t)\right] dt + \nu_t\, dW_t. \tag{39}$$

That is, sampling from the reward-tilted distribution $p^\star$ reduces to running the pretrained sampling SDE (26) with a single additional drift term proportional to $\nabla_{\mathbf{z}_t} V_t(\mathbf{z}_t)$. This is exactly *gradient guidance*: following the gradient of the log value function at each step steers the trajectory toward high-reward regions, and by (35) it does so in precisely the way required to sample from $p^\star$. Specializing to FM with $\nu_t^2 = 4t/(1-t)$, the guidance coefficient becomes $\frac{\tilde\nu_t^2 - \nu_t^2}{2} = -\frac{3t}{1-t}$, so (39) reads

$$d\mathbf{z}_t = \left[u_t(\mathbf{z}_t) - \frac{2t}{1-t}\nabla_{\mathbf{z}_t} \log p_t(\mathbf{z}_t) - \frac{3t}{1-t}\nabla_{\mathbf{z}_t} V_t(\mathbf{z}_t)\right] dt + \nu_t\, dW_t. \tag{40}$$

A.3. Reinforcement Post-training of Diffusion

Setup. The discrete denoising transition of diffusion sampling can be formulated as a Markov Decision Process (MDP). Let the state at step k be $\mathbf{z}_{t_k}$, the action is the denoised estimate $\mathbf{a}_{t_k} := \mathbf{z}_{t_{k-1}}$

predicted by the neural network, and the policy is defined as $\pi(\mathbf{a}_{t_k}|\mathbf{s}_{t_k}) := p_\theta(\mathbf{z}_{t_{k-1}}|\mathbf{z}_{t_k})$. The transition is deterministic, and the initial state distribution is the reference distribution. The reward is defined at the final step, i.e. $t = 0$.

KL-regularized RL induces reward-tilted distribution. We show that the KL-regularized RL objective (5) has the reward-tilted distribution (2) as its unique optimum. Expanding the KL term,

$$\begin{aligned}
\mathbb{E}_{\mathbf{z}_0\sim p_\theta}[r(\mathbf{z}_0)] - D_{\mathrm{KL}}(p_\theta \,\|\, p) &= \int p_\theta(\mathbf{z}_0)\,[r(\mathbf{z}_0) + \log p(\mathbf{z}_0) - \log p_\theta(\mathbf{z}_0)]\, d\mathbf{z}_0 && (41)\\
&= \int p_\theta(\mathbf{z}_0) \log \frac{p(\mathbf{z}_0)\exp(r(\mathbf{z}_0))}{p_\theta(\mathbf{z}_0)}\, d\mathbf{z}_0 && (42)\\
&= \int p_\theta(\mathbf{z}_0) \log \frac{Z_{\mathbf{z}}\, p^\star(\mathbf{z}_0)}{p_\theta(\mathbf{z}_0)}\, d\mathbf{z}_0 && (43)\\
&= \log Z_{\mathbf{z}} - D_{\mathrm{KL}}(p_\theta \,\|\, p^\star), && (44)
\end{aligned}$$

where the third line uses the definition $p^\star(\mathbf{z}_0) = \frac{1}{Z_{\mathbf{z}}} p(\mathbf{z}_0)\exp(r(\mathbf{z}_0))$ from (2). Since $\log Z_{\mathbf{z}}$ is independent of θ and $D_{\mathrm{KL}}(p_\theta \,\|\, p^\star) \geq 0$ with equality iff $p_\theta = p^\star$, the objective is maximized precisely when $p_\theta = p^\star$. Hence optimizing (5) is equivalent to minimizing the reverse KL to the reward-tilted target $p^\star$, and the two formulations share the same global optimum.

A.4. Off-policy Value Model Training

Proposition 1. *Consider the population objective*

$$\mathcal{L}_{\mathrm{value}}(\omega) = \mathbb{E}_t\, \mathbb{E}_{\mathbf{z}_0\sim p_0}\, \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(0,I_d)} \left[\left(V_\omega^{(i^\star,j^\star)}(\mathbf{z}_t) - r_\phi(\mathbf{z}_0) \right)^2 \right], \tag{45}$$

with $\mathbf{z}_t = \alpha_t \mathbf{z}_0 + \sigma_t \boldsymbol{\epsilon}$. Assume the parametric family $\{V_\omega^{(i^\star,j^\star)} : \omega\}$ is expressive enough to represent any function of $(t, \mathbf{z}_t)$. Then the minimizer satisfies

$$V_{\omega^\star}^{(i^\star,j^\star)}(\mathbf{z}_t) \;=\; \mathbb{E}\left[r_\phi(\mathbf{z}_0) \mid \mathbf{z}_t \right], \tag{46}$$

where the conditional expectation is taken under the posterior $p_{0|t}(\mathbf{z}_0|\mathbf{z}_t) \propto p_t(\mathbf{z}_t|\mathbf{z}_0)\, p_0(\mathbf{z}_0)$.

Proof. Rewrite the objective by first conditioning on $(t, \mathbf{z}_t)$:

$$\mathcal{L}_{\mathrm{value}}(\omega) = \mathbb{E}_t\, \mathbb{E}_{\mathbf{z}_t\sim p_t}\, \mathbb{E}_{\mathbf{z}_0\sim p_{0|t}(\cdot|\mathbf{z}_t)} \left[\left(V_\omega^{(i^\star,j^\star)}(\mathbf{z}_t) - r_\phi(\mathbf{z}_0) \right)^2 \right]. \tag{47}$$

For each fixed $(t, \mathbf{z}_t)$, the model output $V_\omega^{(i^\star,j^\star)}(\mathbf{z}_t)$ is a single scalar, while $r_\phi(\mathbf{z}_0)$ varies according to the posterior $p_{0|t}(\cdot|\mathbf{z}_t)$. The inner expectation is therefore a one-dimensional quadratic in this scalar, which is uniquely minimized by the posterior mean:

$$\arg\min_{c\in\mathbb{R}} \mathbb{E}_{\mathbf{z}_0\sim p_{0|t}(\cdot|\mathbf{z}_t)} \left[(c - r_\phi(\mathbf{z}_0))^2 \right] \;=\; \mathbb{E}[r_\phi(\mathbf{z}_0) \mid \mathbf{z}_t]. \tag{48}$$

Since this minimum is attained at every $(t, \mathbf{z}_t)$ by the same function $\mathbf{z}_t \mapsto \mathbb{E}[r_\phi(\mathbf{z}_0) \mid \mathbf{z}_t]$, and the outer expectation is a non-negative weighted average of the inner minima, it is minimized by the same function. The expressiveness assumption ensures this function lies in the parametric family. □

We refer to the *standard* value function as

$$\tilde{V}_t(\mathbf{z}_t) \;:=\; \mathbb{E}[r_\phi(\mathbf{z}_0) \mid \mathbf{z}_t], \tag{49}$$

to distinguish it from the *soft* value function

$$V_t(\mathbf{z}_t) \;:=\; \log \mathbb{E}[\exp(r_\phi(\mathbf{z}_0)) \mid \mathbf{z}_t] \tag{50}$$

defined in (3).

Relation to the soft value function. Applying Jensen's inequality to exp,

$$\tilde{V}_t(\mathbf{z}_t) \;=\; \mathbb{E}[r_\phi(\mathbf{z}_0) \mid \mathbf{z}_t] \;\le\; \log \mathbb{E}[\exp(r_\phi(\mathbf{z}_0)) \mid \mathbf{z}_t] \;=\; V_t(\mathbf{z}_t), \tag{51}$$

with equality if and only if $r_\phi(\mathbf{z}_0)$ is constant on the support of the posterior $p_{0|t}(\cdot|\mathbf{z}_t)$. The two thus coincide in the noiseless limit $t \to 0$, where the posterior concentrates on the corresponding clean latent.

To make the connection precise at finite t, consider the tempered reward $r \mapsto \lambda r$ with inverse temperature $\lambda > 0$, and write

$$V^{(\lambda)}(\mathbf{z}_t) := \log \mathbb{E}\big[\exp(\lambda r_\phi(\mathbf{z}_0)) \mid \mathbf{z}_t\big], \qquad \tilde{V}^{(\lambda)}(\mathbf{z}_t) := \mathbb{E}[\lambda r_\phi(\mathbf{z}_0) \mid \mathbf{z}_t] = \lambda \tilde{V}_t(\mathbf{z}_t). \tag{52}$$

Let $\mu_t := \mathbb{E}[r_\phi(\mathbf{z}_0) \mid \mathbf{z}_t] = \tilde{V}_t(\mathbf{z}_t)$ and $\sigma_t^2 := \mathrm{Var}[r_\phi(\mathbf{z}_0) \mid \mathbf{z}_t]$. Expanding the exponential and the logarithm in powers of λ around $\lambda = 0$,

$$\mathbb{E}[\exp(\lambda r_\phi(\mathbf{z}_0)) \mid \mathbf{z}_t] = 1 + \lambda \mu_t + \frac{\lambda^2}{2}(\mu_t^2 + \sigma_t^2) + O(\lambda^3), \tag{53}$$

and applying $\log(1+x) = x - x^2/2 + O(x^3)$ yields

$$V^{(\lambda)}(\mathbf{z}_t) \;=\; \lambda \tilde{V}_t(\mathbf{z}_t) \;+\; \frac{\lambda^2}{2} \mathrm{Var}[r_\phi(\mathbf{z}_0) \mid \mathbf{z}_t] \;+\; O(\lambda^3), \tag{54}$$

and consequently

$$\nabla_{\mathbf{z}_t} V^{(\lambda)}(\mathbf{z}_t) \;=\; \lambda \, \nabla_{\mathbf{z}_t} \tilde{V}_t(\mathbf{z}_t) \;+\; \frac{\lambda^2}{2} \nabla_{\mathbf{z}_t} \mathrm{Var}[r_\phi(\mathbf{z}_0) \mid \mathbf{z}_t] \;+\; O(\lambda^3). \tag{55}$$

Two consequences make $\tilde{V}$ a faithful surrogate for V in the alignment methods of Section 3.2.

Gradient guidance. Substituting (55) into the gradient-guidance correction (39), the leading-order tilt is

$$\frac{\tilde{\nu}_t^2 - \nu_t^2}{2} \nabla_{\mathbf{z}_t} V^{(\lambda)}(\mathbf{z}_t) \;=\; \lambda \cdot \frac{\tilde{\nu}_t^2 - \nu_t^2}{2} \nabla_{\mathbf{z}_t} \tilde{V}_t(\mathbf{z}_t) \;+\; O(\lambda^2), \tag{56}$$

which is exactly the guidance one obtains by replacing V with $\tilde{V}$ and absorbing the factor λ into the guidance scale c_t in (4). To leading order in the reward scale, gradient guidance with the regressed conditional-mean value $\tilde{V}$ thus samples from the reward-tilted distribution at temperature λ, with the temperature implicit in the chosen guidance coefficient.

Particle methods. FK steering and other SMC-style methods use V only through pairwise comparisons that determine particle weights (Eq. (9)). By (54), $V^{(\lambda)}$ and $\lambda\tilde{V}$ differ by $\frac{\lambda^2}{2}\sigma_t^2 + O(\lambda^3)$, so they induce the same particle ordering up to $O(\lambda^2)$ corrections controlled by the conditional reward variance σ_t^2. The correction is largest in the high-noise regime (where σ_t^2 is largest), but $\tilde{V}$ retains the dominant ranking signal in our experiments.

B. Extended Related Work

Here, we expand the related work in Section 2 with a detailed discussion of (i) inference-time approximations of the value function, (ii) training-time alignment with Monte Carlo approximation, and (iii) the role of value models and noisy latent reward models in existing alignment pipelines.

Inference-time approximations. Inference-time alignment can be viewed as an approximation to a soft optimal denoising policy, in which the value function provides a look-ahead reward prediction [46]. Since rewards are typically defined in clean pixel space, the value function cannot be evaluated directly on noisy latents. The *Tweedie approximation* [31, 32, 51] forms a one-step clean estimate via the Tweedie formula, decodes it through the VAE, and evaluates the pixel-space reward on the result. Most guidance methods [31–37, 72, 73] use the gradient of this approximation to modify the denoising velocity or score, while sequential Monte Carlo (SMC) methods [38, 39, 41, 42] use it to compute particle importance weights and resampling probabilities. This strategy incurs two costs: estimator bias in high-noise regions and an additional denoiser pass per evaluation. The *Monte Carlo approximation* [40, 46] instead estimates the value function by rolling out multiple denoising trajectories from a noisy latent and aggregating the resulting rewards, as in SVDD [40] and search-based methods such as DSearch [43]. This avoids approximation bias but incurs prohibitive trajectory-rollout cost, which compounds with single-trajectory variance in budget-limited regimes.

Training-time alignment with Monte Carlo approximation. RL-based post-training methods [21–30] similarly require value function estimation along the denoising trajectory, which is commonly approximated by Monte Carlo rollouts. Direct reward finetuning [21, 22, 74] backpropagates terminal rewards through full denoising trajectories; PPO-family methods [28, 29, 75–77, 94–97] optimize policy gradients over sampled trajectories; and DiffusionNFT [30] uses terminal rewards from complete generations within a forward-process contrastive objective. These methods inherit the cost of rollout-based training, suffer from high-variance terminal-reward estimates when only one or a few trajectories are used [56], and provide only weak credit assignment to intermediate denoising steps [59].

Value models and noisy latent reward models in alignment pipelines. Several works leverage value models, or more broadly noisy latent reward models, to address the limitations above. LatSearch [60] evaluates intermediate noisy latents instead of fully rolling out every candidate trajectory, thereby reducing the cost of search-based inference. In PPO-style post-training, such models have been used to provide per-step feedback that improves credit assignment over intermediate denoising steps [59] and stabilizes high-variance policy-gradient updates [56]. DPO-style methods [19, 57, 58] likewise utilize noisy latent rewards or preferences to refine credit assignment over intermediate denoising states, while direct reward finetuning with a value model [55] provides feedback before full denoising completion, reducing rollout costs.

C. Additional Methodological Details

C.1. StitchVM Training

This subsection provides details that were left implicit in Section 4.1. We describe the precise form of the stitching layer s_ψ, which extends the closed-form linear component in Eq. (7), and the practical restrictions we impose on the stitching-layer search grid.

Stitching layer architecture. The closed-form solution $W^{\star}_{i^{\star},j^{\star}}$ gives the best *linear* map between the two pretrained representation spaces. To additionally model the nonlinear discrepancy while reusing this least-squares-optimal initialization, we parameterize s_ψ as a residual block built on top of $W^{\star}_{i^{\star},j^{\star}}$:

$$s_\psi(\mathbf{h}) \;=\; \mathrm{Up}\big(F_\psi(\mathbf{h})\big) \;+\; G_\psi\Big(\mathrm{Up}\big(F_\psi(\mathbf{h})\big)\Big), \tag{57}$$

where $\mathbf{h}$ denotes the diffusion-backbone feature at layer $i^\star$, F_ψ is a 1×1 convolution initialized with $W^{\star}_{i^{\star},j^{\star}}$, $\mathrm{Up}(\cdot)$ is a deterministic bilinear resampling operator that matches the spatial resolution of the target reward model representation, and

$$G_\psi = \mathrm{Conv}_{1\times1} \circ \mathrm{SiLU} \circ \mathrm{Conv}_{1\times1} \circ \mathrm{SiLU} \tag{58}$$

is a two-layer pointwise MLP with bottleneck ratio 1:r ($r = 8$ in all stitching configurations). The weights and bias of the final 1×1 convolution in G_ψ are zero-initialized, so that $G_\psi(\cdot) \equiv \mathbf{0}$ at the beginning of training.

Channel and resolution mismatch. The DiT token grid and the reward model patch grid generally have different spatial resolutions, since the two backbones were pretrained with different patch sizes and input resolutions. We bridge this resolution mismatch with a single bilinear resampling step applied after F_ψ, denoted by $\mathrm{Up}(\cdot)$ in the equation above. The channel mismatch between the diffusion backbone and the reward model is handled entirely by F_ψ.

Tokenization. Since the reward model is truncated at layer $j^\star$, the stitched feature must be converted into the token format expected by the remaining reward model suffix $r_\phi^{\geq j}$. We perform this token-format adaptation following the reward model's original tokenization scheme. Specifically, the spatial output of s_ψ is flattened into patch tokens, augmented with the special tokens required by the reward model suffix, and combined with the corresponding positional embeddings before being passed to $r_\phi^{\geq j}$. For CLIP-based reward models, this amounts to prepending the `[CLS]` token and adding the positional embeddings.

Diffusion-backbone conditioning. When the diffusion backbone requires conditioning inputs such as the timestep or text embeddings, we provide the noise-level conditioning corresponding to the forward process and use a fixed null-text conditioning during training.

Stitching layer selection. We rank candidate layer pairs (i, j) according to the closed-form feature-matching loss. We describe below the probe set used to estimate this loss and the practical restrictions imposed on the search grid.

Probe set. The expectation in Eq. (7) is approximated by caching paired features on a held-out subset of $N_{\text{probe}} = 200$ clean images. For each clean latent, we draw $t \sim \mathrm{Unif}[0, 1]$ and follow the same forward path as in Eq. (1).

Closed-form solution for the stitching layer search. We approximate the population objective in Eq. (7) by its empirical counterpart on a fixed probe set $\{(\mathbf{x}_0^{(n)}, t^{(n)}, \boldsymbol{\epsilon}^{(n)})\}_{n=1}^{N_{\text{probe}}}$, where each $\mathbf{z}_t^{(n)} = \alpha_{t^{(n)}}\mathbf{z}_0^{(n)} + \sigma_{t^{(n)}}\boldsymbol{\epsilon}^{(n)}$. Stacking the paired features as columns into matrices

$$F^{(i)} = \Big[u_\theta^{\leq i}(\mathbf{z}_t^{(1)}), \ldots, u_\theta^{\leq i}(\mathbf{z}_t^{(N_{\text{probe}})})\Big] \in \mathbb{R}^{d_i^u \times N_{\text{probe}}}, \tag{59}$$

$$G^{(j)} = \Big[r_\phi^{\leq j-1}(\mathbf{z}_0^{(1)}), \ldots, r_\phi^{\leq j-1}(\mathbf{z}_0^{(N_{\text{probe}})})\Big] \in \mathbb{R}^{d_j^r \times N_{\text{probe}}}, \tag{60}$$

the empirical objective reduces to a standard linear least-squares problem

$$\widehat{W}^{\star}_{i,j} = \arg\min_{W \in \mathbb{R}^{d_j^r \times d_i^u}} \Big\| WF^{(i)} - G^{(j)} \Big\|_F^2, \tag{61}$$

whose minimizer admits the closed-form expression

$$\widehat{W}^{\star}_{i,j} = G^{(j)} \left(F^{(i)}\right)^{+}, \tag{62}$$

where $(\cdot)^{+}$ denotes the Moore-Penrose pseudoinverse.

Restricting the reward model depth. We restrict the search to the early blocks of the reward model and do not sweep over its middle or late blocks. Empirically, we observed two consistent failure modes as j moves past the early reward model blocks, independent of the diffusion layer i:

1. the closed-form fitting loss increases sharply, indicating that $W^{\star}_{i,j}$ can no longer linearly reconstruct the deeper reward model features; and
2. even after Stage-2 finetuning with $\mathcal{L}_{\text{value}}$, the resulting predictions are substantially worse than those from early-block stitches.

Based on these observations, we select $(i^{\star}, j^{\star})$ within this restricted search space.

C.2. FK steering with StitchVM

Algorithm 1 summarizes the FK steering variant with StitchVM. The algorithm keeps the standard FK potential and resampling step unchanged: at steps $k \in \mathcal{S}_{\text{FK}}$, particles are weighted by the FK potential G_k and resampled accordingly. The only modification is applied at the proposal-scaling steps $k \in \mathcal{S}_M$. Instead of drawing a single proposal per particle, each particle draws M local proposals from the transition kernel, scores them with StitchVM, and each particle keeps one proposal with the highest StitchVM score. This nested proposal selection increases the number of candidates explored from N to NM at these steps, while avoiding the additional denoiser and decoder evaluations that Tweedie-based approximation would require. In our experiments, $\mathcal{S}_M$ is chosen as the early denoising steps.

C.3. AlignProp & DRaFT with StitchVM

Direct reward finetuning methods such as AlignProp [21] and DRaFT [24] optimize a differentiable reward by generating a clean sample and backpropagating the reward gradient through the denoising trajectory. We describe this procedure using the latent-space notation of Section 3.1. Let θ denote the trainable parameters of the diffusion generator, initialized from the pretrained parameters θ_0. Starting from $\mathbf{z}_1 \sim p_1 = \mathcal{N}(0, I_d)$, a full reverse rollout produces a clean sample $\mathbf{z}_0 \sim p_\theta(\mathbf{z}_0 \mid \mathbf{z}_1)$. A standard direct reward finetuning objective can be written as

$$\mathcal{L}_{\text{DRFT}}(\theta) = -\mathbb{E}_{\mathbf{z}_1 \sim p_1,\, \mathbf{z}_0 \sim p_\theta(\cdot|\mathbf{z}_1)} \left[r_\phi(\mathbf{z}_0) \right] + \lambda \mathcal{R}(\theta; \theta_0), \tag{63}$$

where we use the shorthand $r_\phi(\mathbf{z}_0) = r_\phi(\mathcal{D}(\mathbf{z}_0))$, and $\mathcal{R}(\theta; \theta_0)$ is a regularizer that limits deviation from the pretrained generator, such as an output regularizer [98].

Because the reward in Eq. (63) is evaluated only at the terminal clean sample, optimizing it requires differentiating through the full reverse trajectory from $t = 1$ to $t = 0$. This is memory-intensive and computationally expensive. Moreover, backpropagation through a long denoising chain can lead to unstable or exploding gradients in practice [30]. For this reason, direct reward finetuning is often restricted to the final, low-noise portion of the trajectory, which provides weak or indirect learning signals for early, high-noise timesteps.

Our stitched value model $V_\omega^{(i^\star, j^\star)}$ provides a learned estimate of the terminal reward from an intermediate noisy latent. Instead of sampling all the way to $\mathbf{z}_0$ and evaluating $r_\phi(\mathbf{z}_0)$, we stop the

Algorithm 1 FK Steering with StitchVM Proposal Scaling M

Require: Transition kernels $p_{t_{k-1}|t_k}$; FK potential function G_k; StitchVM $V_\omega^{(i^\star,j^\star)}$; number of particles N; proposals per particle M; FK-resampling steps $\mathcal{S}_{\mathrm{FK}} \subseteq \{1,\dots,K\}$; proposal-scaling steps $\mathcal{S}_M \subseteq \{1,\dots,K\}$; timestep schedule $t_K > \cdots > t_0$.

Sample initial particles $\mathbf{z}_{t_K}^n \sim \mathcal{N}(0, I)$ for $n = 1,\dots,N$.
for $k = K, K-1, \dots, 1$ **do**
  for $n = 1,\dots,N$ **do**
    if $k \in \mathcal{S}_M$ **and** $M > 1$ **then**
      # StitchVM proposal scaling:
      $\mathbf{z}_{t_{k-1}}^{n,m} \sim p_{t_{k-1}|t_k}(\cdot \mid \mathbf{z}_{t_k}^n)$ for $m = 1,\dots,M$. ▷ Sample M proposals
      $m_n^\star = \arg\max_{m\in[M]} V_\omega^{(i^\star,j^\star)}(\mathbf{z}_{t_{k-1}}^{n,m})$. ▷ argmax over M proposals
      $\bar{\mathbf{z}}_{t_{k-1}}^n = \mathbf{z}_{t_{k-1}}^{n,m_n^\star}$.
    else
      Sample one proposal $\bar{\mathbf{z}}_{t_{k-1}}^n \sim p_{t_{k-1}|t_k}(\cdot \mid \mathbf{z}_{t_k}^n)$.
    end if
  end for
  if $k \in \mathcal{S}_{\mathrm{FK}}$ **then**
    for $n = 1,\dots,N$ **do**
      Compute $G^n = G_k(\mathbf{z}_{t_k}^n, \bar{\mathbf{z}}_{t_{k-1}}^n)$. ▷ standard FK potential
    end for
    Sample $a_{t_{k-1}}^n \sim \mathrm{Multinomial}(G^1,\dots,G^N)$ for $n = 1,\dots,N$.
    Set $\mathbf{z}_{t_{k-1}}^n \leftarrow \bar{\mathbf{z}}_{t_{k-1}}^{a_{t_{k-1}}^n}$ for $n = 1,\dots,N$. ▷ standard FK resampling
  else
    Set $\mathbf{z}_{t_{k-1}}^n \leftarrow \bar{\mathbf{z}}_{t_{k-1}}^n$ for $n = 1,\dots,N$.
  end if
end for
return final particles $\{\mathbf{z}_{t_0}^n\}_{n=1}^N$.

reverse process at an intermediate timestep $\tau \in (0,1)$. That is, we sample $\mathbf{z}_\tau \sim p_\theta(\mathbf{z}_\tau \mid \mathbf{z}_1)$ and replace the remaining rollout from $\mathbf{z}_\tau$ to $\mathbf{z}_0$ with a single evaluation of $V_\omega^{(i^\star,j^\star)}$. This yields the value function-based objective

$$\mathcal{L}_{\text{V-DRFT}}(\theta) = -\mathbb{E}_{\tau,\,\mathbf{z}_1\sim p_1,\,\mathbf{z}_\tau\sim p_\theta(\mathbf{z}_\tau|\mathbf{z}_1)}\left[V_\omega^{(i^\star,j^\star)}(\mathbf{z}_\tau)\right] + \lambda\,\mathcal{R}(\theta;\theta_0), \tag{64}$$

where τ is sampled from a prespecified stopping-time distribution over the denoising schedule. This has two practical advantages. First, it provides direct reward supervision at intermediate noisy states, including high-noise regions where terminal-reward backpropagation is weak or unstable. Second, it avoids rollouts of the entire denoising trajectory, reducing computation.

C.4. DiffusionNFT with StitchVM

DiffusionNFT [30] optimizes a reward-weighted forward-process regression objective. We describe it using the latent-space notation of Section 3.1. Let θ denote the current model and let θ_{old} denote the frozen data-collection policy. Starting from $\mathbf{z}_1 \sim p_1$, the data-collection policy generates clean latents $\mathbf{z}_0$ via the reverse process. As in Section 3.2, we use the shorthand $r_\phi(\mathbf{z}_0) = r_\phi(\mathcal{D}(\mathbf{z}_0))$.

Given samples from θ_{old}, DiffusionNFT first transforms the raw reward into an optimality probability. For a normalization group of samples drawn from the data-collection policy, this is written

as

$$r_{\text{norm}}(\mathbf{z}_0) = \frac{r_\phi(\mathbf{z}_0) - \mathbb{E}_{\mathbf{z}'_0 \sim p_{\theta_{\text{old}}}}\left[r_\phi(\mathbf{z}'_0)\right]}{Z}, \qquad r(\mathbf{z}_0) = \frac{1}{2} + \frac{1}{2}\text{clip}\left(r_{\text{norm}}(\mathbf{z}_0), -1, 1\right), \tag{65}$$

where the expectation is estimated over the same normalization group and $Z > 0$ is a normalization scale[3].

Given a clean latent $\mathbf{z}_0$, DiffusionNFT samples a forward noisy state using Eq. (1),

$$\mathbf{z}_t = \alpha_t \mathbf{z}_0 + \sigma_t \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(0, I_d). \tag{66}$$

The corresponding conditional velocity, expressed in terms of the generating variables $(\mathbf{z}_0, \boldsymbol{\epsilon})$, is

$$u_t(\mathbf{z}_t \mid \mathbf{z}_0) = \dot{\alpha}_t \mathbf{z}_0 + \dot{\sigma}_t \boldsymbol{\epsilon}, \tag{67}$$

where $\dot{\alpha}_t$ and $\dot{\sigma}_t$ denote time derivatives. This form is equivalent to the $(\mathbf{z}_t, \mathbf{z}_0)$-parameterization in Section 3.1 and Appendix A.1[4], but is more convenient when $\mathbf{z}_0$ and $\boldsymbol{\epsilon}$ are the natural sampling variables, as is the case here.

DiffusionNFT then constructs implicit positive and negative velocity fields by interpolating between the frozen data-collection velocity $u_{\theta_{\text{old}}}$ and the trainable velocity u_θ:

$$u^+_\theta(\mathbf{z}_t, t) = (1-\beta)\, u_{\theta_{\text{old}}}(\mathbf{z}_t, t) + \beta\, u_\theta(\mathbf{z}_t, t), \tag{68}$$

$$u^-_\theta(\mathbf{z}_t, t) = (1+\beta)\, u_{\theta_{\text{old}}}(\mathbf{z}_t, t) - \beta\, u_\theta(\mathbf{z}_t, t). \tag{69}$$

The DiffusionNFT objective is then constructed as

$$\mathcal{L}_{\text{NFT}}(\theta) = \mathbb{E}_{\mathbf{z}_0 \sim p_{\theta_{\text{old}}}, t, \boldsymbol{\epsilon}}\Big[r(\mathbf{z}_0) \left\| u^+_\theta(\mathbf{z}_t, t) - u_t(\mathbf{z}_t \mid \mathbf{z}_0) \right\|_2^2 + \left(1 - r(\mathbf{z}_0)\right) \left\| u^-_\theta(\mathbf{z}_t, t) - u_t(\mathbf{z}_t \mid \mathbf{z}_0) \right\|_2^2 \Big]. \tag{70}$$

After the update, the data-collection policy is updated by an exponential moving average,

$$\theta_{\text{old}} \leftarrow \rho\, \theta_{\text{old}} + (1-\rho)\, \theta. \tag{71}$$

Our StitchVM-based variant retains the same weighted regression structure but replaces the terminal clean reward with a reward model estimate at a stopped noisy latent. Instead of running the reverse process all the way to $\mathbf{z}_0$, we stop at an intermediate timestep $\tau \in (0, 1)$ and obtain $\mathbf{z}_\tau$ from the data-collection policy. The raw scalar signal is then

$$\widetilde{r}_{\text{raw}}(\mathbf{z}_\tau) = V_\omega^{(i^\star, j^\star)}(\mathbf{z}_\tau). \tag{72}$$

We convert this estimate into an optimality probability using the same standardization as DiffusionNFT:

$$\widetilde{r}_{\text{norm}}(\mathbf{z}_\tau) = \frac{\widetilde{r}_{\text{raw}}(\mathbf{z}_\tau) - \mathbb{E}_{\mathbf{z}'_\tau \sim p_{\theta_{\text{old}}}}\left[\widetilde{r}_{\text{raw}}(\mathbf{z}'_\tau)\right]}{Z}, \qquad \widetilde{r}(\mathbf{z}_\tau) = \frac{1}{2} + \frac{1}{2}\text{clip}\left(\widetilde{r}_{\text{norm}}(\mathbf{z}_\tau), -1, 1\right). \tag{73}$$

It remains to construct the forward-regression target when the anchor is $\mathbf{z}_\tau$ rather than $\mathbf{z}_0$. For $t > \tau$, the Gaussian path in Eq. (1) gives the conditional bridge coefficients

$$\bar{\alpha}_{t|\tau} = \frac{\alpha_t}{\alpha_\tau}, \qquad \bar{\sigma}_{t|\tau} = \sqrt{\sigma_t^2 - \bar{\alpha}_{t|\tau}^2 \sigma_\tau^2}. \tag{74}$$

[3] Following DiffusionNFT's reward-standardization convention [30]; the same Z is reused in our StitchVM-based variant (Eq. (73)).

[4] Substituting $\boldsymbol{\epsilon} = (\mathbf{z}_t - \alpha_t \mathbf{z}_0)/\sigma_t$ into Eq. (67) recovers $u_t(\mathbf{z}_t \mid \mathbf{z}_0) = \frac{\dot{\sigma}_t}{\sigma_t}\mathbf{z}_t + (\dot{\alpha}_t - \alpha_t \frac{\dot{\sigma}_t}{\sigma_t})\mathbf{z}_0$.

Starting from the stopped latent $\mathbf{z}_\tau$, we sample a noisier latent

$$\mathbf{z}_t = \bar{\alpha}_{t|\tau}\mathbf{z}_\tau + \bar{\sigma}_{t|\tau}\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(0, I_d), \qquad t > \tau, \tag{75}$$

with corresponding conditional velocity target

$$u_{t|\tau}(\mathbf{z}_t \mid \mathbf{z}_\tau) = \dot{\bar{\alpha}}_{t|\tau}\mathbf{z}_\tau + \dot{\bar{\sigma}}_{t|\tau}\boldsymbol{\epsilon}, \tag{76}$$

where $\dot{\bar{\alpha}}_{t|\tau}$ and $\dot{\bar{\sigma}}_{t|\tau}$ denote time derivatives of the bridge coefficients. This mirrors the forward-regression construction in Eq. (67), with the anchor replaced by $\mathbf{z}_\tau$.

The value function-based DiffusionNFT objective is therefore

$$\begin{aligned}\mathcal{L}_{\text{V-NFT}}(\theta) = \mathbb{E}_{\mathbf{z}_\tau \sim p_{\theta_{\text{old}}},\, t>\tau,\, \boldsymbol{\epsilon}} \Big[\widetilde{r}(\mathbf{z}_\tau) \left\| u_\theta^+(\mathbf{z}_t, t) - u_{t|\tau}(\mathbf{z}_t \mid \mathbf{z}_\tau) \right\|_2^2 \\ + \left(1 - \widetilde{r}(\mathbf{z}_\tau)\right) \left\| u_\theta^-(\mathbf{z}_t, t) - u_{t|\tau}(\mathbf{z}_t \mid \mathbf{z}_\tau) \right\|_2^2 \Big].\end{aligned} \tag{77}$$

In summary, the original DiffusionNFT uses the terminal clean reward $r_\phi(\mathbf{z}_0)$ to weight positive and negative forward-regression targets anchored at $\mathbf{z}_0$. Our StitchVM-based variant replaces this terminal reward with $V_\omega^{(i^\star,j^\star)}(\mathbf{z}_\tau)$ and anchors the forward regression at the stopped latent $\mathbf{z}_\tau$. Since $V_\omega^{(i^\star,j^\star)}(\mathbf{z}_\tau)$ estimates the expected terminal reward conditioned on the intermediate state, this construction yields a training signal at intermediate denoising steps without completing the reverse process to $\mathbf{z}_0$.

D. Additional Experimental Details

D.1. Stitched Value Model Experiments

StitchVM Implementation details. For searching the stitching-layer index, we extract paired features $\left(u_\theta^{\le i}(\mathbf{z}_t, t),\, r_\phi^{\le j-1}(\mathbf{z}_0)\right)$ for each candidate pair (i, j). The probe set contains $N_{\text{probe}} = 200$ HPDv2 images with $t \sim \text{Unif}[0, 1]$. We cast the cached features to `float64` and solve $W_{i,j}^\star$ in closed form using `torch.linalg.lstsq`. We restrict the reward model side of the search to the first four transformer blocks ($j \le 4$) for all backbone-reward combinations. On the diffusion side, we sweep all DiT-block indices i.

We finetune the stitching layer $s_\psi = (F_\psi, G_\psi)$ together with the truncated reward model suffix $r_\phi^{\ge j}$. The diffusion backbone remains frozen. Optimization uses fused AdamW [99] with base learning rate 1×10^{-5}, weight decay 0, and 100 linear warmup steps. The stitching layer s_ψ uses an additional $5\times$ learning-rate multiplier on top of the base learning rate, since F_ψ is initialized from the SVD fit and G_ψ is zero-initialized, whereas the reward model suffix already starts from a strong pretrained state and only requires mild adaptation. We use a global batch size of 128 images per optimizer step. Noise levels during StitchVM training are drawn from a center-biased distribution over $\sigma \in [0, 1]$, since both endpoints carry little learning signal.

We optimize Eq. (8) as a multi-level distillation between the stitched model and the original frozen reward model evaluated on the same clean image. The exact loss depends on the reward model. For CLIP, DFN-CLIP, and HPSv2, the final score is computed from the inner product between an image embedding and a text embedding. However, our distillation is performed entirely on the image side: the text encoder is not run or updated during StitchVM training. We therefore distill the reward model at the representation level:

$$\mathcal{L}_{\text{value}} = \ell(\tilde{e}_{\text{StitchVM}},\, \tilde{e}_{\text{RM}}) + \lambda_{\text{tok}}\, \ell(h_{\text{StitchVM}},\, h_{\text{RM}}), \tag{78}$$

where $\ell(\cdot,\cdot)$ is a regression loss specified below, h_{StitchVM} and h_{RM} are the per-token output activations of $r_\phi^{\geq j}$ for the stitched model and the original frozen reward model, respectively, e_{StitchVM} and e_{RM} are the corresponding image embeddings, and $\tilde{e}_{\text{StitchVM}} = e_{\text{StitchVM}}/\|e_{\text{StitchVM}}\|_2$ and $\tilde{e}_{\text{RM}} = e_{\text{RM}}/\|e_{\text{RM}}\|_2$ are their ℓ_2-normalized versions. We instantiate ℓ as the squared ℓ_2 loss and set $\lambda_{\text{tok}} = 1$.

For the aesthetic predictor, the final score is a scalar produced by a fixed MLP head $s_{\text{aes}}(\cdot)$ applied to the normalized CLIP image embedding. We therefore add an explicit score-level term on top of the token and embedding losses:

$$\mathcal{L}_{\text{value}} = \lambda_{\text{tok}}\,\ell(h_{\text{StitchVM}},\, h_{\text{RM}}) + \lambda_{\text{emb}}\,\ell(\tilde{e}_{\text{StitchVM}},\, \tilde{e}_{\text{RM}}) + \lambda_{\text{sc}}\,\ell(s_{\text{aes}}(\tilde{e}_{\text{StitchVM}}),\, s_{\text{aes}}(\tilde{e}_{\text{RM}}))\,. \tag{79}$$

The notation for $h_{\text{StitchVM}}, h_{\text{RM}}, \tilde{e}_{\text{StitchVM}}, \tilde{e}_{\text{RM}}$ matches Eq. (78). For the aesthetic case, we instantiate ℓ as the Smooth-ℓ_1 loss and set $\lambda_{\text{tok}} = \lambda_{\text{emb}} = \lambda_{\text{sc}} = 1$. In all cases, the original reward model parameters are frozen, and for the aesthetic predictor, the score head s_{aes} is also frozen.

The noisy latent retrieval and preference benchmarks in Table 5 use 1024×1024 source images. For the training alignment experiments in Table 3, we used StitchVM trained on 512×512 resolution to match the image resolution in [30, 76].

Baseline implementation details. The VAE-stitching baseline (rows "⊕ SD3 VAE" and "⊕ FLUX VAE" in Table 5 and Fig. 2) follows the same training recipe as our StitchVM, but stitches at the *VAE-encoder level* rather than through the diffusion backbone, in the spirit of VIST3A [15] and VGGRPO [81]. Concretely, we replace the diffusion head $u_\theta^{\leq i}$ in Eq. (6) with the identity on the noisy VAE latent. The noisy latent $\mathbf{z}_t$ is fed directly into a 1×1 stitching convolution and a residual MLP, then bilinearly resampled to the patch resolution expected by the reward model suffix $r_\phi^{\geq j}$. The original VIST3A and VGGRPO formulations stitch at $t=0$ using clean latents. We extend the same architecture to the noisy latent regime by sampling $\mathbf{z}_t$ from the same forward process used for StitchVM. Thus, the only conceptual difference between this baseline and our StitchVM is whether the diffusion DiT, and hence noise-aware diffusion features, is present in the front end. The stitching layer is initialized from the closed-form least-squares fit between $\mathbf{z}_t$ and $r_\phi^{\leq j-1}(\mathbf{z}_0)$. This is analogous to Eq. (7), but with $u_\theta^{\leq i} = \mathbf{z}_t$.

We finetune the stitching layer and the reward model suffix for 15 epochs. This is longer than the 5 epochs used for StitchVM because the absence of diffusion features makes the optimization landscape harder. All other optimization hyperparameters match StitchVM: AdamW with base learning rate 1×10^{-5}, a 5× multiplier on the stitching layer, weight decay 0, 100 linear warmup steps. Resolutions follow the standard image-input size of each base model. The distillation losses are identical to those in Eqs. (78)–(79).

For DiNa-LRM [54], we use the original released checkpoint without any additional training or finetuning.

D.2. Inference-Time Alignment Experiments

Shared setup. We use SD 3.5 Medium and SD 3.5 Large for DPS, and additionally FLUX.1-dev for FK steering, all at 1024×1024 resolution. For both DPS and FK steering, we use DrawBench prompts [9] and report ImageReward, Aesthetic Score, HPSv2, and PickScore [93]. For FK steering, we additionally evaluate compositional alignment on GenEval [18], and include reference comparisons to the base flow-matching sampler and Best-of-N (BoN) sampling with the same $N = 4$.

DPS. We use 100 denoising steps with classifier-free guidance scale 4.5 and SDE sampling under LinearSDE of [39]. The DPS guidance strength is swept independently for standard DPS and DPS

with StitchVM, since the two methods produce gradients of different magnitudes: standard DPS backpropagates through the denoiser, VAE, and pixel-space reward, while DPS with StitchVM evaluates the gradient directly in the noisy latent space. For DPS with StitchVM, we sweep over the range $[0.5, 7.0]$, and for standard DPS, we sweep over the range $[0.01, 0.5]$; in both cases, we report the configuration that maximizes the average of the metrics.

FK steering. We use the default sampling configuration of each base generator. For SD 3.5 Medium and SD 3.5 Large, we use 40 denoising steps with classifier-free guidance [100] scale 4.5. For FLUX.1-dev, we use 28 denoising steps with guidance scale 3.5. For SDE sampling, we use the LinearSDE following [39].

For standard FK steering, we apply resampling every 4 denoising steps within a fixed active window. For SD 3.5 Medium and Large, this window spans denoising steps 8 through 32 out of 40 total steps. For FLUX, it spans denoising steps 6 through 22 out of 28 total steps. For FK steering with StitchVM, we apply proposal scaling on a separate schedule from FK resampling: it is applied at every step in the early high-noise portion of sampling, covering approximately the first 40% of the denoising trajectory. In all FK steering experiments with StitchVM, we use $N = 4$ particles and $M = 2$ proposals per particle unless otherwise specified. Other settings follow the default configuration of FK steering.

D.3. Training-Time Alignment Experiments

Shared setup. For both direct reward finetuning and DiffusionNFT, we follow the protocol of [77]. The base generator is SD3.5 Medium, and training prompts are drawn from HPDv2 [20]. The training reward is defined as the equal-weight sum of DFN-CLIP and HPSv2 rewards:

$$\mathcal{R}(\mathbf{z}_0) = \mathcal{R}_{\text{DFN-CLIP}}(\mathbf{z}_0) + \mathcal{R}_{\text{HPSv2}}(\mathbf{z}_0), \tag{80}$$

where $\mathcal{R}_{\text{DFN-CLIP}}$ denotes the DFN-CLIP image-text alignment score and $\mathcal{R}_{\text{HPSv2}}$ denotes the HPSv2 human preference score. Both rewards are evaluated on the decoded clean image, and we use the shorthand $\mathcal{R}(\mathbf{z}_0) = \mathcal{R}(\mathcal{D}(\mathbf{z}_0))$. For variants with StitchVM, we replace the terminal clean reward with the corresponding StitchVM prediction. All training runs use 4 nodes $\times$ 4 NVIDIA GH200 GPUs, for 16 GPUs in total, at 512×512 resolution.

Direct reward finetuning setup. For all direct reward finetuning methods, we finetune the model with LoRA [101]. We use LoRA rank $r = 32$ and $\alpha = 64$. LoRA is applied to all attention projections of the joint MM-DiT blocks: `to_q`, `to_k`, `to_v`, `to_out.0`, `add_q_proj`, `add_k_proj`, `add_v_proj`, and `to_add_out`. The trainable parameters are optimized with AdamW [99] using weight decay 0, learning rate 5×10^{-5}, gradient clipping at norm 1.0, and EMA on the trainable parameters. The regularization weight in Eqs. (63)–(64) is $\lambda = 0.01$. Training sampling uses a 10-step rectified-flow schedule with shift 3.0 at cfg = 1.0. Per training step, we draw 8 prompts per device with 16 gradient-accumulation steps and run 32 sample-and-update batches per epoch. Across all 16 GPUs, this gives an effective batch size of $8\times16\times16 = 2048$ samples per optimizer update. For DRaFT with StitchVM, the stopping step in Eq. (64) is sampled uniformly from the index window $[3, 7]$ of the 10-step schedule.

DiffusionNFT setup. We follow the multi-reward setup from the official DiffusionNFT implementation[5], except that we use the joint DFN-CLIP and HPSv2 reward defined above. For DiffusionNFT

[5] https://github.com/NVlabs/DiffusionNFT

Table 5 | **Results of StitchVM on noisy latents.** ⊕ denotes stitching of a reward model with a pretrained diffusion module (VAE encoder or DiT).

(a) **Zero-shot image-text retrieval (Recall@1) on MSCOCO and Flickr30K.**

	MSCOCO Recall@1										**Flickr30K** Recall@1									
	Image→Text Retrieval					Text→Image Retrieval					Image→Text Retrieval					Text→Image Retrieval				
	Noise level σ					Noise level σ					Noise level σ					Noise level σ				
Method	0.1	0.25	0.5	0.75	0.9	0.1	0.25	0.5	0.75	0.9	0.1	0.25	0.5	0.75	0.9	0.1	0.25	0.5	0.75	0.9
NoisyCLIP	46.07	45.16	32.71	6.14	2.11	35.41	35.38	28.94	7.62	2.35	76.72	73.61	49.66	10.39	2.48	64.48	62.27	47.58	12.04	3.18
CLIP ViT-L/14			57.90					37.09					87.30					67.36		
⊕ SD3.5 VAE	42.62	38.91	21.14	2.78	0.08	31.88	29.27	17.59	4.18	0.37	73.40	67.25	38.30	7.10	0.40	60.90	56.15	35.86	8.59	1.20
⊕ FLUX VAE	39.60	36.24	24.38	4.86	0.20	29.69	27.70	19.10	6.23	0.84	66.60	62.10	41.90	9.60	0.70	56.12	52.96	38.68	13.30	1.08
⊕ SD3.5-M (StitchVM)	56.82	56.22	52.50	32.52	3.68	38.64	38.50	36.67	23.95	5.37	86.30	85.40	82.70	58.00	8.90	67.66	68.28	65.62	47.20	10.46
⊕ SD3.5-L (StitchVM)	**57.22**	**56.90**	54.36	34.10	3.62	39.27	39.36	38.15	25.56	5.43	**86.80**	**87.00**	84.70	60.60	9.40	69.04	**69.70**	67.88	50.88	10.88
⊕ FLUX (StitchVM)	56.40	56.30	**54.46**	**37.00**	**6.00**	**39.43**	**39.89**	**38.60**	**28.26**	**7.61**	86.20	86.60	**84.80**	**65.80**	**15.10**	**69.28**	68.96	**68.58**	**54.70**	**16.32**
DFN-CLIP			70.62					54.16					92.20					80.86		
⊕ SD3.5 VAE	57.24	54.25	37.42	9.84	1.18	47.53	44.77	33.46	9.97	1.14	81.20	75.35	47.40	11.60	1.01	74.72	69.21	50.40	11.57	1.15
⊕ FLUX VAE	54.22	51.58	40.66	10.92	1.31	45.34	43.20	34.97	11.12	1.48	74.40	70.20	51.00	12.10	1.27	69.94	66.02	53.22	14.28	1.19
⊕ SD3.5-M (StitchVM)	**71.56**	71.22	68.24	46.94	6.28	54.22	54.21	52.34	38.04	8.95	93.60	**93.50**	91.90	74.10	13.90	**81.62**	**81.48**	79.66	62.28	14.98
⊕ SD3.5-L (StitchVM)	71.44	**71.56**	**68.78**	48.58	7.14	54.29	54.00	52.54	38.84	9.19	**94.10**	**93.50**	91.80	72.50	14.10	81.48	81.34	**80.16**	62.18	14.86
⊕ FLUX (StitchVM)	71.30	70.84	68.24	**49.38**	**8.58**	**54.39**	**54.24**	**52.84**	**39.93**	**11.78**	93.20	93.10	**92.00**	**74.60**	**18.70**	81.52	81.28	**80.16**	**63.84**	**19.00**

(b) **Preference accuracy on HPDv2 and ImageReward.**

	HPDv2 Benchmark					**ImageReward Benchmark**				
	Noise level σ					Noise level σ				
Method	0.1	0.25	0.5	0.75	0.9	0.1	0.25	0.5	0.75	0.9
DiNa-LRM [54]	81.36	81.41	80.69	78.16	65.18	60.61	60.83	60.61	59.21	54.13
HPSv2			83.28					67.14		
⊕ SD3 VAE	79.67	79.22	76.44	71.49	62.77	60.72	59.56	57.89	54.59	53.00
⊕ FLUX VAE	79.60	79.12	77.96	73.02	65.23	62.27	62.48	60.76	58.14	52.56
⊕ SD3.5-M (StitchVM)	81.78	81.87	80.92	78.90	74.68	**66.82**	67.00	65.25	63.33	58.19
⊕ SD3.5-L (StitchVM)	82.03	82.09	81.40	78.98	75.39	66.68	**67.28**	66.38	63.04	58.71
⊕ FLUX (StitchVM)	**82.53**	**82.61**	**81.98**	**79.25**	**75.90**	66.23	66.49	**66.67**	**64.77**	**59.04**

(c) **Aesthetic-score correlation on AVA.**

	SRCC				
	Noise level σ				
Method	0.1	0.25	0.5	0.75	0.9
Aesthetic Predictor			0.618		
⊕ SD3 VAE	0.438	0.423	0.334	0.146	0.068
⊕ FLUX VAE	0.455	0.451	0.415	0.322	0.177
⊕ SD3.5-M (StitchVM)	0.609	0.610	0.597	0.538	0.369
⊕ SD3.5-L (StitchVM)	**0.614**	0.615	**0.601**	0.545	0.396
⊕ FLUX (StitchVM)	0.613	**0.616**	0.600	**0.560**	**0.433**

with StitchVM, each rollout is stopped early at a step sampled uniformly from $\{12, \ldots, 17\}$ of the 25-step schedule.

Flow-GRPO-Fast setup. We follow the PickScore setup from the official Flow-GRPO-Fast implementation[6], except that we replace the reward with the joint DFN-CLIP and HPSv2 reward defined above and run the method on the same 16-GPU setup as the other training-time alignment experiments.

Evaluation protocol. For sample generation, we use 40 denoising steps at cfg = 1.0 for all methods. All methods are evaluated on fully denoised samples using the original clean-image reward models, including the variants with StitchVM. We report total GPU-hours per run in the "GPU-h" column of Table 3 as a wall-clock training-cost measure. All timings are measured on the same 16-GH200 layout.

E. Additional Experimental Results

E.1. Full Numerical Results of StitchVM Performance

For completeness, Table 5 reports the full numerical results corresponding to the line plots in Figure 2. The table is organized into three evaluation settings: (a) zero-shot image-text retrieval on MSCOCO [89] and Flickr30K [90], evaluated with Recall@1 in both Image→Text and Text→Image directions, for CLIP ViT-L/14 [49] and DFN-CLIP [70]; (b) preference accuracy on HPDv2 [20] and ImageReward [47], for HPSv2 [20]; and (c) aesthetic score correlation on the AVA test split [88], evaluated with SRCC, for the Aesthetic Predictor [71]. Each setting reports results at noise levels $\sigma \in \{0.1, 0.25, 0.5, 0.75, 0.9\}$.

[6] https://github.com/yifan123/flow_grpo

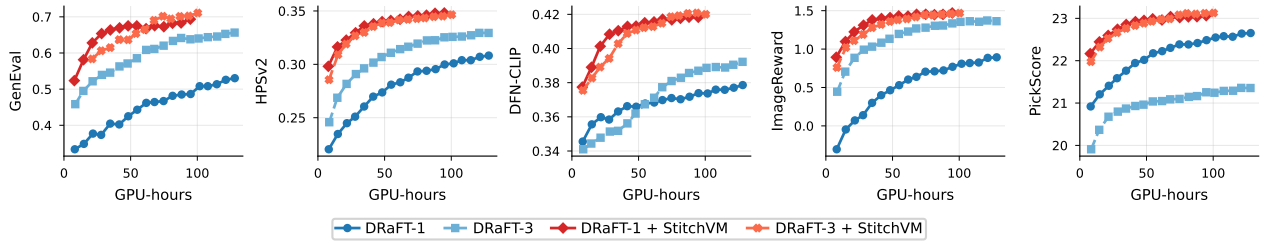


Figure 4 | **StitchVM improves the quality–efficiency trade-off of DRaFT.** GenEval, HPSv2, DFN-CLIP, ImageReward, and PickScore plotted against GPU-hours for DRaFT and DRaFT with StitchVM during finetuning on the joint DFN-CLIP and HPSv2 reward.

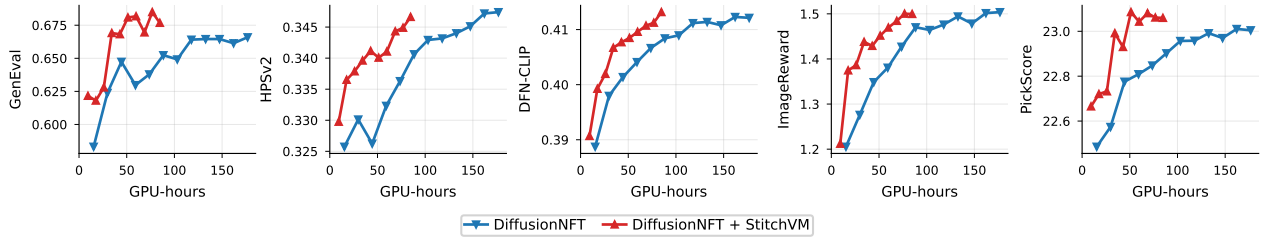


Figure 5 | **StitchVM accelerates DiffusionNFT training.** GenEval, HPSv2, DFN-CLIP, ImageReward, and PickScore plotted against GPU-hours for DiffusionNFT and DiffusionNFT with StitchVM during finetuning on the joint DFN-CLIP and HPSv2 reward.

E.2. Training Curves in Training-Time Alignment

Figures 4 and 5 report training curves for DRaFT and DiffusionNFT during finetuning on the joint DFN-CLIP and HPSv2 reward. We plot GenEval, HPSv2, DFN-CLIP, ImageReward, and PickScore against GPU-hours to evaluate both training-reward metrics and held-out metrics. For DRaFT (Fig. 4), variants with StitchVM reach higher final scores with substantially less compute across metrics, showing an improved quality–efficiency trade-off. For DiffusionNFT (Fig. 5), variants with StitchVM reach a similar plateau to the original method in roughly half the GPU-hours, indicating faster training while preserving final quality. Overall, these curves show that the gains from StitchVM are consistent across compositional alignment, training rewards, and held-out preference metrics.

E.3. Analysis of Stitching Interface Search

The closed-form feature-matching loss in Eq. (7) is intended as a cheap search protocol for the stitch interface (i, j). To test this, we sweep (i, j) over the full grid, fit $W^{\star}_{i,j}$ in closed form, and then train the stitched value model end-to-end. Figure 6 reports the closed-form feature-matching loss alongside the MSCOCO Recall@1 of the trained model.

The closed-form loss sharply filters out poor interfaces but does not precisely identify the best one. Once j moves beyond the early CLIP blocks, the loss rises by roughly an order of magnitude, and Recall@1 drops from around 49 to below 5; Stage-2 finetuning cannot recover from these poor interfaces. In contrast, within the low-loss region ($j \leq 4$), Recall@1 remains uniformly high and only weakly correlates with the loss: the lowest-loss cell $(i, j) = (4, 1)$ reaches Recall@1 of 48.3, compared to a within-region maximum of 49.7.

We exploit this asymmetry in our search. Since high-loss configurations cannot be recovered through finetuning, we restrict the reward model cut to the early CLIP blocks ($j \leq 4$; Appendix C.1) and select the lowest-loss cell within this range. This avoids catastrophic stitch points at a cost of

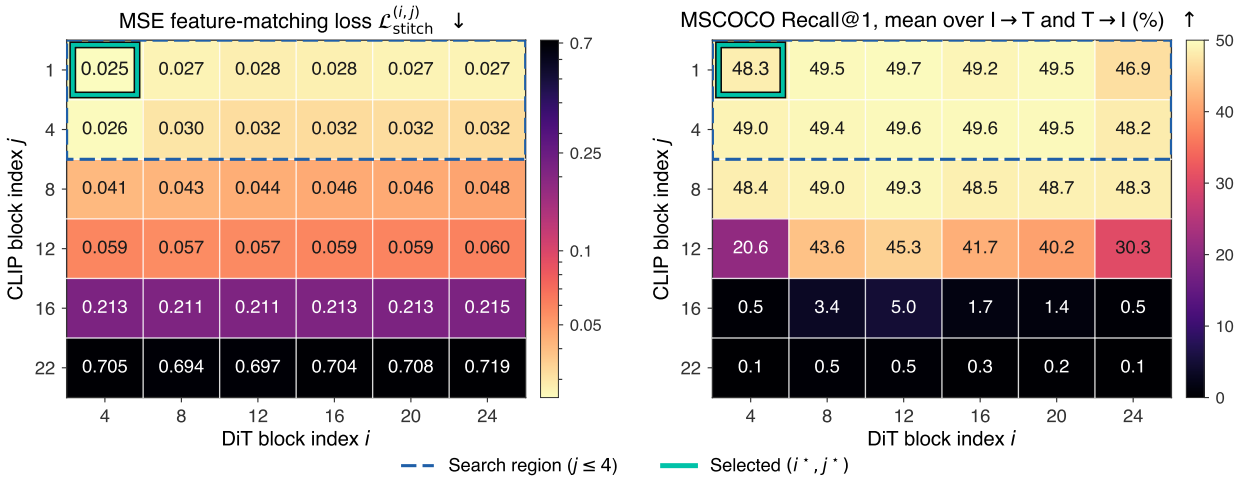


Figure 6 | **Stitching interface analysis on CLIP ViT-L/14 with SD 3.5 Medium.** We sweep the DiT block index i and CLIP block index j, fit the closed-form stitching map, and then run Stage-2 training. *Left*: closed-form MSE feature-matching loss. *Right*: MSCOCO Recall@1 averaged over image-to-text and text-to-image retrieval at $\sigma = 0.1$ after Stage-2 training. The closed-form loss filters out catastrophic interfaces, especially for later CLIP blocks where Recall@1 drops sharply, but is weakly aligned with the final score within the early low-loss region. Based on this observation, we restrict the search to $j \le 4$ (dashed box) and select the lowest-loss cell $(i^\star, j^\star) = (4, 1)$.

about 1.5 Recall@1 points relative to the within-region maximum, in exchange for a much cheaper search.

E.4. Cross-Backbone Generalization of StitchVM

Many diffusion and flow-based generators [8, 68] are released at multiple model scales while sharing the same VAE latent space. Here, we ask whether a StitchVM trained on a smaller backbone can guide a larger generator, and how much performance is lost relative to a StitchVM trained on the same backbone in FK steering.

We keep the FK steering setup of Section 5.2 on SD 3.5 Large, but replace the SD 3.5 Large StitchVM with a StitchVM trained on SD 3.5 Medium. SD 3.5 Medium and SD 3.5 Large share the same SD3 16-channel VAE, so their noisy latents are dimensionally compatible at every noise level. Table 6 reports the same five metrics as Table 2.

Across the fifteen reward–metric cells, the SD 3.5 Medium StitchVM closely matches the SD 3.5 Large StitchVM: HPSv2 differs by at most 0.002, the other metrics remain similar, and under HPSv2 reward, the SD 3.5 Medium StitchVM achieves higher GenEval than the SD 3.5 Large StitchVM (0.72 vs. 0.70). Because StitchVM uses only the early blocks of its diffusion backbone, building it on SD 3.5 Medium rather than SD 3.5 Large reduces the per-step value model cost. A StitchVM trained on the smaller backbone can therefore guide the larger generator at lower inference cost with little loss in alignment quality. This property can further reduce the cost of M-scaling in FK steering, since a smaller StitchVM makes each additional proposal cheaper while still effectively guiding the larger generator.

Table 6 | **A StitchVM with a smaller generator backbone can guide a larger generator with only minor degradation.** We apply FK steering to SD3.5-Large using either an SD3.5-Large StitchVM (same-backbone) or an SD3.5-Medium StitchVM (cross-backbone). ImgRwd: ImageReward, Aes: Aesthetic, Pick: PickScore.

StitchVM backbone	ImgRwd ↑	Aes ↑	HPSv2 ↑	Pick ↑	GenEval ↑
Target reward: HPSv2 score					
SD3.5-Large (same-backbone)	**1.20**	**5.52**	**0.310**	**23.11**	0.70
SD3.5-Medium (cross-backbone)	1.17	5.50	0.309	23.06	**0.72**
Target reward: Aesthetic score					
SD3.5-Large (same-backbone)	1.03	**5.68**	**0.298**	22.87	**0.68**
SD3.5-Medium (cross-backbone)	**1.05**	5.65	0.297	**22.91**	0.67
Target reward: CLIP score					
SD3.5-Large (same-backbone)	**1.20**	5.40	**0.298**	**22.96**	**0.71**
SD3.5-Medium (cross-backbone)	1.10	**5.42**	0.296	22.88	0.69

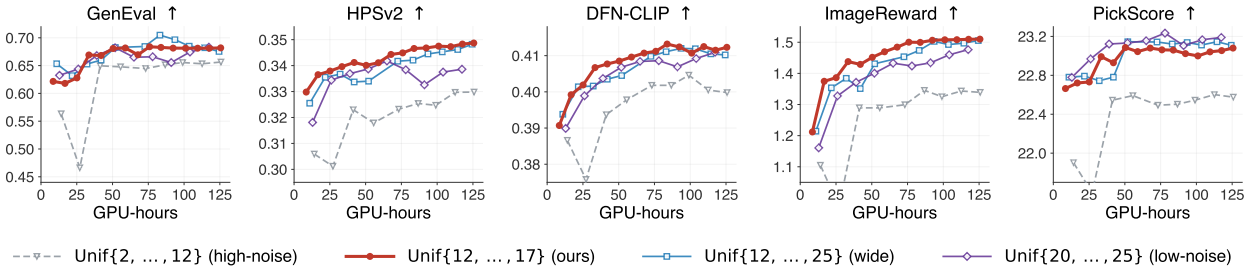


Figure 7 | **Effect of the stopping-step distribution in DiffusionNFT with StitchVM.** We train DiffusionNFT with StitchVM on SD 3.5 Medium with the joint DFN-CLIP and HPSv2 reward, varying the window from which the stopping step is sampled uniformly over a 25-step denoising schedule. Smaller step indices correspond to earlier, higher-noise latents, while larger indices correspond to later, lower-noise latents. The high-noise window Unif$\{2, \ldots, 12\}$ underperforms, while intermediate windows perform substantially better. Our default Unif$\{12, \ldots, 17\}$ achieves a strong quality–efficiency trade-off, reaching competitive final scores while converging faster than the wider window Unif$\{12, \ldots, 25\}$.

E.5. Stopping-Step Distribution for RL Finetuning with StitchVM

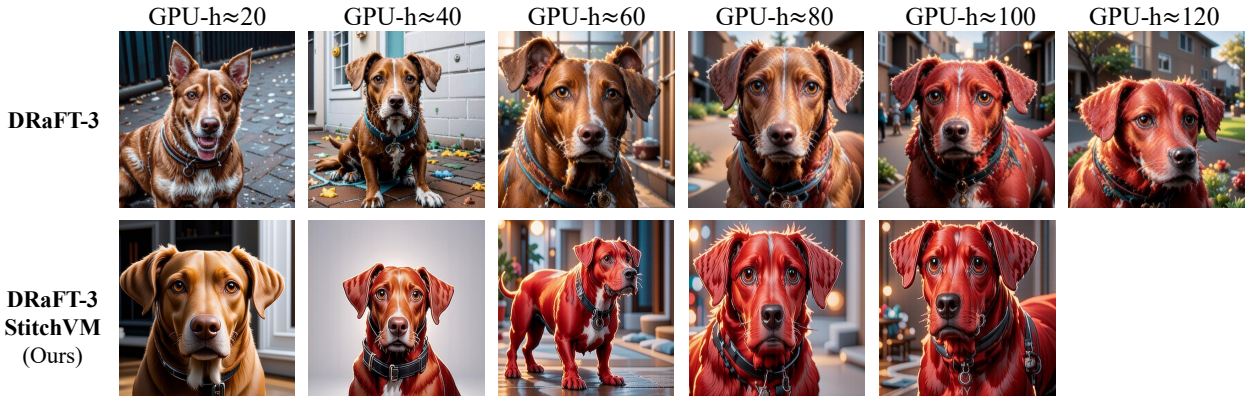
The training-time methods in Section 4.3 stop the rollout at an intermediate noisy latent $\mathbf{z}_\tau$ and use the StitchVM value function $V_\omega^{(i^\star, j^\star)}(\mathbf{z}_\tau)$ in place of the terminal reward. Here, we ablate the distribution from which the stopping step is sampled. In our 25-step denoising schedule, smaller step indices correspond to earlier, higher-noise latents, while larger step indices correspond to later, cleaner latents.

On this schedule, we sample the stopping step uniformly from one of four windows: a high-noise window Unif$\{2, \ldots, 12\}$, a tight intermediate window Unif$\{12, \ldots, 17\}$, a wide intermediate-to-low-noise window Unif$\{12, \ldots, 25\}$, and a low-noise window Unif$\{20, \ldots, 25\}$. The tight intermediate window is our default.

Figure 7 reports GenEval, HPSv2, DFN-CLIP, ImageReward, and PickScore as a function of GPU-hours. The high-noise window Unif$\{2, \ldots, 12\}$ consistently underperforms, suggesting that stopping too early in the denoising trajectory yields value function targets that are less useful for finetuning. In contrast, including intermediate-noise latents substantially improves performance. Among the tested

Prompt: A black colored car.

Figure 8 | **Qualitative comparison between DRaFT-1 and DRaFT-1 with StitchVM across training GPU-hours.**



Prompt: A red colored dog.

Figure 9 | **Qualitative comparison between DRaFT-3 and DRaFT-3 with StitchVM across training GPU-hours.**

choices, Unif$\{12, \ldots, 17\}$ provides the best quality–efficiency trade-off: it reaches strong final scores across metrics and converges faster than the wider window Unif$\{12, \ldots, 25\}$. The low-noise window Unif$\{20, \ldots, 25\}$ remains competitive on some metrics, such as PickScore, but is less stable overall.

We therefore use Unif$\{12, \ldots, 17\}$ as the default stopping-step distribution for all DiffusionNFT and DRaFT runs with StitchVM in Table 3.

E.6. Qualitative Results on RL Finetuning with StitchVM

Figures 8, 9, and 10 show qualitative comparisons of DRaFT-1, DRaFT-3, and DiffusionNFT, with and without StitchVM, across training GPU-hours. Across all three methods, the StitchVM-augmented variants reach the target prompt earlier and produce visually higher-quality samples throughout training (e.g., sharper details and more saturated colors, characteristic of HPSv2-tuned outputs).

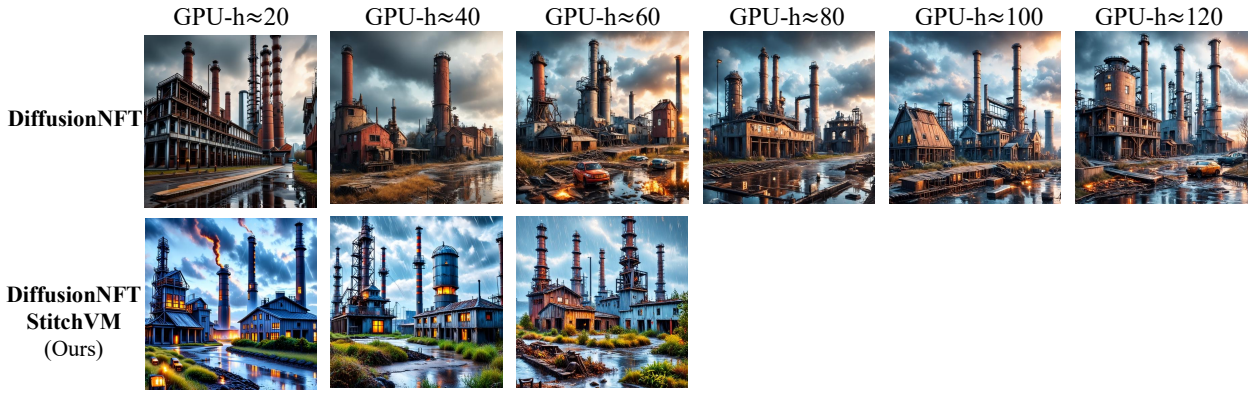


Figure 10 | **Qualitative comparison between DiffusionNFT and DiffusionNFT with StitchVM across training GPU-hours.**

F. Limitation

StitchVM enables transferring feedforward-model-based rewards to noisy latents, but it does not directly apply to rewards that are not implemented as feedforward models. We believe this limitation could be addressed by training surrogate reward models for such rewards, but we leave this direction to the future scope of this work.

Future directions. In this work, we focus on a simple method rather than more complex alternatives that may further improve performance. In our view, timestep-aware training methods [102–106], which have demonstrated effectiveness in diffusion models, represent a promising future direction for improving StitchVM.

G. Broader Impacts

Potential positive impacts. We present StitchVM, a practical framework for training noisy latent value models from existing pretrained reward models. By making value-model training substantially cheaper, StitchVM can improve and accelerate reward-based alignment methods for diffusion and flow models. This may help make generative models more controllable, more aligned with human preferences, and easier to adapt to downstream tasks where clean-image reward models are already available. More broadly, our work suggests a practical direction for value-model-based alignment, which could further accelerate research on safer and more reliable generative modeling.

Potential negative impacts. The same improvements in controllability and reward optimization may also be misused. For example, stronger alignment and steering methods could be used to generate more persuasive synthetic images, including misleading or deceptive visual content. StitchVM may also make it easier to optimize diffusion-based generation toward rewards, including poorly specified or harmful objectives. These risks are not unique to our method, but our work could lower the cost of applying reward-based steering and post-training. We therefore encourage the use of StitchVM together with appropriate safeguards for misuse, bias, and harmful content generation.