Chapter 5: Conditional Diffusion Models and Steering

We introduced diffusion models as learning and sampling from an *unconditional* data distribution P_{data}. In practical applications, however, *conditional* generation is far more common. For instance, in text-to-image generation, text prompts provide conditional information and the generated images should align with the semantic content of the prompts. In protein design, biochemical properties (e.g., stability, binding affinity, solubility) are crucial for validating candidate sequences and structures, requiring generation under desired property constraints. In decision making, actions must be selected based on the current system state, leading to policies that model a conditional distribution of actions given states. These examples motivate conditional diffusion models and, more broadly, *steering* mechanisms that improve condition alignment while retaining sample quality.

Roughly speaking, many high-performing conditional diffusion systems combine (i) a **pre-training** stage and (ii) a **post-training** stage. In the pre-training stage, we follow the same pipeline developed in previous chapters, but replace the unconditional score function by a **conditional score function**. In the post-training stage, we further modify the trained model to improve condition alignment and control the fidelity-diversity tradeoff. This chapter introduces both stages, with an emphasis on post-training methods.

1 Conditional Diffusion Models

A canonical use case of conditional diffusion model is class-conditional image generation, where we generate images in categories such as cats, dogs, or ships. Let X_0 denote the clean image and let Y denote the class label. The goal is to learn and sample from the conditional distribution $P_{\text{data}}(\cdot \mid Y)$.

A conditional diffusion model treats X_0 and Y asymmetrically: Gaussian noise is progressively added to X_0, while the condition Y is maintained throughout the diffusion process. Accordingly, the forward process is identical to the unconditional VP diffusion, except that the initial distribution is conditional:

$$\mathrm{d}X_t = -\frac{1}{2}X_t \mathrm{d}t + \mathrm{d}W_t, \qquad X_0 \sim P_{\text{data}}(\cdot \mid Y), \qquad t \in [0, T], \tag{1.1}$$

where W_t is a Wiener process. For convenience, denote by $P_t(\cdot \mid Y)$ (resp. $p_t(\cdot \mid Y)$) the marginal distribution (resp. density) of X_t given Y. By time reversal, the forward process admits the reverse-time SDE,

$$\mathrm{d}\overline{X}_t = \left[\frac{1}{2}\overline{X}_t + \nabla_x \log p_{T-t}(\overline{X}_t \mid Y)\right] \mathrm{d}t + \mathrm{d}\overline{W}_t, \qquad \overline{X}_0 \sim P_T(\cdot \mid Y). \tag{1.2}$$

Here, the term $\nabla_x \log p_t(x \mid y)$ is the **conditional score function** and $\overline{W}_t$ is an independent Wiener process. Training a conditional diffusion model amounts to learning this conditional score function from paired data $(X_0, Y) \sim P_{\text{data}}$.

Following the same steps used to derive denoising score matching for unconditional diffusion models, we obtain the conditional denoising score matching objective:

$$\min_{s \in \mathcal{F}} \int_0^T \mathbb{E}_{(X_0,Y)\sim P_{\text{data}}} \mathbb{E}_{X \sim \mathsf{N}\left(e^{-t/2}X_0, (1-e^{-t})I\right)} \left[\left\| s(X, Y, t) + \frac{X - e^{-t/2}X_0}{1 - e^{-t}} \right\|_2^2\right] \mathrm{d}t, \tag{1.3}$$

where $\mathcal{F}$ is a function class of conditional score estimators. Compared to the unconditional case, the only change is that the score network takes Y as an additional input.

Classifier-Free Guidance (CFG, [2]). A conditional diffusion model learns a conditional score network $s(x, y, t) \approx \nabla \log p_t(x \mid y)$. In principle, sampling with this conditional score network should already produce samples aligned with label y. In practice, however, conditional generation faces a persistent tension: the "right" condition alignment is *application-dependent* and should be *adjustable at sample generation* without retraining. Classifier-free guidance (CFG) resolves the issue by training one score network to operate in both conditional and unconditional modes.

The key idea of CFG is to randomly drop the condition during training. We introduce a "null" condition $\varnothing$ (e.g., an all-zero embedding) and replace Y by $\varnothing$ with probability γ_{drop} when forming the network input. A typical choice of γ_{drop} is between 0.1 to 0.2. As a result, the score network yields two score estimates:

$$s(x, y, t) \approx \nabla_x \log p_t(x \mid y) \quad \text{and} \quad s(x, \varnothing, t) \approx \nabla \log p_t(x),$$

where the first one is the conditional score and the second is the unconditional score. The training objective of CFG becomes

$$\min_{s \in \mathcal{F}} \int_0^T \mathbb{E}_{(X_0,Y)\sim P_{\text{data}}} \mathbb{E}_{X\sim \mathsf{N}\left(e^{-t/2}X_0,(1-e^{-t})I\right)} \Bigg[(1-\gamma_{\text{drop}}) \left\| s(X,Y,t) + \frac{X - e^{-t/2}X_0}{1-e^{-t}} \right\|_2^2 + \gamma_{\text{drop}} \left\| s(X,\varnothing,t) + \frac{X - e^{-t/2}X_0}{1-e^{-t}} \right\|_2^2 \Bigg] \mathrm{d}t.$$

Once trained, CFG forms a guided score by extrapolating from the unconditional score toward the conditional score:

$$s_{\text{CFG}}(x, y, t) = s(x, \varnothing, t) + w\big(s(x, y, t) - s(x, \varnothing, t)\big) = (1 + w)s(x, y, t) - w \cdot s(x, \varnothing, t), \tag{1.4}$$

where $w \geq 0$ is the guidance scale. Interestingly, setting $w \neq 0$ renders (1.4) not a well-defined gradient field of any log densities. This cast mathematical doubts on the use of guidance scale, however, in practice, setting a positive guidance scale often leads to desired performance; see Figure 1.

Effects of guidance scale. Despite treated as a tuning parameter in applications, the guidance scale has some tractable influence on the generated distribution and samples. A well-known result is that increasing the guidance scale improves the label alignment but reduces diversity within a class as shown in Figure 2. More interestingly, the effect of guidance scale relies on the location of the components in the mixture of distributions and the discretization step size for sample generation. As shown in Figure 3, a large guidance scale can split components and generates artificial point clusters deviating from the ground-truth distribution.

2 Post-training

Pre-training provides a base generative model (unconditional or weakly conditional) that captures the data characteristics and yields high sample quality. Post-training (also called *steering*, *align-*

Model	FID (↓)	IS (↑)
BigGAN-deep, max IS (Brock et al., 2019)	25	253
BigGAN-deep (Brock et al., 2019)	5.7	124.5
CDM (Ho et al., 2021)	3.52	128.8
LOGAN (Wu et al., 2019)	3.36	148.2
ADM-G (Dhariwal & Nichol, 2021)	2.97	-
Ours	$T = 128/256/1024$	
$w = 0.0$	8.11 / 7.27 / 7.22	81.46 / 82.45 / 81.54
$w = 0.1$	5.31 / 4.53 / 4.5	105.01 / 106.12 / 104.67
$w = 0.2$	3.7 / 3.03 / 3	130.79 / 132.54 / 130.09
$w = 0.3$	3.04 / **2.43** / **2.43**	156.09 / 158.47 / 156
$w = 0.4$	3.02 / 2.49 / 2.48	183.01 / 183.41 / 180.88
$w = 0.5$	3.43 / 2.98 / 2.96	206.94 / 207.98 / 204.31
$w = 0.6$	4.09 / 3.76 / 3.73	227.72 / 228.83 / 226.76
$w = 0.7$	4.96 / 4.67 / 4.69	247.92 / 249.25 / 247.89
$w = 0.8$	5.93 / 5.74 / 5.71	265.54 / 267.99 / 265.52
$w = 0.9$	6.89 / 6.8 / 6.81	280.19 / 283.41 / 281.14
$w = 1.0$	7.88 / 7.86 / 7.8	295.29 / 297.98 / 294.56
$w = 2.0$	15.9 / 15.93 / 15.75	378.56 / 377.37 / 373.18
$w = 3.0$	19.77 / 19.77 / 19.56	409.16 / 407.44 / 405.68
$w = 4.0$	21.55 / 21.53 / 21.45	**422.29** / 421.03 / 419.06

Figure 1: Table 2 in [2]. Different guidance scale leads to varying performance of image generation on ImageNet 128 × 128.

Figure 2: Effects of increasing guidance scale in a Gaussian mixture model of three components. From left to right, the guidance scale increases.

ment, or *control*) modifies the sampling dynamics or the network to bias generation toward a desired event or preference *without* sacrificing the base model's generative competence.

In this section, we first formalize the goal of post-training from two complementary viewpoints: (i) a *measure transport / conditional sampling* viewpoint, and (ii) a *reward maximization* viewpoint. We then organize post-training methods into two broad categories: (1) *network-training based* methods and (2) *network-training free* methods.

2.1 Two Viewpoints of Post-training Objective

Viewpoint I: measure transport to a conditioned target. Let P_{data} denote the (unknown) data distribution on X_0. A post-training objective is often specified through an *event* or *constraint*

$$\mathcal{E} \qquad \text{or more generally} \qquad \mathcal{E} = \{x_0 : c(x_0) \le 0\},$$

where c is some measurable function. The goal is to sample from the *conditioned* distribution

$$P_{\text{data}}(X_0 \mid \mathcal{E}). \tag{2.1}$$

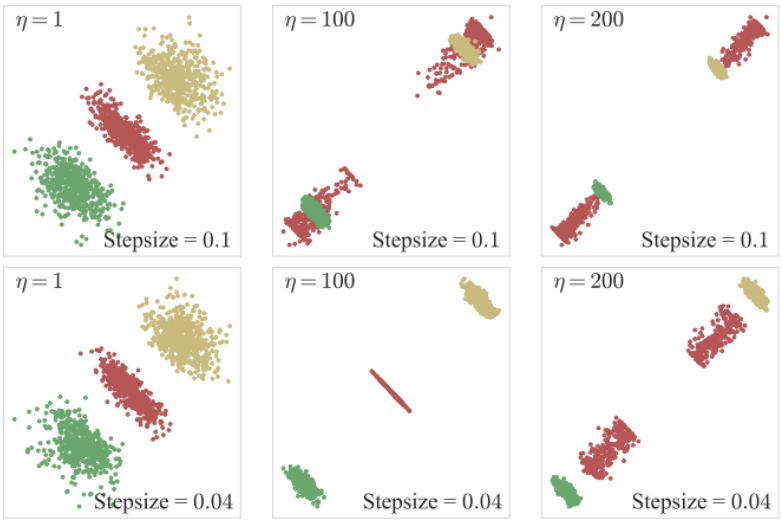

Figure 3: Generated sample under large guidance scale with $\eta = 1 + w$ and different discretization step sizes. The initial distribution is a Gaussian mixture with three components. Their mean vectors are aligned. A large guidance scale splits the middle component.

The ability to transform unconditioned $P_{\text{data}}(\cdot)$ to conditional $P_{\text{data}}(\cdot \mid \mathcal{E})$ can be formulated as a measure transport problem, which motivates methods connecting to optimal transport and stochastic control.

In applications, $\mathcal{E}$ may represent a hard feasibility constraint (e.g., a protein must satisfy a stability threshold), a semantic constraint (e.g., match a text prompt), or a safety constraint (e.g., avoid disallowed content).

Viewpoint II: optimization under a reward. A second viewpoint specifies steering with respect to a *reward* (or *energy*) function

$$r : \mathbb{R}^d \to \mathbb{R}.$$

The reward function prefers samples with larger values, and it encapsulates both explicit performance metrics such as FID and IS scores, and abstract quality measurements such as the aesthetic quality and artistic styles.

With the reward function, we formulate post-training objective as an optimization problem:

$$\max_{P \in \mathcal{P}} \ \mathbb{E}_{X \sim P}[r(X)], \tag{2.2}$$

where $\mathcal{P}$ encodes plausible generative distributions, often taken as a ball around the pre-trained distribution. We note that (2.2) attempts to maximize the average reward of the generated samples, and operates on distributions within a constrained set $\mathcal{P}$. For broader applications, we define $f(r(X); P)$ as a generic reward gauge function, which include the average reward as a special case. Then it is convenient to cast (2.2) into a regularized version:

$$\max_{P} \ f(r(X); P) + \lambda \cdot \texttt{Reg}(P),$$

where λ is the regularization coefficient and `Reg` is chosen to prevent reward hacking as shown in Figure 4.

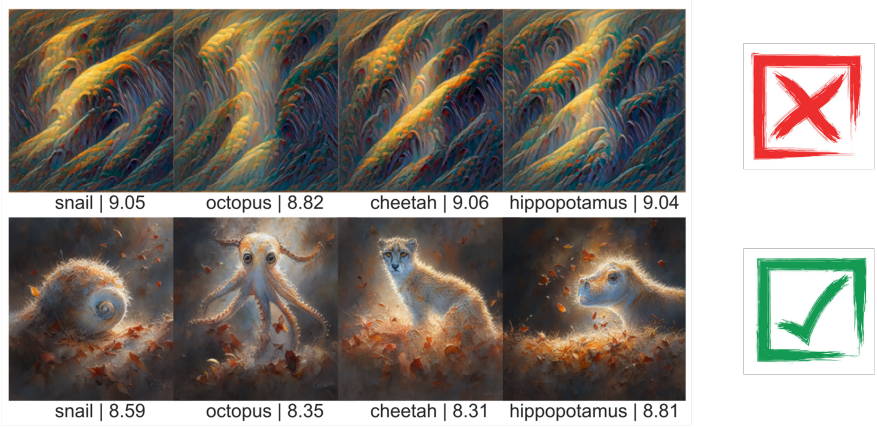

Figure 4: Reward hacking in [4]. Upper row has large reward but meaningless images. Lower row has large reward and meaningful images.

2.2 Training-Based and Training-Free Methods

We group post-training methods by whether they *update network parameters*.

- **Training-based post-training.** One (or more) networks are trained or fine-tuned using an auxiliary signal (labels, a classifier, a reward model, human preference data, or a simulator).
- **Training-free post-training.** No network parameters are updated. Steering is performed at inference time by modifying the sampler, e.g., by adding an explicit guidance term computed from a fixed classifier/reward model, or embedding a sampling based correction step.

In what follows, we start from the most classical training-based idea: *classifier guidance*. This method is historically important because it makes the correction to the pre-trained model explicit via Bayes' rule, and it serves as a template for many modern post-training techniques.

2.3 Training-Based Methods I: Classifier Guidance

Classifier guidance considers again the simplest setting where we want to generate samples according to categories. Via Bayes' rule, we have

$$\log p_t(x \mid y) = \log p_t(x) + \log p_t(y \mid x) - \log p_t(y),$$

hence

$$\nabla_x \log p_t(x \mid y) = \nabla \log p_t(x) + \nabla_x \log p_t(y \mid x). \tag{2.3}$$

Equation (2.3) is the key for classifier guidance: it expresses the *conditional score* as the *unconditional score* plus a correction term given by the gradient of the classifier log-likelihood.

Classifier guidance realizes the measure transport from $P_{\text{data}}(\cdot)$ to $P_{\text{data}}(\cdot \mid y)$, by introducing a dynamic correction in (2.3). Namely, $P_{\text{data}}(\cdot)$ and $P_{\text{data}}(\cdot \mid y)$ are simply terminal distributions of the backward sample generation process. Classifier guidance modifies the backward process at every time t. Such a propagation of terminal distribution requirement to dynamic modification is persistent in diffusion model post-training methods.

Noisy classifier at time t. Recall that p_t is the marginal density in the forward diffusion process, which is time dependent. Classifier guidance therefore trains a time-dependent classifier $c(y \mid x, t)$ that predicts y from a noisy sample. Specifically, at time t, we generate a data set consisting of pairs (X_t, Y), where X_t is obtained by first sampling X_0 and following the forward process $X_t = e^{-t/2}X_0 + \sqrt{1 - e^{-t}}Z$ for $Z \sim \mathsf{N}(0, I)$. Label Y is the original label of X_0. On such a data set, we train a classifier using supervised learning algorithms, e.g., minimizing a cross-entropy loss. After training, we compute a guided score by

$$s_{\mathrm{CG}}(x, y, t) = s(x, t) + w \cdot \nabla_x \log c(y \mid x, t),$$

where s is the pre-trained score function and w is a scaling factor acting identically to the guidance scale in CFG.

Classifier guidance can also be understood from a reward optimization perspective. For a fixed label y, we set the reward function $r_t(x) = \log c(y \mid x, t)$, then classifier guidance is to generate high-reward samples by introducing $\nabla r_t(x)$ into the pre-trained score function.

Limitations of classifier guidance. Classifier guidance can provide condition alignment, but it is sensitive to (i) classifier accuracy under high noise levels, (ii) miscalibration / adversarial directions of $\nabla_x \log c$, and (iii) scale mismatch between the pre-trained score and classifier gradient, which can cause artifacts or mode collapse. Moreover, classifier guidance can only apply to discrete conditions with a relatively small number of categories. For continuous and complicated text prompts conditioning, classifier guidance is hard to implement. These issues motivate both improved training-based approaches (better objectives, joint training, preference modeling) and training-free approaches (more stable inference-time corrections and sampling-based corrections).

As a side note, classifier-free guidance was proposed after classifier guidance. As we have seen, classifier-free guidance does not require the training of a classifier, but jointly trains the conditional and unconditional score functions. This provide stability of the training but at the cost of computational overhead.

2.4 Training-Based Methods II: Universal Guidance

Classifier guidance corrects an unconditional score using an auxiliary noisy classifier trained to predict a (typically discrete) label y from noisy samples (x_t, t). This design does not readily extend to generic or continuous conditions (text prompts, spatial constraints, attributes), for which a classifier likelihood $p_t(y \mid x)$ may be ill-defined or impractical to learn. Moreover, the classifier accuracy for noisy samples bottlenecks the performance.

Universal Guidance (UG) [1] generalizes classifier guidance by replacing the classifier logit gradient with the gradient of a *generic loss function* $\ell(f(x_0); y)$, where f is trainable, mapping features x_0 to response y. Note that UG only trains on clean data pairs (x_0, y) to minimize the loss function. A key caveat arises: Diffusion model evolves in the noisy state space x_t; how to utilize the loss function for dynamically modify the sample generation? A short answer is that UG applies the loss function to an estimate of x_0 computed from x_t.

Guidance via a loss on the clean data estimate. Given a pre-trained diffusion model, the score network can be viewed as a noise predictor given X_t and t. Therefore, we find a clean data

x_0-prediction, which we denote as $\widehat{x}_0(x_t, t)$. Note that this clean data prediction depends on the pre-trained diffusion model.

Let $\ell(x_0; y)$ be a differentiable loss that measures how well a *clean* sample x_0 satisfies condition y. UG defines a time-dependent guidance loss by composing ℓ with the clean data estimate:

$$\mathcal{L}_t(x_t; y) = \ell(f(\widehat{x}_0(x_t, t)); y).$$

The modified score function of UG is then obtained by adding a correction term proportional to the gradient of $\mathcal{L}_t$ with respect to the current noisy state x_t,

$$s_{\mathrm{UG}}(x_t, y, t) = s(x_t, t) - w \cdot \nabla_{x_t} \mathcal{L}_t(x_t; y) = s(x_t, t) - w \cdot \nabla_{x_t} \ell(f(\widehat{x}_0(x_t, t)); y), \tag{2.4}$$

where again s is the pre-trained score function and w is the guidance scale. Equation (2.4) is an analogue of classifier guidance for generic conditions: it steers the noisy trajectory x_t in a direction that improves alignment *in clean data space* as measured by ℓ.

The correction term in (2.4) can be written explicitly by the chain rule,

$$\nabla_{x_t} \ell(f(\widehat{x}_0(x_t, t)); y) = \left(\frac{\partial \widehat{x}_0(x_t, t)}{\partial x_t} \right)^\top \nabla_{x_0} \ell(f(x_0); y) \Big|_{x_0 = \widehat{x}_0(x_t, t)}.$$

Thus, UG uses a gradient in clean space $\nabla_{x_0} \ell(f(x_0); y)$ but transports it back to the current noisy state x_t through the Jacobian of the pre-trained score function.

Practical remarks and limitations. The guidance scale w is often scheduled over time: applying strong guidance too early (at high noise) can destabilize sampling, while applying guidance too late may yield limited control. UG also typically relies on differentiable surrogates loss functions that are naturally discrete or non-differentiable.

Universal guidance generalizes classifier guidance by replacing the classifier-gradient term with the gradient of a generic objective. While this removes the need for a discrete label space, it inherits several limitations of classifier guidance.

First, UG still relies on an *external* network trained on labeled/paired data, and its gradients can be poorly calibrated on intermediate, off-distribution denoised estimates $\widehat{x}_0(x_t, t)$, especially at high noise levels. This mirrors the "noisy classifier" issue in classifier guidance and can lead to weak control early in sampling.

Second, UG introduces an additional scale-matching problem: the magnitude of $\nabla_{x_t} \ell(\widehat{x}_0(x_t, t); y)$ typically varies sharply with t and rarely matches the base score scale, so careful tuning of the guidance schedule is required.

These issues motivate *parameter-efficient adaptation* of the score network, where the condition is internalized into the denoiser via lightweight modules (adapters/LoRA/ControlNet) trained by denoising objectives.

2.5 Training-Based Methods III: Parameter-Efficient Adaptation

Classifier guidance and universal guidance modify pre-trained score function by adding an external gradient term, while keeping the diffusion network fixed. An alternative training-based approach is to *adapt the score network itself* to better incorporate a new condition.

A parameter-efficient template. Let $s_\theta(x, t)$ be a pre-trained score network, where θ denote the weight parameters. Note that now we are explicit on the trainable parameters in the score network. We introduce a lightweight module with new parameters ϕ and define a conditional score network

$$s_{\theta,\phi}(x, y, t) \approx \nabla_x \log p_t(x \mid y),$$

A standard training objective is conditional denoising score matching in (1.3):

$$\min_{\phi} \int_0^T \mathbb{E}_{(X_0,Y)\sim P_{\text{data}}} \mathbb{E}_{X\sim \mathsf{N}(e^{-t/2}X_0,(1-e^{-t})I)} \left\| s_{\theta,\phi}(X, Y, t) + \frac{X - e^{-t/2}X_0}{1 - e^{-t}} \right\|_2^2 \mathrm{d}t.$$

A critical idea is that θ—the pre-trained parameter—is frozen, only ϕ is optimized. There are many different design choices on the lightweight module and the trainable parameters ϕ.

Adapters. Adapters insert small residual blocks into the frozen pre-trained score network. Conceptually, if h_k is the hidden state at layer k, a typical adapter takes the form

$$h'_k = h_k + g_{\phi,k}(h_k, y, t),$$

where $g_{\phi,k}$ is a small network module. With this parameterization, we substitute h'_k into the conditional denoising score matching objective and optimize over ϕ. As particular examples in text-to-image generation, ControlNet [6] applies adapter to U-Net in stable diffusion as demonstrated in Figure 5.

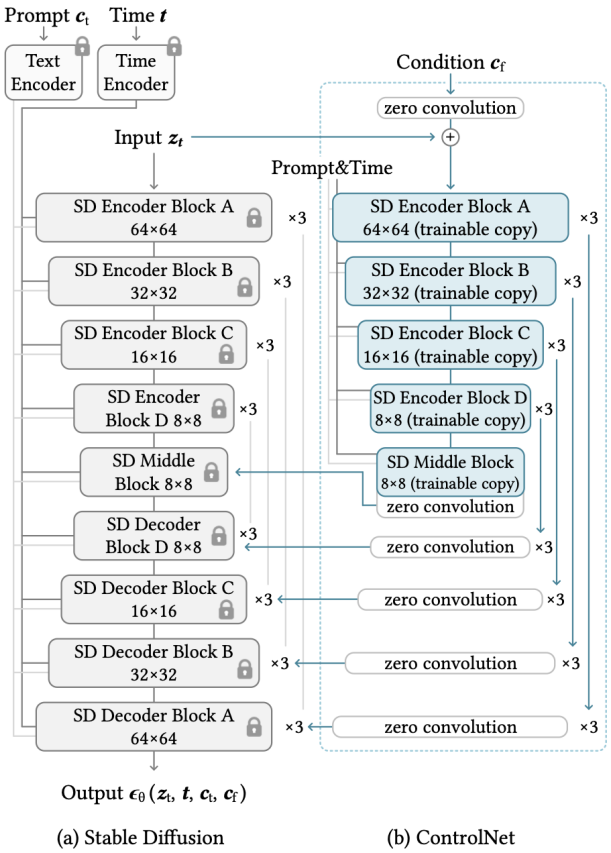

Figure 5: ControlNet for stable diffusion in [6].

We observe that the adaptation operates on the encoding side and the bottleneck of the U-Net, the decoding side is kept the same.

IP-adapter [5] trains an adapter module based on the cross-attention of image features. They innovatively decouples the cross-attention for image and text features. As reported, utilizing a 22M parameter adapter on stable diffusion v1.5 already leads to strong performance.

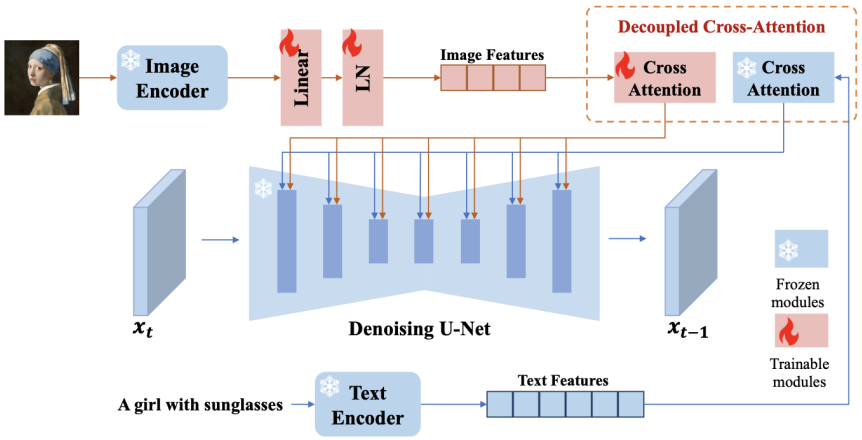

Figure 6: IP-adapter proposed in [5].

LoRA (Low-Rank Adaptation). LoRA [3] was originally proposed for fine-tuning large language models. The key idea is to parameterize weight updates in selected weight matrices as products of low-rank matrices. Let W be a weight matrix in the score network. We write its update as

$$W \mapsto W + \Delta W, \quad \text{with} \quad \Delta W = BA,$$

where matrices A and B are low-rank. In language model fine-tuning, the rank typically varies between 8 to 16. In this case, θ is all the original weight matrices W and ϕ encodes all the low-rank matrices A and B. During the post-training, only A, B matrices are optimized. You may try LoRA for diffusion model fine-tuning at `https://github.com/cloneofsimo/lora`.

Parameter-efficient adaptation addresses the limitation of guidance-only methods: instead of relying on an external gradient $\nabla \ell$, which may be miscalibrated under heavy noise, the score network learns a new module that adapts to new conditions, often reducing artifacts and improving stability.

2.6 Training-Based Methods IV: Reinforcement Learning (RL)

We leave a detailed discussion of RL-based post-training methods to **Chapter 6**. The reason is a bi-directional synergy between diffusion and RL: diffusion models can be *fine-tuned* using RL algorithms, and diffusion models can also *parameterize policies* in RL (diffusion policies) and be integrated with planning/control. Treating them together leads to a unified view of "generative modeling for decision making".

Roughly speaking, diffusion models generate samples following a Markov process. We may rewrite the sampling process as a Markov decision process, where the action is the score function. We

associate a reward function to the Markov decision process (usually a terminal reward) and cast the diffusion model fine-tuning as a policy learning/optimization problem.

2.7 Training-Free Methods I: Gradient-Based Guidance

We now turn to *training-free* post-training methods, which *do not update* any network parameters. They instead steer sampling by injecting an *external signal* into the sample generation dynamics.

The simplest case is that we can evaluate a differentiable reward r (as well as its gradient) on the clean sample. This can be viewed as a simplification of the universal guidance, where a loss function needs to be trained on labeled data. Given a noisy state x_t, we recall the clean data estimate $\widehat{x}_0(x_t, t)$. Then a training-free reward guided score function takes

$$s_{\text{reward}}(x_t, y, t) = s(x_t, t) + w \cdot \nabla_{x_t} r(\widehat{x}_0(x_t, t); y), \tag{2.5}$$

where s is the pre-trained score and w can be time-dependent. The reward guided score is *training-free* because r is treated as a fixed oracle, and the pre-trained diffusion network is unchanged.

2.8 Training-Free Methods II: Sampling-Based Corrections

Gradient-based guidance modifies the reverse-time drift directly. A different training-free family aims to sample (approximately) from a *well-defined target law* by *reweighting / resampling / correcting* samples produced by a pre-trained diffusion model. The central object is a *tilted* distribution.

2.8.1 Density Ratios, Tilted Distributions, and KL-Regularized Optimization

Let $P_{\text{data}}(\cdot)$ denote the original (unconditioned) data distribution. The target distribution is denoted as $P_{\text{data}}(\cdot \mid \mathcal{E})$, where $\mathcal{E}$ is an event of interest.

Measure transport viewpoint: density ratio as the transport factor. Under conditions, the Radon-Nykodim derivative

$$\frac{\mathrm{d}P_{\text{data}}(\cdot \mid \mathcal{E})}{\mathrm{d}P_{\text{data}}(\cdot)}$$

exists and realizes the *measure transport* from $P_{\text{data}}(\cdot)$ to $P_{\text{data}}(\cdot \mid \mathcal{E})$. In the case that densities $P_{\text{data}}(\cdot)$ and $P_{\text{data}}(\cdot \mid \mathcal{E})$ exist, the Radon-Nykodim derivative can be written as $p_{\text{data}}(\cdot \mid \mathcal{E})/p_{\text{data}}(\cdot)$. We denote

$$w(x) = p_{\text{data}}(x \mid \mathcal{E})/p_{\text{data}}(x).$$

Thus, sampling from $P_{\text{data}}(\cdot \mid \mathcal{E})$ is equivalent to sampling from a *tilted* distribution with density

$$w(x) \cdot p_{\text{data}}(x).$$

When $\mathcal{E}$ is defined by discrete labels, $w(x)$ is the familiar classifier likelihood. However, the sampling-based post-training methods do not rely on guidance to correct the score function. They aim to directly sample from the tilted distribution.

Optimization view: KL-regularized reward maximization. When a reward function r is defined on the clean data, we consider the KL-regularized objective

$$P^* \in \operatorname*{argmax}_{P \ll P_{\text{data}}} \mathbb{E}_{X\sim P}[r(X)] - \lambda \cdot \mathrm{KL}(P, P_{\text{data}}). \tag{2.6}$$

Equation (2.6) is of fundamental importance in soft (entropy-regularized) reinforcement learning algorithms. Suppose P_{data} has a density p_{data}. There exists a unique maximizer of (2.6), which is an exponentially tilted distribution with density

$$p^*(x) = \frac{1}{Z} p_{\text{data}}(x) \exp\big(r(x)/\lambda\big), \tag{2.7}$$

where the normalization $Z = \int p_{\text{data}}(x) \exp(r(x)/\lambda) \mathrm{d}x$. If we define $w(x) = Z^{-1} \exp(r(x)/\lambda)$, then the density p^* in (2.7) is written as

$$p^*(x) = w(x) \cdot p_{\text{data}}(x),$$

which resembles the same format in the measure transport viewpoint.

Verification of Equation (2.7). We consider a KL divergence between distribution P and P^*:

$$\begin{aligned}
\mathrm{KL}(P, P^*) &= \mathbb{E}_{X\sim P}\left[\log \frac{p(x)}{p^*(x)}\right] \\
&= \mathbb{E}_{X\sim P}\left[\log \frac{p(x)}{p_{\text{data}}(x)} - \log \frac{p^*(x)}{p_{\text{data}}(x)}\right] \\
&= \mathrm{KL}(P, P_{\text{data}}) - \mathbb{E}_{X\sim P}\left[\log\left(\frac{1}{Z}\exp(r(x)/\lambda)\right)\right] \\
&= \mathrm{KL}(P, P_{\text{data}}) - \frac{1}{\lambda}\mathbb{E}_{X\sim P}\left[\log r(x)\right] + \log Z.
\end{aligned}$$

Rearranging terms leads to

$$\mathbb{E}_{X\sim P}\left[\log r(x)\right] - \lambda \cdot \mathrm{KL}(P, P_{\text{data}}) = -\lambda \cdot \mathrm{KL}(P, P^*) + \lambda \log Z. \tag{2.8}$$

Note that the right-hand side is maximized if $P = P^*$. Therefore, we have shown that P^* is the maximizer of (2.6).

It is also worth mentioning that plugging $P = P^*$ into Equation (2.8) gives rise to the Donsker-Varadhan variational form,

$$\lambda \cdot \log \mathbb{E}_{X\sim P_{\text{data}}}[\exp(r(X)/\lambda)] = \sup_{P \ll P_{\text{data}}} \mathbb{E}_{X\sim P}[r(X)] - \lambda \cdot \mathrm{KL}(P, P_{\text{base}}).$$

□

2.8.2 Terminal Correction: Rejection Sampling

Suppose we can evaluate the weight $w(x_0)$ for clean data x_0, and there exists a constant $M < \infty$ such that $w(x_0) \le M$ for all x_0 in the support of P_{data}. Then we can sample from the tilted distribution via rejection sampling

1. Sample $x_0 \sim P_{\text{data}}$ using the pre-trained diffusion model.
2. Accept x_0 with probability $w(x_0)/M$; otherwise reject and resample.

This produces exact samples from the tilted distribution. In high dimension, however, M is typically astronomically large, making acceptance rates negligible.

A more fundamental difficulty is that the weighting function w is often hard to evaluate. From the measure transport viewpoint, w is the density ratio, however, we typically know neither $p_{\text{data}}(\cdot)$ nor $p_{\text{data}}(\cdot \mid \mathcal{E})$. From the optimization viewpoint, w involves a normalization constant Z, which is expensive to estimate. Therefore, importance sampling based methods can be applied, but are limited in their performance in most applications.

2.8.3 Intermediate Corrections: Diffusion-Time Resampling

Terminal reweighting uses only $w(x_0)$—a weighting function induced by the conditional distribution or the reward function. A more stable idea is to *distribute* the tilt along the reverse-time trajectory and resample multiple times.

We choose a decreasing time grid $T = t_K > t_{K-1} > \cdots > t_0 = 0$ along the backward sample generation process. Define a family of intermediate tilted densities

$$p^{\star}_{t_k}(x) = w_k(x) \cdot p_{t_k,\text{data}}(x),$$

where $p_{t_k,\text{data}}$ is the marginal density of the pre-trained diffusion model at time t_k and w_k is the weighting function at time t_k. Note that $p_{t_k,\text{data}}$ should be interpreted as $\widehat{p}_{t_k,\text{data}}$, since a pre-trained diffusion model can only approximate the data distribution. We omit the hat notation for simplicity. The weighting function w_k is often defined through a surrogate reward, e.g., based on the reward of an estimate clean data $r(\widehat{x}_0(x, t_k))$.

Sequential Monte-Carlo (SMC). SMC maintains N particles $\{x^{(i)}_{t_k}\}_{i=1}^N$ in each step. At time t_k, the following predict-reweight-resample procedure will be executed:

1. **Predict:** propagate each particle by one backward sampling step using the pre-trained diffusion model.
2. **Reweight:** compute weights using the tilted densities and weighting function w_k.
3. **Resample:** resample particles when the effective sample size is small.

As can be seen, SMC gradually steers the population toward higher-reward regions, instead of relying soly on the terminal data.

2.8.4 Doob's h-Transform

Sampling-based corrections can be pushed further: rather than correcting samples, one can modify the backward sample generation dynamics so that the process is (approximately) *already targeting* the conditioned/tilted distribution. A conceptual gold standard is Doob's h-transform.

A non-rigorous introduction of Doob's h-transform is to introduce a weighting function for any backward diffusion time t:

$$h(t, x) = \mathbb{E}\big[\psi(X_0) \mid X_t = x\big], \tag{2.9}$$

where the conditional expectation is taken under the pre-trained backward process. Here, ψ is any measurable function, e.g., $\psi(X_0) = \exp(r(X_0)/\lambda)$ reproduces the weighting function in the KL-regularized optimization. In principle, Doob's h-transform is a special instance of the intermediate correction, where the weighting function w_k is $h(t_k, \cdot)$. Moreover, the terminal distribution of applying Doob's h-function is precisely

$$p_{\text{Doob}}(x) \propto \psi(x) \cdot p_{\text{data}}(x).$$

A key property of Doob's h-transform is that it introduces a correction term to the pre-trained score function:

$$s_{\text{Doob}}(x, t) = s(x, t) + \nabla_x \log h(t, x). \tag{2.10}$$

The extra term $\nabla \log h$ is a *principled, time-consistent* correction ensuring that the terminal distribution is tilted by ψ. This can also be viewed as an exact analogue of guidance.

The practical challenge of applying Doob's h-transform is that $h(t, x)$ is generally intractable, since it is a conditional expectation over future denoising trajectories. Doob-style methods therefore rely on approximations.

References

[1] Arpit Bansal, Hong-Min Chu, Avi Schwarzschild, Soumyadip Sengupta, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Universal guidance for diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 843–852, 2023.

[2] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[3] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3, 2022.

[4] Masatoshi Uehara, Yulai Zhao, Kevin Black, Ehsan Hajiramezanali, Gabriele Scalia, Nathaniel Lee Diamant, Alex M Tseng, Tommaso Biancalani, and Sergey Levine. Fine-tuning of continuous-time diffusion models as entropy-regularized control. *arXiv preprint arXiv:2402.15194*, 2024.

[5] Hu Ye, Jun Zhang, Sibo Liu, Xiao Han, and Wei Yang. Ip-adapter: Text compatible image prompt adapter for text-to-image diffusion models. *arXiv preprint arXiv:2308.06721*, 2023.

[6] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3836–3847, 2023.