



UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
ESCUELA DE POSTGRADO Y EDUCACIÓN CONTINUA

**EXTENDING REINFORCEMENT LEARNING TECHNIQUES FOR
DIFFUSION MODELS**

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIA DE DATOS

CRISTÓBAL PATRICIO ALCÁZAR CARRASCO

PROFESOR GUÍA:
FELIPE TOBAR HENRÍQUEZ

MIEMBROS DE LA COMISIÓN:
JAVIER RUIZ DEL SOLAR
JOAQUÍN FONTBONA TORRES

Este trabajo ha sido parcialmente financiado por:
GOOGLE & FONDECYT REGULAR 1210606

SANTIAGO DE CHILE
2024

RESUMEN DE LA TESIS PARA OPTAR
AL TÍTULO DE MAGÍSTER EN CIENCIAS
DE DATOS
POR: CRISTÓBAL PATRICIO ALCÁZAR CARRASCO
FECHA: 2024
PROF. GUÍA: FELIPE TOBAR HENRÍQUEZ

EXTENSIÓN DE TÉCNICAS DE APRENDIZAJE POR REFUERZO PARA MODELOS DE DIFUSIÓN

El Aprendizaje por Refuerzo (RL) se ha convertido en una herramienta crucial para alinear modelos generativos complejos, superando las limitaciones de los métodos de aprendizaje supervisado tradicionales. Su capacidad para optimizar recompensas arbitrarias, incluyendo funciones escalares no diferenciables o retroalimentación humana, es especialmente útil para modelos a gran escala como los Modelos de Lenguaje Grandes (LLMs) y los modelos de difusión. Esta tesis investiga la aplicación de técnicas de RL a modelos de difusión preentrenados, utilizando métodos de gradiente de políticas para adaptar estos modelos a nuevas tareas. Explora cómo los modelos de difusión pueden considerarse agentes que generan muestras para maximizar atributos específicos, como la calidad estética o la compresibilidad, y realiza un análisis empírico de las señales de recompensa a lo largo de las trayectorias de muestra. El trabajo incluye la implementación de algoritmos de optimización de políticas de vanguardia (DDPO) e integra la retroalimentación humana para proporcionar herramientas y perspectivas prácticas. Esta investigación ofrece una ruta para comprender el uso de RL en el ajuste de modelos de difusión preentrenados y proporciona ideas para posibles adaptaciones futuras.

THESIS SUMMARY TO QUALIFY FOR
THE DEGREE OF MASTER OF SCIENCE
IN DATA SCIENCE
BY: CRISTÓBAL PATRICIO ALCÁZAR CARRASCO
DATE: 2024
ADVISOR: FELIPE TOBAR HENRÍQUEZ

EXTENDING REINFORCEMENT LEARNING TECHNIQUES FOR DIFFUSION MODELS

Reinforcement Learning (RL) has become a pivotal tool for aligning complex generative models, addressing the limitations of traditional supervised learning methods. Its capability to optimize arbitrary rewards, including non-differentiable scalar functions or human feedback, is particularly useful for large-scale models such as Large Language Models (LLMs) and diffusion models. This thesis investigates the application of RL techniques to pretrained diffusion models, employing policy gradient methods to adapt these models for new tasks. It explores how diffusion models can be viewed as agents generating samples to maximize specific attributes—such as aesthetic quality or compressibility—and conducts an empirical analysis of reward signals over sample trajectories. The work includes the implementation of state-of-the-art policy optimization algorithms (DDPO) and integrates human feedback to provide practical tools and insights. This research offers a pathway for understanding the use of RL in finetuning pretrained diffusion models and provides insights for potential future adaptations.

A mi familia

Table of Content

1. Introduction	1
1.1. Related Work	3
1.2. Contributions and Outline	3
2. Diffusion Models	5
2.1. Denoising Diffusion Probabilistic Models	5
2.2. Recursive Reparameterization Trick	7
2.3. Optimization	9
2.3.1. Variational Lower Bound	10
2.3.2. Denoising Matching Term	11
2.4. Score-based generative models	13
2.5. Sampling	14
2.6. Conditioning the model	15
2.6.1. Classifier Guidance (CG)	15
2.6.2. Classifier Free Guidance (CFG)	17
2.7. Denoising Diffusion Implicit Models	18
2.8. Summary	21
3. Reinforcement Learning	22
3.1. The Framework for Learning to Act	22
3.2. Policy Optimization	24
3.2.1. Learning the Policy	25
3.2.2. Gradient Estimation via Score Function	25
3.3. Vanilla Policy Gradient, aka REINFORCE	27
3.4. Actor-Critic Methods	31
3.5. Improving Sample Efficiency: Behavior and Target Policies	32
3.6. Trust Region and Proximal Policy Optimization	33
3.7. Reinforcement Learning From Human Feedback	33
3.8. Summary	35
4. Extending Reinforcement Learning in Diffusion Models	37
4.1. Diffusion Model as Sequential Decision-making Process	37
4.2. Empirical Analysis on Reward Trajectory Dynamics	41
4.3. Experiments	44
5. Results on Reward Finetuning using DDPO	46
5.1. Reward Finetuning on Face Generation	47
5.1.1. JPEG Compressibility	48

5.1.2. JPEG Incompressibility	49
5.1.3. Aesthetic Quality	51
5.1.4. OVER50	54
5.2. Beyond Face Generation	56
5.3. Discussion & Limitations	60
5.4. Future Work	62
6. Conclusion	63
Bibliography	65
Annex	70
A. Implementation details	70
B. Additional Samples: Celebrity faces by DDPO	73
C. Additional Transitions: from DDPM to DDPO	77
D. Additional Samples: Church images by DDPO	85

List of Tables

- 5.1. **Mean and standard error for each downstream task across two pre-trained models.** All samples were generated using the same initial noise to ensure a fair comparison. **Baseline** refers to the generative capabilities of the pretrained model, as represented by \mathcal{S}_0 (see Section 4.2). **DDPO** displays results from the finetuned models using DDPO with importance sampling (see Section 4.1). 47
- A.1. **Experiment details** with corresponding model checkpoints on Hugging Face and experiment dashboards on Weight & Biases, including logging information. Multiple runs indicate that the experiment continued training from the previous run, using the last saved checkpoint. 70
- A.2. Hyperparameters for finetuning [google/ddpm-celebahq-256](#) on JPEG Compressibility, Incompressibility, and Aesthetic Quality tasks using DDPO. 72

List of Figures

- 1.1. **Iterative Process of Finetuning a Diffusion Model Using Reinforcement Learning.** The process begins with (i) collecting a dataset of generated samples using a pretrained diffusion model, represented by the “Generate Samples” block. Next, (ii) these samples are evaluated by a reward model to obtain a reward signal, as shown in the “Reward Model” block. This reward information is then used to (iii) fit the model by optimizing the expected return of the samples, using methods like Monte Carlo gradient estimation, depicted in the “Fit a Model” block. Finally, (iv) the optimization improves the policy, aligning the diffusion model’s output with higher expected rewards, as indicated in the “Improve the Policy” block. This cycle iterates, refining the model based on feedback. When the reward model is based on human preferences, this approach is known as Reinforcement Learning from Human Feedback (RLHF). 2
- 2.1. **Example of a forward transition.** The image of Grogu (Baby Yoda) on the left is gradually degraded by Gaussian noise throughout the diffusion process (middle), ultimately resulting in isotropic Gaussian noise at the final stage (right). 5
- 2.2. **Noise scheduler used in DDPM [15]. Left:** The variance scheduler increase linearly from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$, over the forward pass of $T = 1000$ timesteps, progressively corrupting the data x_0 . **Middle:** The noiseless part of x_t is scale by a function of the variance, $\sqrt{1 - \beta_t}$. **Right:** In DDPM, $\alpha_t = 1 - \beta_t$ is defined. Given the addition of independent Gaussian noise, the data structure factor $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$ to allows us to scale x_0 and obtain x_t in just one evaluation (see Section 2.2). 6
- 2.3. **The reparameterization trick enabling backpropagation. Left:** A stochastic node associated with z sampled from $q(z | \phi, x)$, where direct backpropagation is not possible. **Right:** By reparameterizing z as a linear function $g(\phi, x, \epsilon)$, the stochasticity is removed, allowing effective backpropagation of the loss function f with respect to ϕ and x . **Source:** Variational Auto-Encoders and Extension, NeurIPS Workshop, by Kingma Diederik (2015). 8
- 2.4. **Denoising matching term in action. Left:** x_T is a pure Gaussian noise. **Middle:** Transition from a noisy intermediate state to a less noisy one; the denoising matching term forces $p_\theta(x_{t-1} | x_t)$ to be similar to the posterior forward kernel $q(x_{t-1} | x_t)$, making comparable and provide feedback to update the parameters μ_θ and Σ_θ . **Right:** x_0 the input image during training. 12
- 2.5. **CLIP overview.** Text-to-image joint embedding space. **Source:** Learning Transferable Visual Models From Natural Language Supervision, by Alec Radford et al. (2021) [42]. 16

2.6.	CLIP classifier guidance for image generation controlled via prompting. The first row shows four samples generated by the pretrained model google/ddpm-celebahq-256 . The second row shows the same initial noise used to generate the previous images but guided by the CLIP classifier with a guidance scale of $\gamma = 20$, using the prompt “old, senior, oldster, elderly, golden-ager” to transform the subjects into older individuals. Notable changes include the appearance of wrinkler and thinner hair, but artefacts also emerge, such as in the first image of the second row where glasses distort the facial structure. The third row illustrates a failure mode, where a much higher guidance scale ($\gamma = 2000$) results in heavily distorted images.	17
2.7.	DDIM non-Markovian forward process. Left: Illustration of accelerated generation skipping the uneven intermediate steps, the denoiser network $\epsilon_\theta^{(t)}$ predicts the amount of noise added to x_t from step $t - 2$, instead t . Right: the non-Markovian graphical model.	18
2.8.	DDIM inversion example using 50 inference steps. A Pedro Pascal photo to estimate the initial noise \tilde{x}_T using DDIM with the pretrained model google/ddpm-celebahq-256 on the celebrity faces dataset CelebaHQ. Then, the input is reconstructed using the estimated noise as starter point. Below images are different results skipping the first n inference steps of the denoising process.	20
3.1.	Left: A loop representation of a Markov Decision Process (MDP). Right: An unrolled MDP depicting an episodic case with a finite horizon T and a parameterized policy π_θ	23
3.2.	Illustration of a taxonomy of model-free RL algorithms. Source: Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs by John, Schulman (2016) [48].	24
3.3.	Illustration of three simulated trajectories , denoted as $\{\tau^{(i)}\}$ where $i = (1, 2, 3)$, traversing the parametric space $\theta \in \mathbb{R}^2$ under the policy π_θ . Each trajectory is marked with a colored symbol (cross, check) representing its <i>goodness</i> based on the reward function $R(\tau^{(i)})$. Source: Policy Gradients Lecture, Deep Reinforcement Learning Course by Sergey Levine.	29
3.4.	Left: A standard RL settings. Right: A RLHF setting considering reward modeling. Source: A Survey of Reinforcement Learning from Human Feedback (Kaufmann et al., 2024) [23]. Notice how the reward model is decoupled from the environment and the relation highlighted between the reward model and an oracle (i.e. labeler) who provides a label to a given query.	34
4.1.	Equivalence of the backward process of a diffusion model as a sequential decision-making process. The initial state distribution of this MDP corresponds to an isotropic Gaussian, $\rho_0(s_0) \sim \mathcal{N}(0, I)$, where we assign the noise instance to the initial state $s_0 := x_T$. The agent follows a sequence of decisions a_t determined by the policy $\pi_\theta(a_t s_t) := p_\theta(x_{T-t-1} x_{T-t})$, moving from a noisy state x_{T-t} to a less noisy one x_{T-t-1} until it reaches to the sample x_0 , illustrated as the terminal state s_T in the diagram. This process together with the sample x_0 generates the whole denoising trajectory $\tau = \{x_T, x_{T-1}, x_{T-2}, \dots, x_0\}$, which is associated with a reward. In the case of DDPO, the reward is considered only for the final sample $r(x_0)$	38

4.2.	Reward finetuning diagram using DDPO. The finetuning process occurs in two stages: (i) collecting a dataset \mathcal{D}^{π_θ} of samples by using the diffusion model as policy rollouts, and (ii) using \mathcal{D}^{π_θ} to estimate the gradient of the parameters $\nabla_{\theta}\mathcal{J}$, with objectives such as DDPO _{SF} , and applying a first-order optimization algorithm (e.g., gradient ascent) to update the model. Given that the dataset originates from a non-stationary distribution (changing policy), if we want to reuse the samples $\mathcal{D}^{\pi_{\theta_{\text{old}}}}$ to adjust the parameters more than once, we need to use the DDPO _{IS} objective.	41
4.3.	Visualizing reward signal during sample trajectories. Left: Evolution of the aesthetic score as reward signal over the six states $x_{\tilde{t}}$, summarizing each of the 1000 trajectories (\uparrow better). Right: Image size in kbs after JPEG compression, providing another form of reward signal for the same set of trajectories (\downarrow better). Top: Rewards computed over the noisy intermediate states $x_{\tilde{t}}$. Bottom: Rewards computed over the denoised states $\tilde{x}_{\tilde{t}}$	43
4.4.	Denoised sample trajectories. Each row, from top-to-bottom , represents the best (6.22) and worst (3.86) aesthetic scores, and the highest (34.38) and lowest (4.07) filesizes in kilobytes after JPEG compression for the final samples x_0 in \mathcal{S}_o . Each column, from left-to-right , summarizes the states for each denoised sample’s trajectory using Equation 4.5. The rows correspond to the trajectories highlighted in the green and red lines of Figure 4.3.	44
5.1.	Qualitative comparison of the effects of DDPO finetuning versus the pretrained model. The top-left panel shows samples from the DDPM pretrained model google/ddpm-celebahq-256 . The other panels display samples generated from the same initial noise, but using models finetuned with DDPO for different reward functions: aesthetic quality (top right), JPEG compressibility (bottom left), and incompressibility (bottom right). Additional samples are provided in Appendix B.	48
5.2.	Emergent effects in face generation using JPEG compressibility as a reward. Each panel compares a pretrained model sample (left) with its finetuned version (right). Notable changes include melanotropism, realistic facial expressions, and altered gender presentation due to hair detail loss and shading effects.	49
5.3.	Learning curves from DDPO finetuning on face generation tasks. The evolution of the mean reward (black line) and the reward distribution (hex-bin) are shown for each <i>downstream task</i> . The reward estimates were computed in each step using 100 samples from the model. The top-left panel shows the learning curve for the LAION aesthetic quality reward (aesthetic score), the top-right panel for the OVER50 reward (sum of logits for relevant classes), the bottom-left panel for JPEG compressibility (negative file size in kB), and the bottom-right for JPEG incompressibility (file size in kB).	50
5.4.	Sample transformation when optimizing the pretrained google/ddpm-celebahq-256 model using JPEG compressibility as the reward function. The top-left image shows a sample generated by the pretrained model. Moving from left to right and top to bottom, the sample is regenerated from the same initial noise after each update of the model’s parameters. The final result of the finetuning process is shown in the bottom-right image.	51

5.5.	Emergent effects in face generation using JPEG incompressibility as a reward. Each panel compares a pretrained model sample (left) with its finetuned version (right). Notable changes include increased hair volume, hair definition, skin tone lightening, and overall increased illumination with reduced shadows.	52
5.6.	Sample transformation when optimizing the pretrained <code>google/ddpm-celebahq-256</code> model using JPEG incompressibility as the reward function. The top-left image shows a sample generated by the pretrained model. Moving from left to right and top to bottom, the sample is regenerated from the same initial noise after each update of the model’s parameters. The final result of the finetuning process is shown in the bottom-right image.	53
5.7.	Aesthetic quality training dynamics. Left: The learning rate scheduler with a linear warm-up reaching a peak learning rate of 3.64×10^{-5} at the first quarter of training, followed by a half-cosine decay for the remaining three-quarters. Right: Mean aesthetic score on the evaluation set during training. The yellow line corresponds to training with the learning rate schedule described on the left, while the other lines represent training with fixed lower learning rates of 7×10^{-8} and 9×10^{-8} . Aesthetic quality requires a more complex training dynamic to achieve higher rewards.	54
5.8.	Emergent effects in face generation using LAION aesthetic predictor as a reward. Each panel compares a pretrained model sample (left) with its finetuned version (right). Notable changes are increased prevalence of female faces, a younger appearance, warmer tones, and more intense gazes, reflecting aesthetic preferences learned by the LAION predictor.	55
5.9.	Sample transformation when optimizing the pretrained <code>google/ddpm-celebahq-256</code> model using LAION aesthetic score as the reward function. The top-left image shows a sample generated by the pretrained model. Moving from left to right and top to bottom, the sample is regenerated from the same initial noise after each update of the model’s parameters. The final result of the finetuning process is shown in the bottom-right image.	56
5.10.	Face age distribution using the ViT age classifier [25] prediction classes on DDPM samples (left) and DDPO finetuned model samples (right). Finetuning with DDPO using the OVER50 reward function, which maximizes the logits sum for age classes 50–59, 60–69, and ≥ 70 , increases the proportion of faces over 50 years old from 6.1 % in the baseline to 78.7 % in the finetuned samples.	57
5.11.	Sample transformation when optimizing the pretrained <code>google/ddpm-celebahq-256</code> model using OVER50 as the reward function. The top-left image shows a sample generated by the pretrained model. Moving from left to right and top to bottom, the sample is regenerated from the same initial noise after each update of the model’s parameters. The final result of the finetuning process is shown in the bottom-right image.	58

5.12.	Qualitative comparison of the effects of DDPO finetuning versus the pretrained model. The top-left panel shows samples from the DDPM pretrained model google/ddpm-church-256 . The other panels display samples generated from the same initial noise, but using models finetuned with DDPO for different reward functions: aesthetic quality (top right), JPEG compressibility (bottom left), and incompressibility (bottom right). Additional samples are provided in Appendix D	59
5.13.	Sample transformation when optimizing the pretrained google/ddpm-church-256 model using the reward functions: JPEG compressibility (top) and incompressibility (middle), and LAION aesthetic score (bottom). First column shows the sample generated by the pretrained model. Moving from left to right the sample is regenerated from the same initial noise after each update of the model’s parameters. Final result of the finetuning process is shown in the last column.	60
5.14.	A CLIP feature embedding space coexisting DDPM and DDPO Samples. Both sets of samples were generated from the same initial noise. Notably, samples optimized for aesthetic quality cluster near the highest aesthetic score sample in the DDPM set (Figure 4.4), illustrating a clear <i>mode collapse effect</i>	61
A.1.	Table for Monitoring Development Set Progress. This table compares images generated by the pretrained model with those refined using DDPO. It displays rewards for each image, their differences, and a graph illustrating reward curves throughout the diffusion model’s generation process.	71
B.1.	256x256 celebrity face samples generated by the pretrained model google/ddpm-celebahq-256	73
B.2.	256x256 celebrity face samples generated by the DDPO finetuned model alkzar90/ddpo-compressibility-celebahq-256 , optimized for JPEG compressibility.	74
B.3.	256x256 celebrity face samples generated by the DDPO finetuned model alkzar90/ddpo-incompressibility-celebahq-256 , optimized for JPEG incompressibility.	75
B.4.	256x256 celebrity face samples generated by the DDPO finetuned model alkzar90/ddpo-aesthetic-celebahq-256 , optimized for aesthetic quality.	76
C.1.	Example 1 of JPEG compressibility transformation during model updates, starting with a pretrained DDPM model and optimized with DDPO to maximize image file size reduction after JPEG compression.	77
C.2.	Example 2 of JPEG compressibility transformation during model updates, starting with a pretrained DDPM model and optimized with DDPO to maximize image file size reduction after JPEG compression.	78
C.3.	Example 1 of JPEG incompressibility transformation during model updates, starting with a pretrained DDPM model and optimized with DDPO to maximize image file size after JPEG compression.	79
C.4.	Example 2 of JPEG incompressibility transformation during model updates, starting with a pretrained DDPM model and optimized with DDPO to maximize image file size after JPEG compression.	80
C.5.	Example 1 of aesthetic quality transformation during model updates, starting with a pretrained DDPM model and optimized with DDPO to maximize aesthetic quality.	81

- C.6. **Example 2 of aesthetic quality transformation during model updates**, starting with a pretrained DDPM model and optimized with DDPO to maximize aesthetic quality. 82
- C.7. **Example 1 of OVER50 transformation during model updates**, starting with a pretrained DDPM model and optimized with DDPO to maximize the sum of logits for classes ≥ 50 years old using the ViT Age classifier. 83
- C.8. **Example 2 of OVER50 transformation during model updates**, starting with a pretrained DDPM model and optimized with DDPO to maximize the sum of logits for classes ≥ 50 years old using the ViT Age classifier. 84
- D.1. 256 \times 256 church samples generated by the pretrained model [google/ddpm-church-256](#). 85
- D.2. 256 \times 256 church samples generated by the DDPO finetuned model [alkzar90/ddpo-compressibility-church-256](#), optimized by JPEG compressibility. 86
- D.3. 256 \times 256 church samples generated by the DDPO finetuned model [alkzar90/ddpo-incompressibility-church-256](#), optimized for JPEG incompressibility. 87
- D.4. 256 \times 256 church samples generated by the DDPO finetuned model [alkzar90/ddpo-aesthetic-church-256](#), optimized for aesthetic quality. 88

1

Introduction

“You take the red pill, you stay in Wonderland, and I show you how deep the rabbit hole goes. Remember, all I’m offering is the truth. Nothing more.”

– Morpheus, *The Matrix*

Reinforcement learning (RL) has shown the capacity to orchestrate or align highly complex generative models, which often proves impossible using supervised learning objectives such as matching distributions or incorporating specific goals into loss functions [1–5]. Beyond merely circumventing these challenges, RL should be regarded as a transformative user-model interface endeavor, offering advanced mechanisms for users to explore and manipulate tasks within generative models from a human-computer interaction (HCI) perspective [6–8]. Despite the computational demands and implementation complexities associated with RL, it provides unparalleled flexibility by optimizing arbitrary expected rewards, where the reward can be a non-differentiable scalar function or a model trained using human feedback. This becomes particularly relevant in the context of large models like LLMs [9] and diffusion models [10–14], which have driven significant research towards improving sample efficiency and prioritizing inference. These advancements make the exploration of RL agents atop these generative models especially compelling.

An agent in the context of RL is a decision-making entity that interacts with an environment, learning to take actions that maximize cumulative reward. Conversely, a diffusion model excels at learning highly complex distributions, enabling it to generate samples from the learned distribution. This allows us to interpret the diffusion model as an agent that interacts with the environment to generate samples, with the goal of maximizing a desired attribute, such as compressibility or aesthetic quality, as measured by a reward function. Figure 1.1 illustrates this process: it begins with generating a set of samples using a pretrained diffusion model, followed by obtaining rewards that measure the alignment of each generation with the task goal. The next step involves fitting a model by estimating the gradients of the expected reward of the samples with respect to the diffusion model parameters. Finally, these estimates are used to update the model parameters, directing the diffusion model to generate samples with higher expected returns. When executed correctly, this iterative process results in a generative model that is aligned with the goal encoded in the reward signal.

In this thesis, we explore the application of reinforcement learning techniques to adapt pretrained diffusion models for new tasks. We provide the necessary background to understand the intersection of diffusion models and policy gradient methods from the field of reinforce-

ment learning, aiming to serve as a comprehensive guide for those looking to contribute to this area. We then implement the iterative process depicted in Figure 1.1, based on the work *Training Diffusion Models with Reinforcement Learning* (Black, 2023 [1]), which introduces policy optimization algorithms in the realm of diffusion models (DDPO).

Using the pretrained models [google/ddpm-celebahq-256](#) and [google/ddpm-church-256](#), both trained with a foundational diffusion model approach (DDPM) [15], simplifies the understanding of the core components in the finetuning process with reinforcement learning. These models are significantly smaller, with an 8x reduction in parameters compared to the stable diffusion v1.4 model used in the DDPO experimentation paper [1]. The reduction in model size not only lowers the VRAM requirements, making the models easier to run, but also streamlines the experimental setup. Unlike stable diffusion, which operates with text-to-image capabilities and requires a variational autoencoder (VAE) for latent space processing, these DDPM models operate directly in pixel space. By eliminating the need for latent space decoding, the task of exploring intermediate states becomes more straightforward, allowing researchers to focus on refining the model across the entire generation process rather than solely on the final output. This capability facilitates a more comprehensive approach to extending rewards throughout the generation. The code, model checkpoints, and training dynamics are provided with this work (see Appendix A).

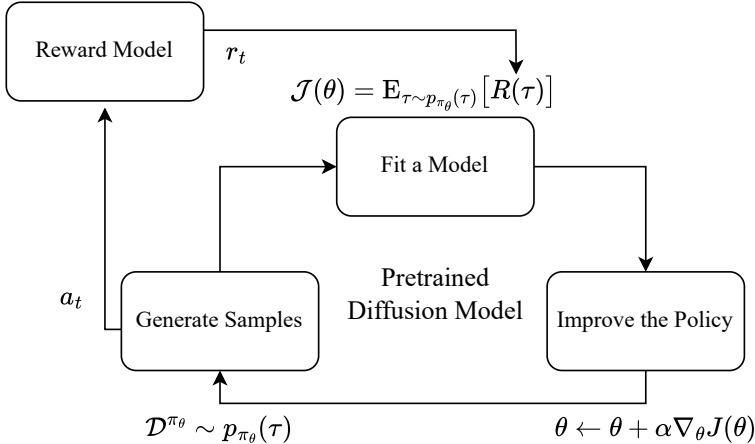


Figure 1.1: Iterative Process of Finetuning a Diffusion Model Using Reinforcement Learning. The process begins with (i) collecting a dataset of generated samples using a pretrained diffusion model, represented by the “Generate Samples” block. Next, (ii) these samples are evaluated by a reward model to obtain a reward signal, as shown in the “Reward Model” block. This reward information is then used to (iii) fit the model by optimizing the expected return of the samples, using methods like Monte Carlo gradient estimation, depicted in the “Fit a Model” block. Finally, (iv) the optimization improves the policy, aligning the diffusion model’s output with higher expected rewards, as indicated in the “Improve the Policy” block. This cycle iterates, refining the model based on feedback. When the reward model is based on human preferences, this approach is known as Reinforcement Learning from Human Feedback (RLHF).

1.1. Related Work

Diffusion Models. Diffusion models and score-based models represent a significant advancement in the field of generative models. These models learn a distribution $p(x)$ from a dataset \mathcal{D} , enabling evaluation and sampling of complex data types such as images or audio. They begin with a simple prior distribution (e.g., isotropic-Gaussian) and iteratively transform it into the target distribution through a denoising process. Recent advancements have focused on making the sampling process more efficient [12, 16, 17]. The progress in diffusion models has led to impressive results in tasks like text-conditional image generation [18, 19], super-resolution, in-painting, style transfer, and combining different data modalities.

Controlling Diffusion Models. Controlling diffusion models for new tasks is a challenging and evolving area of research. Given the large cost and resources required to train generative models from scratch, adapting pretrained diffusion models is commonplace. This adaptation allows the models to learn new concepts, such as specific objects or scenes, with a reasonable amount of data. Techniques like using a small set of images to teach a model new concepts without losing its diversity [20], and textual inversion to embed new concepts [21], have shown promising results. Additionally, ControlNet provides advanced control over the generation process, enabling inputs like Canny edges [22]. However, many downstream tasks cannot be easily expressed through text prompts or loss functions due to their context-dependent or subjective nature.

Reinforcement Learning from Human Feedback (RLHF). Recently, attention to using human feedback in reinforcement learning has increased [23]. The core idea is to capture human feedback into a reward model that can be used to train the policy that dictates the agent’s behavior. This approach allows different types of feedback, such as binary, continuous, or even more complex signals. Instead of designing the reward function—*or using feature engineering*—we can give the agent access to the reward model to obtain the necessary feedback for trajectories and optimize its behavior to learn the task.

1.2. Contributions and Outline

This thesis investigates the **hypothesis** that reinforcement learning, particularly through policy optimization algorithms, can effectively finetune pretrained diffusion models for new tasks by optimizing reward signals. The primary focus is on guiding the sampling process within these models to better align with task-specific goals, thereby enhancing performance through optimized rewards.

The **general objective** is to explore and validate the use of RL for finetuning pretrained diffusion models, aiming to improve task performance by optimizing specific reward signals and integrating human feedback into the learning process. The **specific objectives** are:

1. Provide foundational background on the intersection of diffusion models and reinforcement learning, emphasizing key concepts such as policy gradient methods and associated challenges.

2. Understand the intersection between diffusion models and reinforcement learning, particularly in framing the diffusion model as a problem to be solved by an RL agent using policy gradient methods, as discussed in the work *Training Diffusion Models with Reinforcement Learning* [1].
3. Conduct empirical analyses of reward signals, evaluating their impact on the intermediate steps of the diffusion process and deriving insights for optimizing generative outcomes.
4. Implement and adapt the Denoising Diffusion Policy Optimization (DDPO [1]) algorithm for smaller pretrained DDPM models. Validate the effectiveness of RL-based reward finetuning using base models such as [google/ddpm-celebahq-256](#) and [google/ddpm-church-256](#), which generate RGB images of human faces and church scenes at a resolution of 256×256 pixels, respectively.
5. Assess the adaptability of RL techniques across different diffusion models and tasks, focusing on various generative objectives. These include JPEG compressibility, aimed at generating images with smaller file sizes; incompressibility, which targets larger file sizes; and improvements in aesthetic quality using the LAION aesthetic predictor [24]. Additionally, evaluate a custom-designed OVER50 reward function based on a face age classifier [25], which aims to increase the likelihood of generating images of individuals over 50 years old.

These specific objectives guide the structure of the thesis. **Chapter 2** introduces the fundamentals of diffusion models, including their formulation, training objectives, and methods for guiding and speeding up sample generation. **Chapter 3** provides an overview of the reinforcement learning field, focusing on Markov Decision Processes (MDP) and Policy Optimization algorithms, which are essential for understanding how agents can learn optimal actions within an environment.

Building on this foundation, **Chapter 4** explores the intersection of diffusion models and reinforcement learning, presenting the formulation of the RL objective Denoising Diffusion Policy Optimization (DDPO) [1]. Next, with a clear focus on maximizing the reward using DDPO, we conduct an empirical study of the reward signal throughout the diffusion process to generate samples. This chapter also illustrates the diffusion process as an MDP, where we aim to maximize the agent’s performance in generating samples with a particular attribute.

Equipped with this knowledge, **Chapter 5** focuses on the implementation of DDPO, applying the algorithm to smaller generative models and reproducing some of the experiments from the work that introduced DDPO. This chapter also explores the design and implementation of a new reward function, addressing challenges and opportunities in applying RL to generative tasks. Finally, **Chapter 6** presents the conclusions drawn from this work.

2

Diffusion Models

In this chapter, we introduce the Diffusion Model, a family of generative models that have proven to be a valuable framework for novel image generation, text-to-image, text-to-video, and practical applications such as molecular graph modeling and medical image reconstruction [15, 16, 26–30]. Proving to be a useful tool and keep this kind of model to be an active research line in the last few years [31].

We will review the formulation proposed in the work titled *Denoising Diffusion Probabilistic Models* [15], or DDPM for short. The primary motivation to build on this foundational paper is to understand how the pretrained model in this work was trained. In addition, DDPM serves as a framework to understand subsequent improvements and enhancements without significant modifications to the primary ingredients of the recipe. Furthermore, DDPM has a deep connection with theoretical work of score-based generative models [32–34] and Variational Autoencoders [35].

2.1. Denoising Diffusion Probabilistic Models

The key idea in Denoising Diffusion Probabilistic Models (DDPM) is to learn a mapping from a complex data distribution to a simple prior distribution, such as a Gaussian. This is achieved by successively corrupting the data with noise, transforming observations of the complex distribution into observations of the simple one. Concurrently, a model is trained to denoise the sequence of noisy observations and recover the original data.

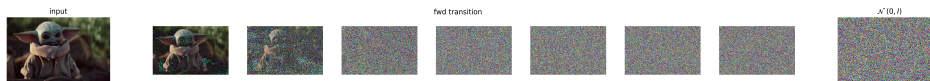


Figure 2.1: **Example of a forward transition.** The image of Grogu (Baby Yoda) on the **left** is gradually degraded by Gaussian noise throughout the diffusion process (**middle**), ultimately resulting in isotropic Gaussian noise at the final stage (**right**).

Concretely, a forward process which starts from the raw image x_0 creates a sequence of intermediate steps between x_1 and x_{T-1} , also known as latent states, which are noise perturbations in some degree of the original image’s structure as shown in Figure 2.1. At the end of the sequence T —if the sequence is infinitely large—the image is ultimately converted into an

isotropic Gaussian noise $p(x_T) \sim \mathcal{N}(0, \mathbf{I})^1$.

The DDPM authors model this forward process $q(x_1 \dots x_T | x_0)$ as a Markov chain, and it follows a transition Gaussian kernel without learnable parameters,

$$q(\mathbf{x}_{1:T} | \mathbf{x}_0) = \prod_{t=1}^T q(\mathbf{x}_t | \mathbf{x}_{t-1}) \quad (2.1)$$

$$q(\mathbf{x}_t | \mathbf{x}_{t-1}) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{1 - \beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I}\right). \quad (2.2)$$

The process of corrupting data with noise during the progression of the chain is managed by a time-dependent function that determines the variance $\sigma(t) = \beta_t$ at each timestep t . This function is known as the noise (or variance) scheduler, and it is a key component in diffusion models. On the left side of Figure 2.2, the noise scheduler used in DDPM is shown, where the variance increases linearly from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$, over the course of $T = 1000$ timesteps during the forward pass, progressively corrupting the input \mathbf{x}_0 until it fully turns into noise, represented by \mathbf{x}_T .

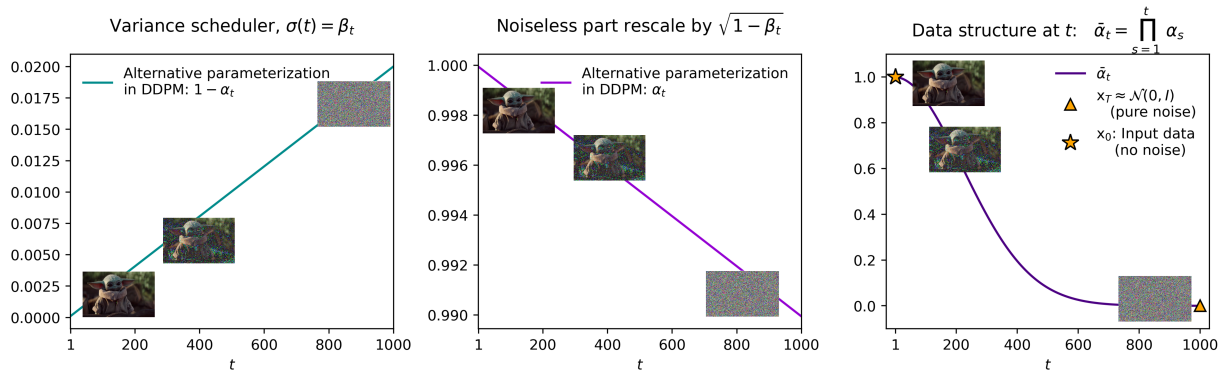


Figure 2.2: **Noise scheduler used in DDPM [15]. Left:** The variance scheduler increase linearly from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$, over the forward pass of $T = 1000$ timesteps, progressively corrupting the data \mathbf{x}_0 . **Middle:** The noiseless part of \mathbf{x}_t is scale by a function of the variance, $\sqrt{1 - \beta_t}$. **Right:** In DDPM, $\alpha_t = 1 - \beta_t$ is defined. Given the addition of independent Gaussian noise, the data structure factor $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$ to allows us to scale \mathbf{x}_0 and obtain \mathbf{x}_t in just one evaluation (see Section 2.2).

The intermediate states \mathbf{x}_t in the forward process are a blend of the data structure from the previous timestep \mathbf{x}_{t-1} and the additive Gaussian noise ϵ_{t-1} . This relationship is illustrated in the middle of Figure 2.2, where the noiseless component also relies on the noise scheduler. The kernel $q(\mathbf{x}_t | \mathbf{x}_{t-1}) \sim \mathcal{N}(\mathbf{x}_t; \sqrt{1 - \beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I})$ and the noise scheduler together create a **location-scale family**², allowing us to sample \mathbf{x}_t as a linear combination of the previous state and Gaussian noise, with the variance β_t set by the scheduler,

$$\mathbf{x}_t = \sqrt{1 - \beta_t} \mathbf{x}_{t-1} + \sqrt{\beta_t} \epsilon_{t-1}. \quad (2.3)$$

¹ Isotropic is a fancy word for “equal shape”, and in this context means that the direction of the covariance matrix is all equal.

² A quick recap about Gaussians distributions: if Z is a standard normal random variable and $X = \mu + \sigma Z$, then X , is a Gaussian variable with mean μ and variance σ^2 , i.e., $X \sim \mathcal{N}(\mu, \sigma^2)$.

At the same time, there is a backward process—*another Markov chain, but in this case, it moves in the reverse timestep direction and has a learnable transition kernel modelled by $p_\theta(\mathbf{x}_{T:0})$* . This kernel allows us to learn the data distribution $q(\mathbf{x}_0)$ by learning the reconstruction process from our prior distribution $p(\mathbf{x}_T)$, which we know how to sample from and evaluate. Specifically, the backward process takes the following form:

$$p_\theta(\mathbf{x}_{T:0}) = p(\mathbf{x}_T) \prod_{t=1}^T p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t) \quad (2.4)$$

$$p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)). \quad (2.5)$$

We know that $p(\mathbf{x}_T)$ is a standard normal distribution when $\beta_T \approx 1$ in Equation (2.3). Additionally, as we will see in the following sections, there are design options for learning the parameters of the kernel mentioned above, namely μ_θ and Σ_θ . In most cases where the data distribution is complex, such as with images or audio, the backward kernel is parameterized by deep neural networks.

2.2. Recursive Reparameterization Trick

The *reparameterization trick*, originally introduced in the work on Variational Autoencoders (VAEs) [36], is also utilized in the diffusion framework with some modifications. The main goal in the VAE context is to remove the stochasticity associated with a node in the directed acyclic graph that represents the dependencies between the loss function, parameters, and operations within the model. Figure 2.3 illustrated this: on the left, a stochastic node is associated with the variable z , which is obtained by sampling the distribution $q(z | \phi, x)$. The variable z depends on both ϕ and x . However, if we want to propagate and compute the partial derivatives of f with respect to ϕ and x , the operation used to obtain z is incompatible with the automatic differentiation techniques of frameworks like PyTorch or JAX [37].

For the VAE, since the distribution of z is Gaussian, it is possible to reparameterize z as a linear function $g(\phi, x, \epsilon)$. This reparameterization removes the stochasticity from the node z and allows dependencies on parameters like ϕ and the input x to be handled. As simple as it may seem, this trick enables the construction of a computational graph that supports proper backpropagation using automatic differentiation, as shown on the right of Figure 2.3. In the diffusion context, we will see that a *recursive* application of the reparameterization trick achieves more than just enabling backpropagation.

Let $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^t \alpha_i$. Now we proceed to unwind the t index until $t = 0$ in Equation (2.3), we end up with a reparameterization of $q(\mathbf{x}_t | \mathbf{x}_{t-1})$ in terms of \mathbf{x}_0 , and it is possible to sample from it by scaling the normal standard distribution $\sim \mathcal{N}(0, \sigma^2)$ accordingly

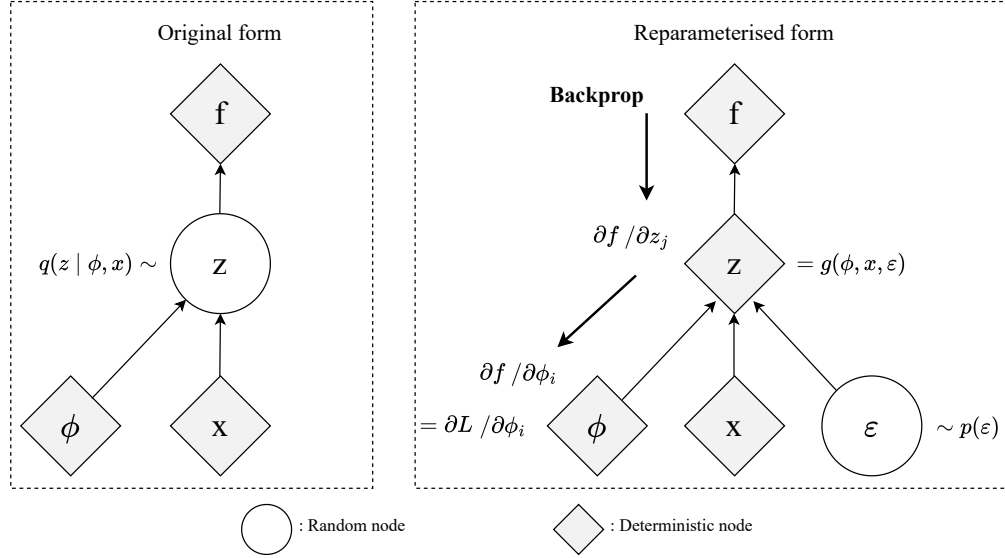


Figure 2.3: **The reparameterization trick enabling backpropagation. Left:** A stochastic node associated with z sampled from $q(z | \phi, x)$, where direct backpropagation is not possible. **Right:** By reparameterizing z as a linear function $g(\phi, x, \epsilon)$, the stochasticity is removed, allowing effective backpropagation of the loss function f with respect to ϕ and x . **Source:** [Variational Auto-Encoders and Extension, NeurIPS Workshop, by Kingma Diederik \(2015\)](#).

its parameters:

$$\begin{aligned}
x_t &= \sqrt{\alpha_t} x_{t-1} + \sqrt{1 - \alpha_t} \epsilon_{t-1} \\
&= \sqrt{\alpha_t} (\sqrt{\alpha_{t-1}} x_{t-2} + \sqrt{1 - \alpha_{t-1}} \epsilon_{t-2}) + \sqrt{1 - \alpha_t} \epsilon_{t-1} \\
&= \sqrt{\alpha_t \alpha_{t-1}} x_{t-2} + \sqrt{\alpha_t - \alpha_t \alpha_{t-1}} \epsilon_{t-2} + \sqrt{1 - \alpha_t} \epsilon_{t-1} \\
&= \sqrt{\alpha_t \alpha_{t-1}} x_{t-2} + \sqrt{1 - \alpha_t \alpha_{t-1}} \bar{\epsilon}_{t-2}, \quad \bar{\epsilon}_{t-2} \sim \mathcal{N}(\mathbf{0}, (\mathbf{1} - \alpha_t + \alpha_t - \alpha_t \alpha_{t-1})\mathbf{I}) \\
&= \dots \\
&= \sqrt{\alpha_t \alpha_{t-1} \dots \alpha_0} x_0 + \sqrt{1 - \alpha_t \alpha_{t-1} \dots \alpha_0} \bar{\epsilon}_0 \\
&= \sqrt{\prod_{i=1}^t \alpha_i} x_0 + \sqrt{1 - \prod_{i=1}^t \alpha_i} \bar{\epsilon}_0 \\
&= \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \bar{\epsilon}_0.
\end{aligned} \tag{2.6}$$

Technically, in the derivation above, the use of a Gaussian transition kernel $q(x_t | x_{t-1})$ allows us to *combine pairs of independent Gaussians into a single Gaussian distribution by simple adding their means and variances*. This is exemplified in Equation (2.6), where the variables ϵ_{t-1} and ϵ_{t-2} are grouped into $\bar{\epsilon}_{t-2}$. When scaled by the square root of the sum of the variance of ϵ_{t-1} and ϵ_{t-2} , we can see that it represents noise belonging to the new Gaussian distribution. By repeatedly applying this property, we eventually marginalize the joint distribution in Equation (2.1) to obtain an analytical form of $q(x_t | x_0)$ for all timesteps $t \in \{0, 1, \dots, T\}$ [26],

$$q(x_t | x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t} x_0, (1 - \bar{\alpha}_t)\mathbf{I}). \tag{2.7}$$

The remarkable outcome of having a closed form solution to compute x_0 is the ability to

sample from $x_0 \rightarrow x_t$ without explicitly passing through the intermediate steps x_1, \dots, x_{t-1} . This means we can transition from x_0 to any arbitrary t in just one evaluation—*since $\bar{\alpha}_t$ is predetermined by the noise scheduler (see the function on the right in Figure 2.2 used in DDPM)*—making the whole diffusion framework computationally feasible.

2.3. Optimization

We can sample a batch of inputs from the data distribution $x_0 \sim q(x_0)$, such as images, and corrupt them with noise $\epsilon \sim \mathcal{N}(0, I)$ according to a scale factor $\bar{\alpha}_t$. This results in intermediate states at timesteps $t \sim \mathcal{U}[1, T]$, which are obtained using Equation (2.6). For training DDPM, the authors focus on learning the mean of the backward kernel and use a time-dependent constant σ_t^2 for the kernel variance³.

Rather than directly modeling the mean of the backward kernel, the approach involves predicting the noise perturbation $\epsilon_\theta^{(t)}$ at timestep t . This predicted noise serves as an indirect method for learning the mean. The key idea is that with an accurate noise prediction, we can determine how much to blend the noise with the structure at timestep t , based on the noise scheduler (see Figure 2.2). This blending process allows us to recover the mean. Consequently, the model is parameterized by θ to learn this predicted noise. The training objective in DDPM is then to minimize the difference between the actual noise ϵ and the predicted noise $\epsilon_\theta^{(t)}$, scaled by a time-dependent factor $\lambda(t)$. This objective is expressed as:

$$\mathbb{E}_{t \sim \mathcal{U}[1, T], x_0 \sim q(x_0), \epsilon \sim \mathcal{N}(0, I)} \left[\lambda(t) \|\epsilon - \epsilon_\theta^{(t)}(x_t)\|^2 \right]. \quad (2.8)$$

Where $\lambda(t) = \beta_t^2 / 2\sigma_t^2 \alpha_t (1 - \bar{\alpha}_t)$. Based on empirical results, the DDPM author shows that it is possible to discard $\lambda(t)$ without affecting sample quality, resulting in the following simplified objective:

$$\mathbb{E}_{t \sim \mathcal{U}[1, T], x_0 \sim q(x_0), \epsilon \sim \mathcal{N}(0, I)} \left[\|\epsilon - \epsilon_\theta^{(t)}(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon)\|^2 \right]. \quad (2.9)$$

The simplified loss is just a mean squared error, and a description of the training procedure is shown in Algorithm 1.

Algorithm 1 DDPM Training

```

repeat
   $x_0 \sim q(x_0)$ 
   $t \sim \mathcal{U}(1, T)$ 
   $\epsilon \sim \mathcal{N}(0, I)$ 
  Take gradient descent step on
     $\nabla_\theta \|\epsilon - \epsilon_\theta^{(t)}(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon)\|^2$ 
until converged

```

Algorithm 2 DDPM Sampling

```

 $x_T \sim \mathcal{N}(0, I)$ 
for  $t = T, \dots, 1$  do
   $z \sim \mathcal{N}(0, I)$ 
   $x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta^{(t)}(x_t) \right) + \sigma_t z$ 
end for
return  $x_0$ 

```

In the next sections we will justify the use of the above loss function and the training algorithm in the context of the Variational Lower Bound (VLB).

³ The authors add a diagonal learnable variance $\Sigma_\theta(x_t)$ but they obtained unstable training and poor sample quality.

2.3.1. Variational Lower Bound

DDPM are trained by optimizing the Variational Lower Bound (aka ELBO) [15, 35] on the negative likelihood of the target distribution using the backward transition kernel p_θ ,

$$\begin{aligned}
-\log p_\theta(\mathbf{x}_0) &= -\log \int p_\theta(\mathbf{x}_{0:T}) d\mathbf{x}_{1:T} \\
&= -\log \int \frac{p_\theta(\mathbf{x}_{0:T})q(\mathbf{x}_{1:T} | \mathbf{x}_0)}{q(\mathbf{x}_{1:T} | \mathbf{x}_0)} d\mathbf{x}_{1:T} \\
&= -\log \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} | \mathbf{x}_0)} \right] \\
&\leq \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} | \mathbf{x}_0)} \right] \tag{2.10} \\
&\leq \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[-\log p(\mathbf{x}_T) - \sum_{t \geq 1}^T \log \frac{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_t | \mathbf{x}_{t-1})} \right] \\
L &:= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[-\log p(\mathbf{x}_T) - \sum_{t > 1}^T \log \frac{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_t | \mathbf{x}_{t-1})} - \log \frac{p_\theta(\mathbf{x}_0 | \mathbf{x}_1)}{q(\mathbf{x}_1 | \mathbf{x}_0)} \right].
\end{aligned}$$

The first step in Equation (2.10) is to apply the definition of the negative log-likelihood and integrate out $\mathbf{x}_{1:T}$ from the joint distribution $p_\theta(\mathbf{x}_{0:T})$. Then, we multiply by 1, introducing $q(\mathbf{x}_{1:T} | \mathbf{x}_0)/q(\mathbf{x}_{1:T} | \mathbf{x}_0)$, and use the expectation operator. We use the Jensen’s inequality to bound the negative log-likelihood by the expectation of the negative log-likelihood.

In the final steps, we break down the expectation into the sum of the negative log-likelihoods of both the forward and backward processes, leveraging the Markov property for each. Note that the final term \mathbf{x}_T decouples from $p_\theta(\mathbf{x}_{0:T})$, as with a sufficiently long diffusion chain, $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$ holds. Similarly, we can isolate the transition involving the initial state \mathbf{x}_0 —*which represents the reward data or images*—from the rest of the process.

The training objective is derived from the VLB, following the methods outlined in the relevant literature [15, 26, 35]. Due to space constraints, we refer to $\mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}$ as \mathbb{E}_q ,

$$\begin{aligned}
L_{\text{VLB}} &= \mathbb{E}_q \left[-\log p(\mathbf{x}_T) - \sum_{t>1}^T \log \frac{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_t | \mathbf{x}_{t-1})} - \log \frac{p_\theta(\mathbf{x}_0 | \mathbf{x}_1)}{q(\mathbf{x}_1 | \mathbf{x}_0)} \right] \\
&= \mathbb{E}_q \left[-\log p(\mathbf{x}_T) - \sum_{t>1}^T \log \frac{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)} \frac{q(\mathbf{x}_{t-1} | \mathbf{x}_0)}{q(\mathbf{x}_t | \mathbf{x}_0)} - \log \frac{p_\theta(\mathbf{x}_0 | \mathbf{x}_1)}{q(\mathbf{x}_1 | \mathbf{x}_0)} \right] \\
&= \mathbb{E}_q \left[-\log p(\mathbf{x}_T) - \sum_{t>1}^T \log \frac{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)} - \sum_{t>1}^T \log \frac{q(\mathbf{x}_{t-1} | \mathbf{x}_0)}{q(\mathbf{x}_t | \mathbf{x}_0)} - \log \frac{p_\theta(\mathbf{x}_0 | \mathbf{x}_1)}{q(\mathbf{x}_1 | \mathbf{x}_0)} \right] \\
&= \mathbb{E}_q \left[-\log p(\mathbf{x}_T) - \sum_{t>1}^T \log \frac{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)} - \log \frac{q(\mathbf{x}_1 | \mathbf{x}_0)}{q(\mathbf{x}_T | \mathbf{x}_0)} - \log \frac{p_\theta(\mathbf{x}_0 | \mathbf{x}_1)}{q(\mathbf{x}_1 | \mathbf{x}_0)} \right] \\
&= \mathbb{E}_q \left[-\log p(\mathbf{x}_T) - \sum_{t>1}^T \log \frac{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)} - \log q(\mathbf{x}_1 | \mathbf{x}_0) + \log q(\mathbf{x}_T | \mathbf{x}_0) - \log \frac{p_\theta(\mathbf{x}_0 | \mathbf{x}_1)}{q(\mathbf{x}_1 | \mathbf{x}_0)} \right] \\
&= \mathbb{E}_q \left[-\left(\log p(\mathbf{x}_T) - \log q(\mathbf{x}_T | \mathbf{x}_0) \right) - \sum_{t>1}^T \log \frac{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)} - \left(\log q(\mathbf{x}_1 | \mathbf{x}_0) + \log \frac{p_\theta(\mathbf{x}_0 | \mathbf{x}_1)}{q(\mathbf{x}_1 | \mathbf{x}_0)} \right) \right] \\
&= \mathbb{E}_q \left[-\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T | \mathbf{x}_0)} - \sum_{t>1}^T \log \frac{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)} - \log p_\theta(\mathbf{x}_0 | \mathbf{x}_1) \right] \\
&= \mathbb{E}_q \left[\underbrace{D_{\text{KL}}(q(\mathbf{x}_T | \mathbf{x}_0) || p(\mathbf{x}_T))}_{L_T} + \sum_{t>1}^T \underbrace{D_{\text{KL}}(q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) || p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t))}_{L_{t-1}} - \underbrace{\log p_\theta(\mathbf{x}_0 | \mathbf{x}_1)}_{L_0} \right] \tag{2.11}
\end{aligned}$$

The key points from the above derivation are as follows:

1. Isolating $\log q(\mathbf{x}_t | \mathbf{x}_0)$ with $\log p(\mathbf{x}_T)$ because both are invariant by the learning process.
2. Reversing $q(\mathbf{x}_t | \mathbf{x}_{t-1})$ by $q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)$ (Bayes theorem) in step 2.
3. The first two terms at the end of Equation (2.11) appear due to the definition of the Kullback-Leibler divergence.

In summary, the variational lower bound loss is a sum of terms that result from the reverse diffusion process $L_{\text{VLB}} = L_T + L_{T-1} + \dots + L_0$. The final term L_T can be discarded because it is constant, given that q has not learnable parameters and \mathbf{x}_T follows a standard normal distribution. The remaining terms are the denoising matching terms, which are crucial in the loss function because their number depends on the length of diffusion chain T (e.g. $T = 1000$ in DDPM).

2.3.2. Denoising Matching Term

The model’s goal is to minimize the discrepancy, measured by the Kullback-Leibler divergence, between the posterior of the encoder—which tells us how to remove the real noise—and the decoder, which predicts how to “reverse” the processes and has learnable parameters θ . Therefore, the loss consists of a sum of denoising matching terms across the diffusion sequence, as shown in Figure 2.4.

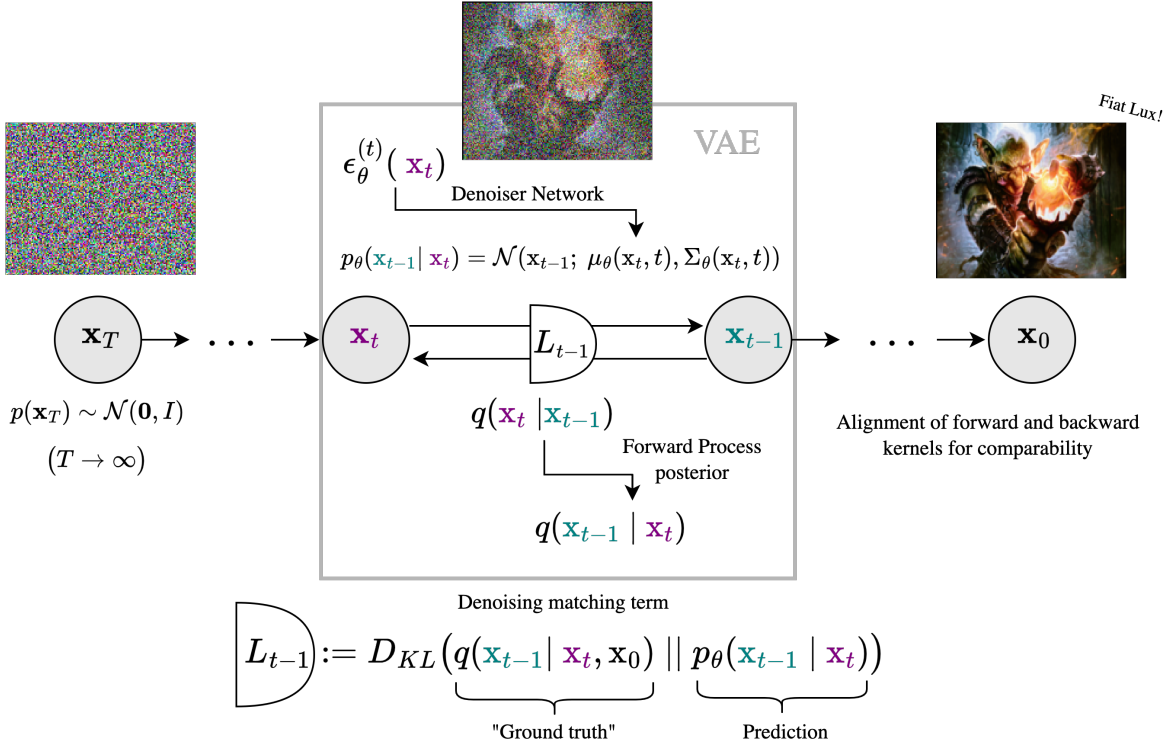


Figure 2.4: **Denoising matching term in action.** **Left:** \mathbf{x}_T is a pure Gaussian noise. **Middle:** Transition from a noisy intermediate state to a less noisy one; the denoising matching term forces $p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)$ to be similar to the posterior forward kernel $q(\mathbf{x}_{t-1} | \mathbf{x}_t)$, making comparable and provide feedback to update the parameters μ_θ and Σ_θ . **Right:** \mathbf{x}_0 the input image during training.

Reversing the forward kernel to $q(\mathbf{x}_{t-1} | \mathbf{x}_t)$ is intractable as it requires access to the entire dataset for computation [38]. To address this, we can make the process tractable by conditioning on \mathbf{x}_0 , effectively anchoring the transition to the data or ground truth signal. This leads to the following expression⁴:

$$q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\mu}_t(\mathbf{x}_t), \tilde{\beta}_t I) \quad (2.12)$$

$$\tilde{\mu}_t(\mathbf{x}_t) = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_t \right) \quad (2.13)$$

$$\tilde{\beta}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t. \quad (2.14)$$

The parameterization in the backward process will be to match $\tilde{\mu}_t$,

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta^{(t)}(\mathbf{x}_t) \right) \quad (2.15)$$

$$\mathbf{x}_{t-1} = \mathcal{N} \left(\mathbf{x}_{t-1}; \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(\mathbf{x}_t, t) \right), \Sigma_\theta(\mathbf{x}_t, t) \right). \quad (2.16)$$

⁴ (See [38] for a detailed derivation)

DDPM demonstrates acceptable performance by learning only the mean, with a time-dependent variance given by $\Sigma_\theta(\mathbf{x}_t, t) = \sigma_t^2 I$. The authors also attempted to learn the variance, but this led to poor results and unstable training. Subsequent studies have shown methods to stabilize and learn the variance as well [17]. However, we will focus solely on parameterizing the mean in the backward kernel, as defined earlier in Equation (2.4),

$$p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \sigma_t^2 I). \quad (2.17)$$

Thus, by working out the KL divergence between the described in Equation (2.12) and Equation (2.17), we derive the objective function presented in Equation (2.8) at the beginning of this section.

Finally, Figure 2.4 highlights the denoising matching term, which resembles the VAE loss term. However, there are key differences between VAEs and DDPMs: (i) VAEs operate in the latent space, whereas DDPMs optimize directly in the data space (e.g., CelebA-HQ 256×256); (ii) VAEs use a forward kernel with learnable parameters, while DDPMs employ a fixed kernel; and (iii) VAEs perform a single sampling step, whereas DDPMs execute multiple steps, making them more comparable to hierarchical VAEs [35].

2.4. Score-based generative models

In this section, we aim to establish a connection between DDPM and score-based generative models [32–34]. The purpose is more conceptual than practical concerning the work of this thesis, as the literature on how to condition diffusion models is built on this theoretical perspective, and it is a way to increase the degree of control over diffusion models.

It turns out that the objective of Equation (2.8), derived from the variational lower bound (Section 2.3.1) and the reparameterization of predicting $\epsilon_\theta^{(t)}$ versus directly the latent states, is equivalent to the objective function of Noise Conditional Score Network (NCSN) [33]. This is a generative model that seeks to learn the data distribution through the score function. We will briefly contextualize this family of models below.

The score function $s(\mathbf{x}_0)$ of the data distribution $q(\mathbf{x}_0)$ is defined as the gradient of the log-likelihood w.r.t \mathbf{x}_0 , i.e. $\nabla_{\mathbf{x}_0} \log q(\mathbf{x}_0)$, and it is invariant to scale changes in the distribution.

First, let’s start by model the data distribution $q(\mathbf{x}_0)$ with $p_\theta(\mathbf{x}_0)$:

$$p_\theta(\mathbf{x}_0) = \frac{e^{-f_\theta(\mathbf{x}_0)}}{Z_\theta}. \quad (2.18)$$

f_θ is the unnormalized probabilistic model and Z_θ is the normalizing constant to ensure that the distribution integrates to 1. The problem is that computing Z_θ is intractable for high-dimensional data, but we can avoid this by inducing the score function $s_\theta(\mathbf{x}_0)$ in Equation (2.18):

$$\begin{aligned}
\log p_\theta(\mathbf{x}_0) &= -f_\theta(\mathbf{x}_0) - \log Z_\theta \\
\nabla_{\mathbf{x}_0} \log p_\theta(\mathbf{x}_0) &= -\nabla_{\mathbf{x}_0} f_\theta(\mathbf{x}_0) - \nabla_{\mathbf{x}_0} \log Z_\theta \\
s_\theta(\mathbf{x}_0) &= -\nabla_{\mathbf{x}_0} f_\theta(\mathbf{x}_0)
\end{aligned} \tag{2.19}$$

$$\mathbb{E}_{p(\mathbf{x})} \left[\|\nabla_x \log p(\mathbf{x}) - s_\theta(\mathbf{x})\|_2^2 \right] \sum_{i=1}^L \lambda(i) \mathbb{E}_{p_{\sigma_i}(\mathbf{x})} [\nabla_x \log p_{\sigma_i}(\mathbf{x}) - s_\theta(\mathbf{x}, i)]. \tag{2.20}$$

By generalizing the number of noise scales to infinity, we further proved that score-based generative models and diffusion probabilistic models can both be viewed as discretizations to stochastic differential equations determined by score functions. This work bridges both score-based generative modeling and diffusion probabilistic modeling into a unified framework.

In DDPM work [15], the authors establish that the denoiser function $\epsilon_\theta^{(t)}$ is equivalent to the score function $s_\theta(\mathbf{x}_t)$ in the score-based generative models. The following equation shows the relationship between the score function and the denoiser function, where is derive by combining Tweedie’s formula and the reparameterization trick [35]:

$$\begin{aligned}
\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t) &= -\frac{1}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta^{(t)}(\mathbf{x}_t) \\
\epsilon_\theta^{(t)}(\mathbf{x}_t) &= -\sqrt{1 - \bar{\alpha}_t} \nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t).
\end{aligned} \tag{2.21}$$

2.5. Sampling

In diffusion models, sampling is a critical process involving the generation of new samples from a learned distribution. After training a model, sampling starts by drawing a sample from a prior distribution, often a standard Gaussian distribution, $p_\theta(\mathbf{x}_T) \sim \mathcal{N}(0, \mathbf{I})$. Ancestral sampling is then applied by iteratively using the denoiser network through a reverse Markovian process. This process involves applying the denoiser at each step to progressively refine the sample until the final output, \mathbf{x}_0 , is achieved. The detailed steps of this procedure are outlined in Algorithm 2, which demonstrates how each iteration in the reverse process refines the sample.

Diffusion Probabilistic Models (DDPM) require a significant number of steps, T , to closely approximate a Gaussian distribution in the reverse process. For example, a typical setting might use $T = 1000$ steps. This large number of steps is crucial for achieving high-quality samples but also makes the sampling process computationally expensive. Each step t in the reverse process depends on the previous state, and thus, the ancestral sampling requires sequential computation from T down to 0, making the process time-consuming. This is reflected in Algorithm 2, where each step in the reverse process involves computing an update based on the previous state and the model’s prediction of noise, $\epsilon_\theta^{(t)}$, to iteratively refine the sample.

In the following sections we will discuss a method to reduce the number of steps required for sampling, as well as a technique to condition the model on additional information.

2.6. Conditioning the model

So far, we know how to indirectly learning the unconditional probability distribution $p(\mathbf{x})$ via the score function $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ (Section 2.4), but how can we condition \mathbf{x} given a signal y , such as a text prompt, another image, or an audio?

By Bayes’ theorem, log operations, and taking the gradient with respect to \mathbf{x} we obtain:

$$\begin{aligned} p(\mathbf{x} | y) &= \frac{p(y | \mathbf{x}) \cdot p(\mathbf{x})}{p(y)} \\ \implies \log p(\mathbf{x} | y) &= \log p(y | \mathbf{x}) + \log p(\mathbf{x}) - \log p(y) \\ \implies \nabla_{\mathbf{x}} \log p(\mathbf{x} | y) &= \nabla_{\mathbf{x}} \log p(y | \mathbf{x}) + \nabla_{\mathbf{x}} \log p(\mathbf{x}). \end{aligned} \tag{2.22}$$

To obtain $\nabla_{\mathbf{x}} \log p(\mathbf{x} | y)$, which is the score-based equivalent of training a DDPM model conditioned on y , it is necessary to solve the discriminative component $\nabla_{\mathbf{x}} \log p(y | \mathbf{x})$ (discriminative model). We will explore how to achieve this in the following sections.

2.6.1. Classifier Guidance (CG)

Classifier Guidance (CG) is a technique to condition the model by using a classifier to guide the sampling process once the diffusion model is trained. By attaching a cost function to an external classifier over \mathbf{x}_t [26, 34, 39, 40] we will be able to compute the gradient of the classifier with respect to the image \mathbf{x}_t in Equation (2.22):

$$\nabla_{\mathbf{x}_t} \log p_{\gamma}(\mathbf{x}_t | y_t) = \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t) + \gamma \nabla_{\mathbf{x}_t} \log p_{\phi}(y_t | \mathbf{x}_t). \tag{2.23}$$

This guidance involves using a discriminative classifier $p_{\phi}(y | \mathbf{x})$ (Murphy, section 9.4 [41]), parameterized by ϕ . Any differentiable classifier that processes images \mathbf{x} , or image-to-feature mappings $\omega(\mathbf{x})$, can be connected to predict a label y . This classifier is needed to compute the term $\nabla_{\mathbf{x}_t} \log p_{\phi}(y | \mathbf{x})$, which is required to adjust the score in the direction of y , as described in Equation (2.23). The classifier’s gradient is scaled by a constant γ , called the guidance scale, which modulates the conditional signal y effect on the sample x . When $\gamma > 1$, is amplify the conditioning, and in the extreme focus in the classifier’ modes, i.e. the distribution temperature⁵ is lowered [39, 40].

Classifier Guidance with CLIP Embeddings. Figure 2.6 illustrates an experiment using classifier guidance to control image generation. Specifically, the OpenAI CLIP model [42] guides a pretrained diffusion model ([google/ddpm-celebahq-256](https://github.com/google/ddpm-celebahq-256)) to generate images of elderly people. The CLIP model includes an image encoder and a text encoder trained to match images with text descriptions (Figure 2.5).

First, we compute the text embedding T_y for the prompt y = “old, senior, oldster, elderly,

⁵ A temperature is a hyperparameter that controls the outcome distribution in models. Higher temperatures amplify lower probability outputs, making the distribution more uniform at the extremes. Conversely, lower temperatures create a more peaked distribution, increasing confidence in the most likely outputs, ultimately collapsing to a delta function at zero. For more details, see this [interactive article](#).

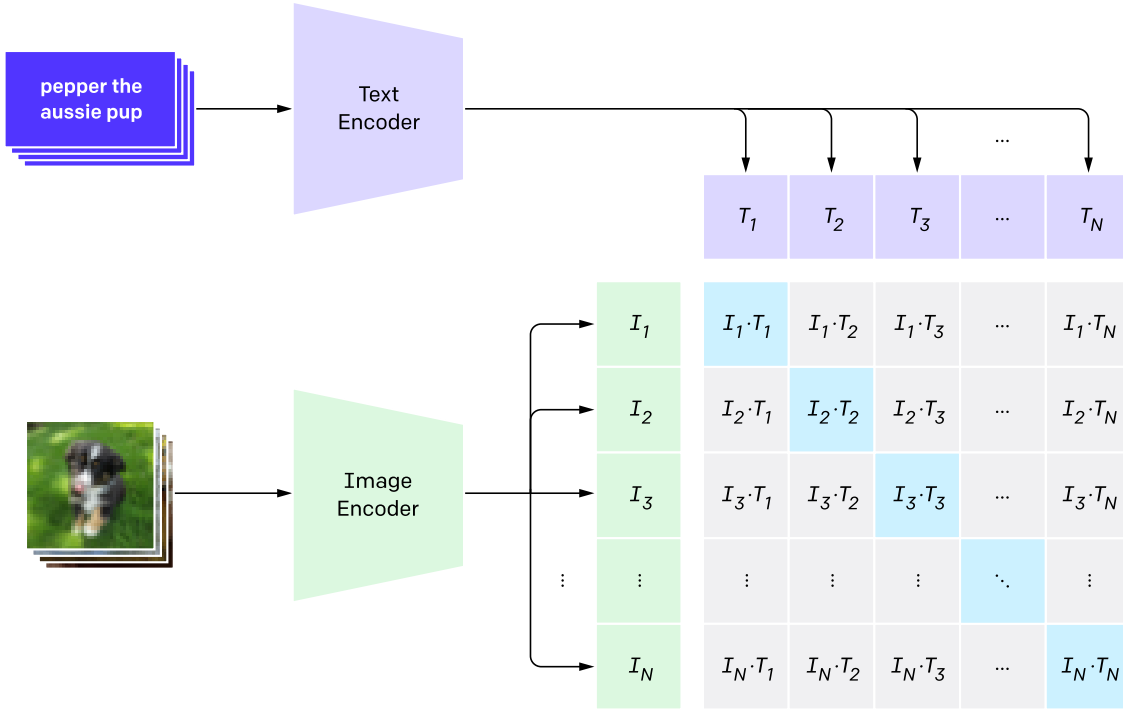


Figure 2.5: **CLIP overview.** Text-to-image joint embedding space. **Source:** Learning Transferable Visual Models From Natural Language Supervision, by Alec Radford et al. (2021) [42].

golden-ager”, representing the desired image attribute. During sampling, we pass a denoised version of the steps $\hat{x}_{t \rightarrow 0}$ (see Section 2.7) to the CLIP image encoder to obtain an image embedding I_t . A loss function $\ell(I_t, T_y)$ measures the distance between these embeddings at each timestep. Notice that T_y is fixed during the whole sampling process, providing the gold standard for signal y . We then backpropagate to obtain the gradient of the loss with respect to x_t , $\nabla_x \log p_\phi(y | x)$.

Using this conditional gradient, we guide the sampling process towards regions where the conditional log-likelihood is higher. By repeating this process throughout the Markovian steps, we can generate samples that match the prompt description.

Limitations of Classifier Guidance. While classifier guidance is powerful, it has limitations:

1. The signal must come from a differentiable classifier, which is not always the case (e.g. decision trees or random forests are not compatible).
2. The gradient must be computed at every denoising step, making it impractical for large models due to computational cost.
3. Classifiers that are robust to noise are uncommon, as most struggle to capture meaningful signals in noisy environments. As a result, during the early stages of diffusion, when the sample is predominantly noise, the classifier provides little effective guidance. This limitation can be mitigated by utilizing denoised versions of the intermediate steps [43]



Figure 2.6: **CLIP classifier guidance for image generation controlled via prompting.** The **first row** shows four samples generated by the pretrained model [google/ddpm-celebahq-256](https://github.com/google/ddpm-celebahq-256). The **second row** shows the same initial noise used to generate the previous images but guided by the CLIP classifier with a guidance scale of $\gamma = 20$, using the prompt “old, senior, oldster, elderly, golden-ager” to transform the subjects into older individuals. Notable changes include the appearance of wrinkles and thinner hair, but artefacts also emerge, such as in the first image of the second row where glasses distort the facial structure. The **third row** illustrates a failure mode, where a much higher guidance scale ($\gamma = 2000$) results in heavily distorted images.

Overall, while classifier guidance offers precise control over diffusion models, these limitations must be considered.

2.6.2. Classifier Free Guidance (CFG)

Classifier-free Guidance (CFG) is a technique for guiding diffusion models without using a separate classifier model [44]. The key is to jointly train the unconditional and conditional model, introducing a null label \emptyset that represents the unconditional model, without any signal effect. This means that conditioning information is dropout during training at some percentage. When this happens, the conditional signal y is randomly discarded and replaced by \emptyset with probability p_{uncond} to train the unconditional model.

During sampling, the output of the model is extrapolated in the direction of $\epsilon_{\theta}^{(t)}(x_t | y)$ and away from $\epsilon_{\theta}(x_t | \emptyset)$ as follows:

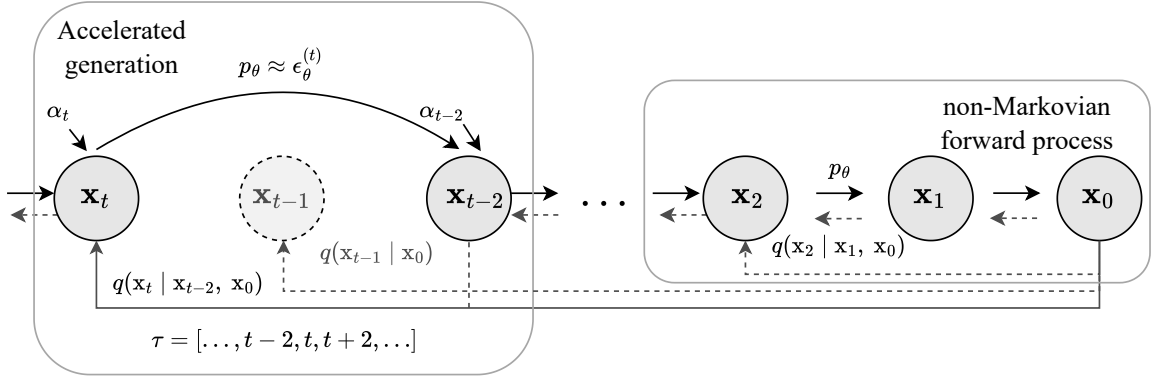


Figure 2.7: **DDIM non-Markovian forward process.** **Left:** Illustration of accelerated generation skipping the uneven intermediate steps, the denoiser network $\epsilon_\theta^{(t)}$ predicts the amount of noise added to \mathbf{x}_t from step $t - 2$, instead t . **Right:** the non-Markovian graphical model.

$$\hat{\epsilon}_\theta^{(t)}(\mathbf{x}_t | \mathbf{y}) = \epsilon_\theta^{(t)}(\mathbf{x}_t | \emptyset) + \gamma \cdot (\epsilon_\theta^{(t)}(\mathbf{x}_t | \mathbf{y}) - \epsilon_\theta^{(t)}(\mathbf{x}_t | \emptyset)). \quad (2.24)$$

2.7. Denoising Diffusion Implicit Models

The work on *Denoising Diffusion Implicit Models* (DDIM) demonstrates that the forward process in DDPM can be reformulated as a family of non-Markovian generative processes. The resulting variational training objectives share a common surrogate objective, which aligns closely with the one used to train DDPM. This reformulation enables the use of non-Markovian diffusion processes during inference with the pretrained DDPM model, allowing for significantly shorter generative Markov chains that can be sampled in just a few steps. While this improves sampling efficiency, it introduces a trade-off in sample quality.

Prediction of noise-free observations. We can obtain a denoised observation $\tilde{\mathbf{x}}_{t \rightarrow 0}$ from any arbitrary intermediate state \mathbf{x}_t by simple isolating \mathbf{x}_0 in Equation (2.6) (Section 2.2)⁶. The caveat is that instead of sampling $\epsilon \sim \mathcal{N}(0, I)$, we use the noise prediction given by the pretrained denoiser network $\epsilon_\theta^{(t)}$ at level t ,

$$f_\theta^{(t)}(\mathbf{x}_t) = \frac{\mathbf{x}_t - \sqrt{1 - \alpha_t} \epsilon_\theta^{(t)}(\mathbf{x}_t)}{\sqrt{\alpha_t}} = \tilde{\mathbf{x}}_{t \rightarrow 0}. \quad (2.25)$$

Using Bayes rule we obtain the following expression for the non-Markovian forward process, illustrated in Figure 2.7:

$$q_\sigma(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\sqrt{\alpha_{t-1}} \mathbf{x}_0 + \sqrt{1 - \alpha_{t-1} - \sigma_t^2} \cdot \frac{\mathbf{x}_t - \sqrt{\alpha_t} \mathbf{x}_0}{\sqrt{1 - \alpha_t}}, \sigma_t^2 \mathbf{I}). \quad (2.26)$$

Then, by replacing Equation (2.25) in Equation (2.26) we can compute $q_\sigma(\mathbf{x}_{t-1} | \mathbf{x}_t, f_\theta^{(t)}(\mathbf{x}_t))$ and sample \mathbf{x}_{t-1} by:

⁶ In DDIM paper they use $\alpha_t = \alpha_t / \alpha_{t-1}$ to avoid using β_t , α_t , and $\bar{\alpha}_t$ (See Appendix C.2 in [16])

$$\begin{aligned}
x_{t-1} &= \sqrt{\alpha_{t-1}}x_0 + \sqrt{1 - \alpha_{t-1} - \sigma_t^2} \cdot \frac{x_t - \sqrt{\alpha_t}x_0}{\sqrt{1 - \alpha_t}} + \sigma_t\epsilon_t \\
x_{t-1} &= \sqrt{\alpha_{t-1}} \left(\frac{x_t - \sqrt{1 - \alpha_t}\epsilon_\theta^{(t)}(x_t)}{\sqrt{\alpha_t}} \right) + \sqrt{1 - \alpha_{t-1} - \sigma_t^2} \\
&\quad \cdot \frac{x_t - \sqrt{\alpha_t}((x_t - \sqrt{1 - \alpha_t}\epsilon_\theta^{(t)}(x_t))/\sqrt{\alpha_t})}{\sqrt{1 - \alpha_t}} + \sigma_t\epsilon_t \\
x_{t-1} &= \underbrace{\sqrt{\alpha_{t-1}} \left(\frac{x_t - \sqrt{1 - \alpha_t}\epsilon_\theta^{(t)}(x_t)}{\sqrt{\alpha_t}} \right)}_{\text{“predicted } x_0\text{”}} + \underbrace{\sqrt{1 - \alpha_{t-1} - \sigma_t^2} \cdot \epsilon_\theta^{(t)}(x_t)}_{\text{“direction pointing to } x_t\text{”}} + \underbrace{\sigma_t\epsilon_t}_{\text{random noise}} .
\end{aligned} \tag{2.27}$$

Multiple generative processes are contained in Equation (2.27) depending of σ election. Again, the main takeaway is that all share the same surrogate objective with DDPM, it is possible to use the same denoiser model $\epsilon_\theta^{(t)}$ without any additional training for different inference processes. The stochasticity comes from $\epsilon_t \sim \mathcal{N}(0, I)$, a Gaussian noise independent of x_{t-1} , and modulate by the noise scheduler.

1. The generative process, obtained by reversing Markovian forward process in DDPM, can be derived from Equation (2.27) by using $\sigma_t = \sqrt{(1 - \alpha_{t-1})/(1 - \alpha_t)}\sqrt{1 - \alpha_t/\alpha_{t-1}}$ for all t .
2. A deterministic forward process arise when $\sigma_t = 0$, for all t , except for the initial noise. The authors names this specific generative process as denoising diffusion implicit models (DDIM).

Accelerated inference. With the possibility to design non-Markovian forward processes using the same pretrained model, the DDIM authors propose to accelerate the inference process by skipping intermediate steps. Subset of latent variables $\{x_{\tau_1}, \dots, x_{\tau_s}\}$ where τ is an increasing subset sequence of $[1, \dots, T]$, of length S . We define a forward process over the subset such that $q(x_{\tau_i} | x_0) = \mathcal{N}(\sqrt{\alpha_{\tau_i}}x_0, (1 - \alpha_{\tau_i})I)$ matches the marginals, see left part of the Figure 2.7. Therefore, we can accelerate the inference increasing the number of steps between the sequence subset, and introducing a trade-off between sample quality and computational cost in doing so.

Introducing the hyperparameter $\eta \in \mathbb{R}^+$ for control the sampling stochasticity from DDIM ($\eta = 0$) to DDPM ($\eta = 1$) in Equation (2.27), and use subsequence τ of different length for control the sample acceleration,

$$\sigma_{\tau_i}(\eta) = \eta\sqrt{(1 - \alpha_{\tau_{i-1}})/(1 - \alpha_{\tau_i})}\sqrt{1 - \alpha_{\tau_i}/\alpha_{\tau_{i-1}}} . \tag{2.28}$$

It is possible to produce samples with a similar quality of DDPM 1000 steps using 20 – 100 DDIM steps by measuring the quality with the Fréchet inception distance (FID). This means a speed ups of $10\times$ to $100\times$ over DDPM generation process [16]. In practice, DDIM is commonly used as a sampler and an efficient way to accelerate the sampling process, as utilized in this thesis.

Sample consistency with DDIM. As a consequence of the deterministic forward process, we have *sample consistency*. Given any arbitrary length τ , the image generated from the initial sample step x_T are fairly similar, in the DDIM authors words “*it would appear that x_T alone would be an informative latent encoding of the image*”. Hence, they suggest that high level features of the images x_0 are determined by the initial sample step x_T .

The sample consistency property can be useful for inverse problems. For instance, in Figure 2.8, given an input image (Pedro Pascal face) we can estimate the initial noise \tilde{x}_t , and run the generative process to recover the input. However, the input is not perfectly recovered, the reason could be:

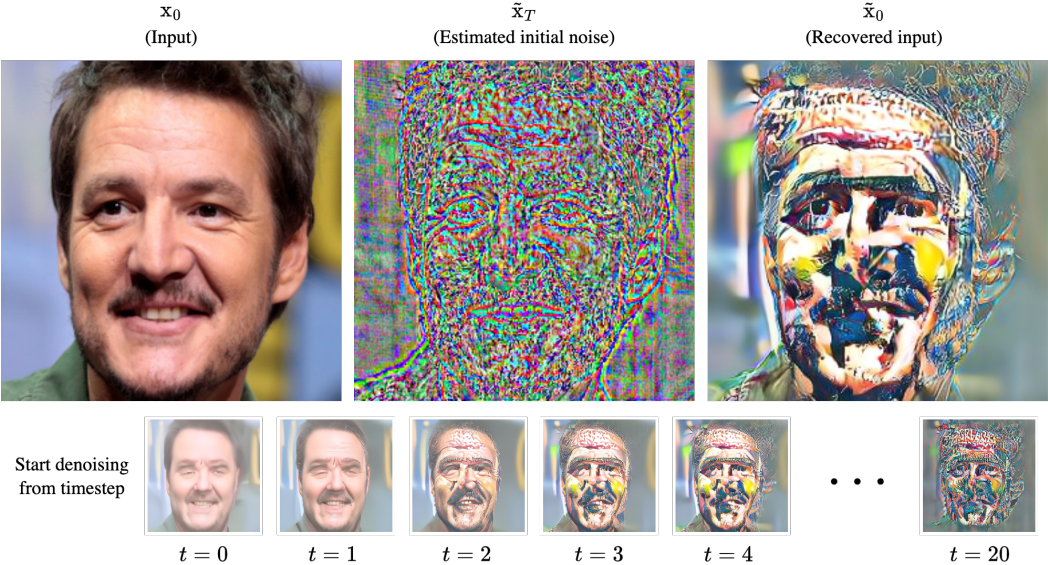


Figure 2.8: **DDIM inversion example using 50 inference steps.** A Pedro Pascal photo to estimate the initial noise \tilde{x}_T using DDIM with the pretrained model [google/ddpm-celebahq-256](#) on the celebrity faces dataset CelebaHQ. Then, the input is reconstructed using the estimated noise as starter point. Below images are different results skipping the first n inference steps of the denoising process.

- It is interesting that high level features are encoded by x_T and doesn't vary as long as increase the number of steps to generate the final sample. However, the quality is better as long as the length of the trajectory increase. Therefore, a trick to obtain a recovered input with higher fidelity is to skip the first steps and start the denoising process from there, avoiding higher feature modification.
- The estimation error is illustrate as a trade-off between denoising from the first step that conserve face semantic but change higher features and skipping a set of denoiser steps conserving semantic but with lower quality.
- External input not generated by the model are not guarantee to be recover given the generative capacities of ϵ_θ . In the illustration, the input is external but is the generative process is perform by a DDPM trained model on a dataset of celebrities faces.

2.8. Summary

In summary, a typical diffusion model framework can be implemented as follows:

A **forward process** that consists of a Markovian chain of states $\{x\}_{0:T}$ that takes the original image x_0 and iteratively adds noise until getting isotropic Gaussian noise x_T .

In the reverse, or **backward process**, a model learns how to gradually remove the noise to recover the data structure. In most implementations model this denoising endeavour using some U-net neural network architecture with attention modules.

The sequence of intermediate states provides a detailed roadmap when destroying the observation such as images and leaves a trail to the denoiser network to recover the input structure. Therefore, the forward process acts in a **self-supervised** manner generating the labels for the data itself.

It is possible to use a model to predict the noisy image directly, or indirectly via predicting the noise, and then subtracting from the state. Nevertheless, **the core idea of the training objective is to reduce the error between the noise prediction and the true noise** used to destroy the sample at a given step.

The noise used to destroy the structure in data is carefully handled by a deterministic function called **noise scheduler**. It determines the variance schedule, β_1, \dots, β_T , or the specific amount of noise injected in each timestamp t during the Markovian chain.

Finally, we have a model that can **generate novel samples** from the data distribution by **sampling from a standard normal distribution and then successively remove noise**.

3

Reinforcement Learning

Reinforcement learning (RL) [45] is all about the interaction between an agent and its environment, where learning occurs through trial-and-error. The agent observes the current state of the environment, takes actions based on these observations, and influences new possible state configurations while receiving rewards based on its actions. The primary objective is to maximize cumulative rewards, which drives the agent’s sequence of decisions towards achieving specific goals, such as escaping from a maze, **winning an Atari Game** [46], or **defeating the world champion of Go** [47]. But how does the agent learn to act effectively to achieve its goal? RL algorithms are designed to maximize the total rewards obtained by the agent, thereby guiding its actions towards these objectives.

In this chapter, we will introduce the essential concepts of RL required to implement these agents. We will specifically focus on model-free RL, where the agent learns to act without constructing a model of its environment, as opposed to model-based RL, which involves such modeling. The goal is to design agents that learn to perform well solely by consuming experiences from their environment. By understanding the fundamentals of designing such agents, we will explore policy optimization methods, such as REINFORCE and PPO, which are used to refine the agent’s behavior. Additionally, we will briefly discuss the use of reward models and reinforcement learning from human feedback (RLHF).

With the knowledge gained from this chapter, we will be equipped to explore the connection between RL and diffusion models in the next chapter. This will allow us to build agents based on diffusion models that can generate enhanced samples aligned with the objectives specified by reward models.

3.1. The Framework for Learning to Act

The starting point for designing agents that learn to act is the Markov Decision Process (MDP) framework [45]. An MDP is a mathematical object that describes the interaction between the agent and the environment. This interaction is characterized by a tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \rho_0, \gamma \rangle$, where:

1. \mathcal{S} , **state space**, set of possible states in the environment.
2. \mathcal{A} , **action space**, set of possible actions available to the agent.

3. $P : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$, **transition probability distribution**, which gives the probability of the environment for transitioning to a new state s_{t+1} with a reward r_t given the current state s_t and action a_t .
4. $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, **reward function**, which provides a scalar feedback signal r_t (aka reward) to the agent after taking an action a_t and reaching the subsequent state s_{t+1} .
5. ρ_0 , **initial state distribution**, which determines the probability of the agent starting in a particular state.
6. $\gamma \in [0, 1]$ is the **discount factor**, which determines the importance of future rewards.

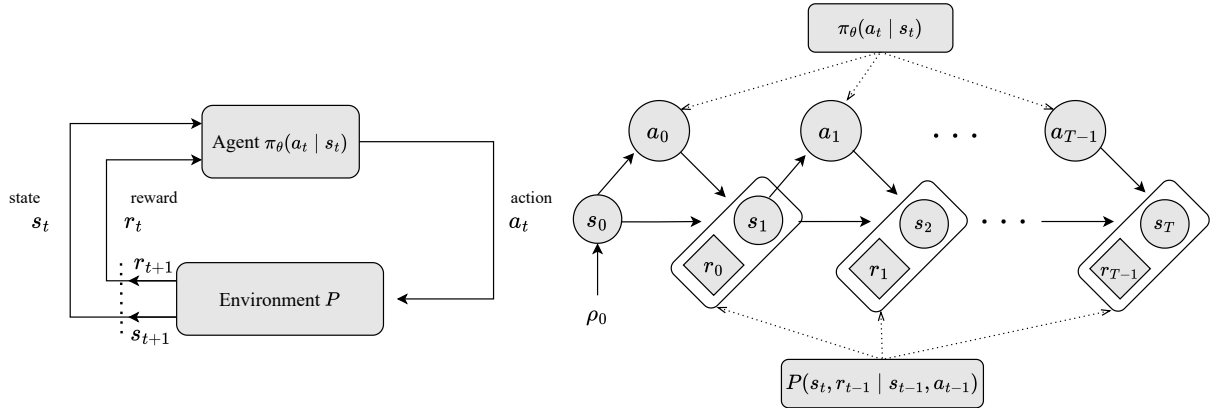


Figure 3.1: **Left:** A loop representation of a Markov Decision Process (MDP). **Right:** An unrolled MDP depicting an episodic case with a finite horizon T and a parameterized policy π_θ .

Markov Decision Processes generate sequences of state-action pairs, or trajectories τ , starting from an initial state $s_0 \sim \rho_0$. The agent’s behavior is determined by a policy $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$, which maps states to a probability distribution over actions. An action $a_0 \sim \pi(s_0)$ is chosen, leading to the next state s_1 according to the transition distribution P , and a reward $r_0 = R(a_0, s_0)$ is received. This cycle repeats iteratively, with the agent selecting actions, transitioning through states, and receiving rewards, as shown on the left side of Figure 3.1. Thus, the trajectory τ encapsulates the dynamic sequence of state-action pairs resulting from the agent’s interaction with its environment.

The process can continue indefinitely, known as an infinite horizon, or be confined to episodes that end in the terminal state s_T , referred to as episodic tasks, such as winning or losing a game, as illustrated on the right side of Figure 3.1. It is important to note that the transition to the next state depends only on the current state and action, not on the sequence of prior events. This characteristic is known as the *Markov property*, which states that the future and the past are conditionally independent, given the present (*memoryless*). In this work, we focus on the episodic setting, where the trajectory begins at s_0 and concludes at s_T , with a finite horizon T . Therefore, the trajectory τ is defined as $\tau = (s_0, a_0, \dots, s_{T-1}, a_{T-1}, s_T)$, summarizing the agent’s behavior throughout the episodic task.

In reinforcement learning, the primary goal is for the agent to develop a behavior that maximizes the expected return from its actions results within the environment. This concept of

maximization is formalized through the objective function $\mathcal{J}_{\text{RL}}(\theta)$, which aims to maximize the expected return over a collection of trajectories $\{\tau^{(i)}\}_{1:N}$ generated by the policy π , commonly referred to as “policy rollouts”⁷. The objective function is defined as follows:

$$\mathcal{J}_{\text{RL}} = \underset{\pi}{\text{maximize}} \mathbb{E}_{\tau \sim \pi} [R(\tau)].$$

The return over a trajectory τ is defined as the accumulated discounted rewards of the trajectory, $R(\tau) = \sum_{t=0}^{T-1} \gamma^t r_t$. The reward signals r_t are the immediate effect of taking the actions, and the return is the cumulative rewards obtained during the trajectory, considering a discount factor γ , which gives more importance to the rewards of nearer actions than to future rewards.

3.2. Policy Optimization

In reinforcement learning there are different approaches to solve the MDP formulated in the previous section, which are summarized in Figure 3.2. The most common are value-based methods and policy-based methods. In value-based methods, the agent learns which state is more valuable and take action that leads to it. In policy-based methods, the agent learns a policy that directly maps states to actions. In this work we will focus on the latter methods, specifically in policy gradients.

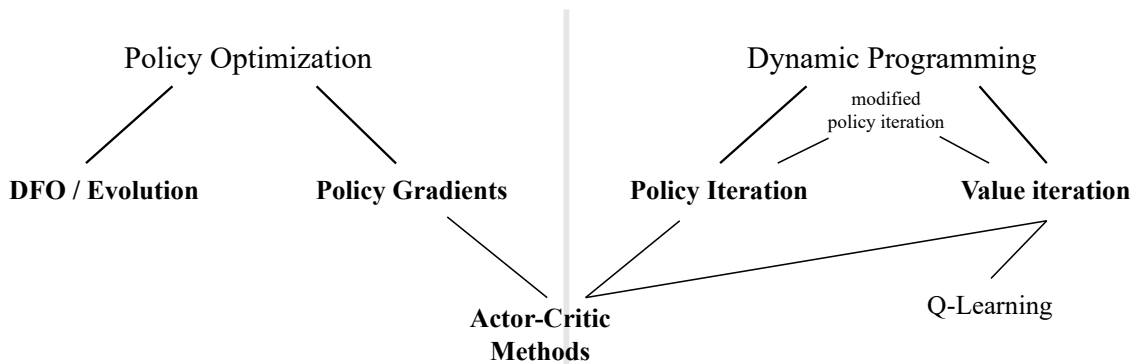


Figure 3.2: Illustration of a taxonomy of model-free RL algorithms. **Source:** [Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs](#) by John, Schulman (2016) [48].

Other approaches for finding a policy is by non solving the MDP, but by directly optimizing the policy. This is the case of derivative free optimization (DFO), or evolutionary algorithms, in which the policy is parameterized by a vector θ , and the agent explores the space of parameters by searching. Nothing of the temporal structure and actions of the MDPs are considered in this kind of solution.

Policy gradient methods provide a way to reduce reinforcement learning to stochastic gradient descent, by providing a connection between how function approximation is solved in supervised learning settings.

⁷ The term “rollout” is used to describe the process of simulating the agent’s behavior in the environment by executing the policy π and observing the resulting trajectory τ .

3.2.1. Learning the Policy

The starting point is to think of trajectories as units of learning instead of individual observations (i.e., actions). What dynamics generate a trajectory? Given a policy π_θ , represented as a function with parameter $\theta \in \mathbb{R}^d$, whose input is a representation of the state and whose output is action selection probabilities, we can deploy the agent into its environment at an initial state s_0 and observe its actions in inference mode or *evaluation phase* [49]. The agent continuously promotes actions based on the current state s_t until the episode ends in a terminal state, when $t = T$. At this point, we can determine if the goal was accomplished, such as winning the ATARI Pong game, *or generating aesthetically pleasing samples from a diffusion model*. The returns are the scalar value that assesses performance whether we have achieved the ultimate goal, effectively acting as a “proxy” of a label for the overall trajectory. Thus, the trajectory serves as our unit of learning, and the remaining task is to establish the feedback mechanism for the *learning phase*.

Intuitively, we want to collect the trajectories and make the good trajectories and actions more probable, and push the actions towards better actions.

Mathematically, we aim to perform stochastic optimization to learn the agent’s parameters. This involves obtaining gradient information from sample trajectories, with performance assessed by a scalar-value function (i.e. reward). The optimization is stochastic because both the agent and the environment contain elements of randomness, meaning we can only compute estimates of the gradient. Crucially, we are estimating the gradient of the expected return with respect to the policy parameters. To address this, we employ Monte Carlo Gradient Estimation [50], specifically using the score function method. From a machine learning perspective, this involves dealing with the stochasticity of the gradient estimates, \hat{g} , and using gradient ascent algorithms to update the policy parameters based on these estimates, along with a learning rate α to control the step size of the optimization process,

$$\theta \leftarrow \theta + \alpha \hat{g}_N. \tag{3.1}$$

3.2.2. Gradient Estimation via Score Function

The gradient estimation can be obtained using the score function gradient estimator. Let’s introduce the following probability objective \mathcal{F} , defined in the **ambient space** $\mathcal{X} \in \mathbb{R}^n$ and with parameters $\theta \in \mathbb{R}^n$,

$$\mathcal{F}(\theta) = \int_{\mathcal{X}} p(\mathbf{x}; \theta) f(\mathbf{x}) \, d\mathbf{x} = \mathbb{E}_{p(\mathbf{x}; \theta)} [f(\mathbf{x})]. \tag{3.2}$$

Here, f is a scalar-valued function, similar to how the reward is represented in the reinforcement learning setting. The *score function* is the derivative of the log probability distribution $\nabla_\theta \log p(\mathbf{x}; \theta)$ with respect to its parameters θ . We can use the following identity to establish a connection between the score function and the probability distribution $p(\mathbf{x}; \theta)$,

$$\begin{aligned} \nabla_\theta \log p(\mathbf{x}; \theta) &= \frac{\nabla_\theta p(\mathbf{x}; \theta)}{p(\mathbf{x}; \theta)} \\ p(\mathbf{x}; \theta) \nabla_\theta \log p(\mathbf{x}; \theta) &= \nabla_\theta p(\mathbf{x}; \theta). \end{aligned} \tag{3.3}$$

Therefore, taking the gradient of the objective $\mathcal{F}(\theta)$ with respect to the parameter θ , we have

$$\begin{aligned}
g &= \nabla_{\theta} \mathbb{E}_{p(\mathbf{x};\theta)}[f(\mathbf{x})] = \nabla_{\theta} \int_{\mathcal{X}} p(\mathbf{x};\theta) f(\mathbf{x}) d\mathbf{x} \\
&= \int_{\mathcal{X}} \nabla_{\theta} p(\mathbf{x};\theta) f(\mathbf{x}) d\mathbf{x} \\
&= \int_{\mathcal{X}} p(\mathbf{x};\theta) \nabla_{\theta} \log p(\mathbf{x};\theta) f(\mathbf{x}) d\mathbf{x} \\
&= \mathbb{E}_{p(\mathbf{x};\theta)} [f(\mathbf{x}) \nabla_{\theta} \log p(\mathbf{x};\theta)].
\end{aligned} \tag{3.4}$$

The use of the log-derivative rule of Equation (3.3) to introduce the score function in Equation 3.4 is also known as the *log-derivative trick*. Now, we can compute an estimate of the gradient, \hat{g} , using Monte Carlo estimation with samples from the distribution $p(\mathbf{x};\theta)$ as follows:

$$\hat{g}_N = \frac{1}{N} \sum_{i=1}^N f(\hat{\mathbf{x}}^{(i)}) \nabla_{\theta} \log p(\hat{\mathbf{x}}^{(i)}; \theta). \tag{3.5}$$

We draw N samples $\hat{\mathbf{x}} \sim p(\mathbf{x};\theta)$, compute the gradient of the log-probability for each sample, and multiply by the scalar-valued function f evaluated at the sample. The average of these terms is an unbiased estimate of the gradient of the objective g , which we can use for gradient ascent using Equation (3.1).

There are two important points to mention about Equation (3.5).

1. The function f can be any arbitrary function we can evaluate on \mathbf{x} . Even if f is non-differentiable with respect to θ , it can still be used to compute the gradient estimation \hat{g} .
2. The expectation of the score function is zero, meaning that the gradient estimator is unbiased

$$\begin{aligned}
\mathbb{E}_{p(\mathbf{x};\theta)} [\nabla_{\theta} \log p(\mathbf{x};\theta)] &= \int_{\mathcal{X}} p(\mathbf{x};\theta) \nabla_{\theta} \log p(\mathbf{x};\theta) d\mathbf{x} \\
&= \int_{\mathcal{X}} p(\mathbf{x};\theta) \frac{\nabla_{\theta} p(\mathbf{x};\theta)}{p(\mathbf{x};\theta)} d\mathbf{x} \\
&= \int_{\mathcal{X}} \nabla_{\theta} p(\mathbf{x};\theta) d\mathbf{x} \\
&= \nabla_{\theta} \int_{\mathcal{X}} p(\mathbf{x};\theta) d\mathbf{x} = \nabla_{\theta} 1 = 0.
\end{aligned} \tag{3.6}$$

The last point is particularly useful because we can replace f with a shifted version given a constant β , and still obtain an unbiased estimate of the gradient, which can be beneficial for the optimization task:

$$\hat{g}_N = \mathbb{E}_{p(\mathbf{x};\theta)} [(f(\mathbf{x}) - \beta) \nabla_{\theta} \log p(\mathbf{x};\theta)]. \tag{3.7}$$

Using a **baseline function** to determine β , that does not depend on the parameter θ , can reduce the variance of the estimator [50]. The baseline function, which satisfies the property

in Equation (3.6), can be any function independent of θ . When a baseline is chosen to be close to the scalar-valued function f , it effectively reduces the variance of the estimator. This reduction in variance helps stabilize the updates by minimizing fluctuations in the gradients estimates, leading to more reliable and efficient learning.

3.3. Vanilla Policy Gradient, aka REINFORCE

The REINFORCE algorithm [51] translates the previous derivation of gradient estimation via the score function into reinforcement learning terminology. This is the earliest member of the Policy Gradient family (Figure 3.2), where the objective is to maximize the expected return of the trajectory τ under a policy π parameterized by θ (e.g., a neural network). At each state s_t , the agent takes an action a_t according to the policy π , which generates a probability distribution over actions $\pi(a_t | s_t; \theta)$. Here, we will use the notation $\pi_\theta(\cdot)$ instead of $\pi(\cdot; \theta)$.

As we mentioned in previous section, a trajectory τ represents the sequence of state-action pairs resulting from the agent’s interaction with its environment. From the initial state s_0 to the terminal state s_T , the trajectory τ is a sequence of states and actions, $\tau = (s_0, a_0, \dots, s_{T-1}, a_{T-1}, s_T)$, which describes how the agent acts during the episodic task. Let $p_\theta(\tau)$ be the probability of obtaining the trajectory under the policy π_θ .

We thus have a distribution of trajectories. Remember that the trajectory τ is the learning unit for our policy π_θ , as it tells us if the consequences of each action led to a favorable final outcome on the terminal state s_T (e.g. win/lose). The goal is to maximize the expected return of the trajectories on average, and the return $R(\tau)$ could be the cumulative rewards obtained during the *episode* or the discounted rewards. The expected return is given by the following expression:

$$\mathcal{J}(\theta)_{\text{RL}} = \mathbb{E}_{\tau \sim p_\theta(\tau)} [R(\tau)]. \quad (3.8)$$

This is the objective we want to maximize, which is a particular case of Equation (3.2) with the scalar-valued function $f(\mathbf{x}) = R(\tau)$, representing the return of the trajectory. Let’s use the techniques from the previous section to compute the gradient of the objective in Equation (3.8) with respect to the policy parameter θ . The gradient estimation is given by:

$$\nabla_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} [R(\tau)] = \mathbb{E}_{\tau \sim p_\theta(\tau)} [R(\tau) \nabla_\theta \log p_\theta(\tau)]. \quad (3.9)$$

What is $p_\theta(\tau)$ exactly? Given that the trajectory is a sequence of states and actions, and assuming the Markov property imposed by the MDP, the probability of the trajectory is defined as follows:

$$\begin{aligned} p_\theta(\tau) &= p_\theta(s_0, a_0, s_1, a_1, \dots, s_{T-1}, a_{T-1}, s_T) \\ &= \rho(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t | s_t) P(s_{t+1}, r_t | a_t, s_t). \end{aligned} \quad (3.10)$$

In the above expression, $\rho(s_0)$ denotes the distribution of initial states, while $P(s_{t+1}, r_t | a_t, s_t)$ represents the transition model, which updates the environment context based on the action a_t taken in the current state s_t . A crucial step in estimating the gradient is computing the

logarithm of the trajectory probability. Following this, we calculate the gradient with respect to the policy parameter θ ,

$$\begin{aligned}\log p_\theta(\tau) &= \log \rho(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t | s_t) + \log P(s_{t+1}, r_t | a_t, s_t) \\ \nabla_\theta \log p_\theta(\tau) &= \cancel{\log \rho(s_0)} + \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) + \cancel{\log P(s_{t+1}, r_t | a_t, s_t)} \\ \nabla_\theta \log p_\theta(\tau) &= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t).\end{aligned}\tag{3.11}$$

The distribution of initial states and the transition probabilities are disregarded because they are independent of θ , thereby simplifying significantly the computations needed for gradient estimation. By substituting the final expression from Equation (3.11) into the gradient estimation of the objective in Equation (3.9), we derive the REINFORCE gradient estimator

$$\begin{aligned}g &= \nabla_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} [R(\tau)] \\ &= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) R(\tau) \right] \\ \hat{g} &= \frac{1}{|\mathcal{D}^{\pi_\theta}|} \sum_{\tau \in \mathcal{D}^{\pi_\theta}} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) R(\tau) \right].\end{aligned}\tag{3.12}$$

The core concept is to collect a set of trajectories \mathcal{D}^{π_θ} under the policy π_θ and update the policy parameters θ to increase the likelihood of high-reward trajectories while decreasing the likelihood of low-reward ones, as illustrated in Figure 3.3. This trial-and-error learning approach, described in Algorithm 3, repeats this process over multiple iterations, reinforcing successful trajectories and discouraging unsuccessful ones, thus encoding the agent’s behavior in its parameters.

Reducing the variance of the estimator. Using two techniques, *reward-to-go* and *baseline*, we can improve the quality of the gradient estimator in Equation (3.12).

Algorithm 3 Vanilla Policy Gradient, aka REINFORCE

Initialize policy π_θ , set learning rate α

for iteration = 0, 1, 2, . . . , N **do**

 Collect a set of trajectories $\mathcal{D}^{\pi_\theta} = \{\tau^{(i)}\}$ by sampling from the current policy π_θ

 Calculate the returns $R(\tau)$ for each trajectory $\tau \in \mathcal{D}^{\pi_\theta}$

 Update the policy: $\theta \leftarrow \theta + \alpha \left(\frac{1}{|\mathcal{D}^{\pi_\theta}|} \sum_{\tau \in \mathcal{D}^{\pi_\theta}} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) R(\tau) \right] \right)$

end for

The reward-to-go technique is a simple trick that can reduce the variance of the gradient estimator by taking advantage of the *temporal structure* of the problem. The idea is to weight the gradient of the log-probability of an action a_t by the sum of rewards from the current timestep t to the end of the trajectory $T - 1$. This way, the gradient of the log-

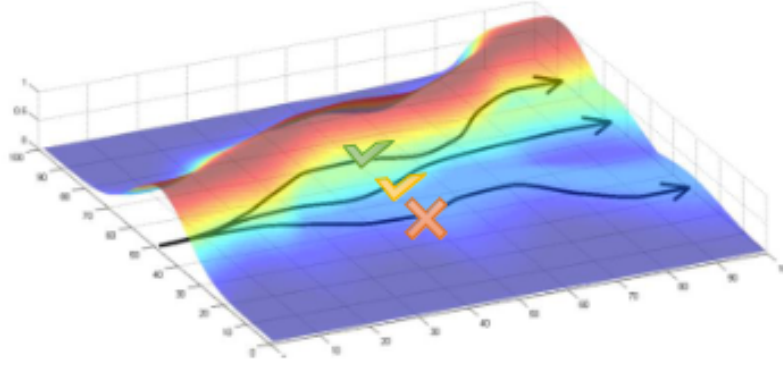


Figure 3.3: **Illustration of three simulated trajectories**, denoted as $\{\tau^{(i)}\}$ where $i = (1, 2, 3)$, traversing the parametric space $\theta \in \mathbb{R}^2$ under the policy π_θ . Each trajectory is marked with a colored symbol (cross, check) representing its *goodness* based on the reward function $R(\tau^{(i)})$. **Source:** Policy Gradients Lecture, Deep Reinforcement Learning Course by Sergey Levine.

probability of an action is only weighted by the consequence of that action on the future rewards, removing terms that do not depend on a_t . Let's introduce this technique by using the gradient estimation in Equation (3.12) and replacing $R(\tau)$ naively using the sum of total trajectory reward⁸

$$\begin{aligned}
 \hat{g}_\theta &= \frac{1}{|\mathcal{D}^{\pi_\theta}|} \sum_{\tau \in \mathcal{D}^{\pi_\theta}} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \sum_{t=0}^{T-1} r_t \right] \\
 &= \frac{1}{|\mathcal{D}^{\pi_\theta}|} \sum_{\tau \in \mathcal{D}^{\pi_\theta}} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \left(\sum_{t'=0}^{t-1} r_{t'} + \sum_{t'=t}^{T-1} r_{t'} \right) \right] \\
 &= \frac{1}{|\mathcal{D}^{\pi_\theta}|} \sum_{\tau \in \mathcal{D}^{\pi_\theta}} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \sum_{t'=t}^{T-1} r_{t'} \right].
 \end{aligned} \tag{3.13}$$

As we saw at the end of Section 3.2.2, it is possible to reduce the variance of the gradient estimator by using a baseline function, $b(s_t)$, without biasing the estimator. However, is the expectation of the score still unbiased in this setting?

$$\nabla_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \left(\sum_{t'=t}^{T-1} r_{t'} - b(s_{t'}) \right) \right]. \tag{3.14}$$

The proof follows a similar argument as shown in Equation (3.6), with the key difference being that the expectation is taken with respect $p_\theta(\tau)$, which is a sequence of random variables. By leveraging the linearity of the expectation property, we can focus on a single term at step t of Equation (3.14) to demonstrate that the baseline does not affect the expectation of the score function. We split the trajectory sequence τ at step t into: $\tau_{0:t}$ and $\tau_{t+1:T-1}$, and then

⁸ The same applies for discounted returns or other kind of returns $R(\tau)$.

expand it into state-action pairs⁹

$$\begin{aligned}
\mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\nabla_\theta \log \pi_\theta(a_t | s_t) b(s_t) \right] &= \mathbb{E}_{\tau_{(0:t)}} \left[\mathbb{E}_{\tau_{(t+1:T-1)}} \left[\nabla_\theta \log \pi_\theta(a_t | s_t) b(s_t) \right] \right] \\
&= \mathbb{E}_{s_{0:t}, a_{0:t-1}} \left[\mathbb{E}_{s_{t+1:T}, a_{t:T-1}} \left[\nabla_\theta \log \pi_\theta(a_t | s_t) b(s_t) \right] \right] \\
&= \mathbb{E}_{s_{0:t}, a_{0:t-1}} \left[b(s_t) \mathbb{E}_{s_{t+1:T}, a_{t:T-1}} \left[\nabla_\theta \log \pi_\theta(a_t | s_t) \right] \right] \\
&= \mathbb{E}_{s_{0:t}, a_{0:t-1}} \left[b(s_t) \mathbb{E}_{a_t} \left[\nabla_\theta \log \pi_\theta(a_t | s_t) \right] \right] \tag{3.15} \\
&= \mathbb{E}_{s_{0:t}, a_{0:t-1}} \left[b(s_t) \nabla_\theta \mathbb{E}_{a_t} \left[\log \pi_\theta(a_t | s_t) \right] \right] \\
&= \mathbb{E}_{s_{0:t}, a_{0:t-1}} \left[b(s_t) \nabla_\theta 1 \right] \\
&= 0.
\end{aligned}$$

We can remove irrelevant variables from the expectation over the portion of the trajectory $\tau_{(t+1):(T-1)}$ because we are focusing on the term at step t . The only relevant variable is a_t , and the expectation $\mathbb{E}_{a_t} \log \pi_\theta(a_t | s_t)$ is 1. Given that the gradient with respect to θ of a constant is zero, and $b(s_t)$ is multiplying it, the effect of the baseline on the expectation is nullified. This argument can be applied to any other term in the sequence due to the linearity of the expectation. Therefore, we have proven that using a baseline also keeps the gradient estimator unbiased in the policy gradient setting.

Choosing an appropriate baseline is a critical decision in reinforcement learning [52], as different methods can offer unique strengths and limitations. Common baselines include fixed values, moving averages, and learned value functions.

1. Constant baseline: $b = \mathbb{E} [R(\tau)] \approx \frac{1}{m} \sum_{i=1}^m R(\tau^{(i)})$.
2. Optimal constant baseline: $b = \frac{\sum_i (\nabla_\theta \log P_\theta(\tau^{(i)}))^2 R(\tau^{(i)})}{\sum_i (\nabla_\theta \log P_\theta(\tau^{(i)}))^2}$.
3. Time-dependent baseline: $b_t = \frac{1}{m} \sum_{i=1}^m \sum_{k=t}^{T-1} R(s_k^{(i)}, a_k^{(i)})$.
4. State-dependent expected return: $b(s_t) = \mathbb{E} [r_t + r_{t+1} + r_{t+2} + \dots + r_{T-1}] = V^\pi(s_t)$.

The control variates method can significantly reduce estimator variance, enhancing the stability and performance of RL algorithms [53]. Despite the nuances and differences among baseline methods, the primary concept is the *advantage*, shown in Equation (3.16), which refers to increase log probabilities of action a_t proportionally to how much its returns, r_t , are better than the expected return under the current policy, which is determined by the value function $V^\pi(s_t)$

$$\mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \underbrace{\left(\sum_{t'=t}^{T-1} R(a_{t'}, s_{t'}) - V^\pi(s_t) \right)}_{\text{advantage}} \right]. \tag{3.16}$$

What remains is how do we get estimates for V^π in practice.

⁹ A criterion used when splitting the trajectory is that state-action pairs are formed given that s_t is a consequence of action a_{t-1} , and taking action a_t results in state s_{t+1} . Notice both expectations from step 1 and 2 in Equation (3.15).

3.4. Actor-Critic Methods

Actor-Critic referred to learn concurrently models for the policy and the value function. This methods are more data efficient because they amortize the samples collected \mathcal{D}^{π_θ} used for Monte Carlo estimations while reducing the variance of the gradient estimator. The actor controls how the agent behaves—*by updating the policy parameters θ as we see in previous sections*—whereas the critic measures how good the taken action is, and could be a state-value (V) or action-value (Q)¹⁰ function. Notice that we are combining in some way both approaches for solving MDPs as is depicted in Figure 3.2.

We are introducing a new function approximator for the value function, $V_\phi(s_t)$, where ϕ are the parameters of the value function

$$\mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \left(\sum_{t'=t}^{T-1} R(a_{t'}, s_{t'}) - V_\phi^\pi(s_t) \right) \right]. \quad (3.17)$$

The objective is to minimize the mean squared error (MSE) between the estimated value and the empirical return, i.e. we are regress the value against empirical return in a supervised learning fashion

$$\phi \leftarrow \arg \min_\phi \frac{1}{|\mathcal{D}^{\pi_\theta}|} \sum_{\tau \in \mathcal{D}^{\pi_\theta}} \sum_{t=0}^{T-1} \left[\left(\sum_{t'=t}^{T-1} R(a_{t'}, s_{t'}) \right) - V_\phi(s_t) \right]^2. \quad (3.18)$$

Algorithm 4 describes the steps for a REINFORCE variant with advantage, which combines the actor-critic approach with the traditional REINFORCE algorithm. More components were introduced and can influence in the performance when the algorithm is implemented. For instance, the policy and value networks can share parameters or not. A useful study that make abalations and suggestions to pay attention when these algorithms are implemented is *What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study* (Andrychowicz, 2020 [54]).

Algorithm 4 REINFORCE with advantage

Initialize policy π_θ

Initialize value V_ϕ

Set learning rates α_a and α_c

for iteration = 0, 1, 2, ..., N **do**

 Collect a set of trajectories $\mathcal{D}^{\pi_\theta} = \{\tau^{(i)}\}$ by sampling from the current policy π_θ

 Calculate the returns $R(\tau)$ for each trajectory $\tau \in \mathcal{D}^{\pi_\theta}$

 Update the policy:

$$\theta \leftarrow \theta + \alpha_a \left(\frac{1}{|\mathcal{D}^{\pi_\theta}|} \sum_{\tau \in \mathcal{D}^{\pi_\theta}} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \left(\sum_{t'=t}^{T-1} R(a_{t'}, s_{t'}) - V_\phi^{\pi_\theta}(s_t) \right) \right] \right)$$

 Update the value:

$$\phi \leftarrow \phi + \alpha_c \left(\frac{1}{|\mathcal{D}^{\pi_\theta}|} \sum_{\tau \in \mathcal{D}^{\pi_\theta}} \left[\sum_{t=0}^{T-1} \left(\sum_{t'=t}^{T-1} R(a_{t'}, s_{t'}) - V_\phi^{\pi_\theta}(s_t) \right) \nabla_\phi V_\phi^{\pi_\theta}(s_t) \right] \right)$$

end for

¹⁰ Action-value function (Q) refers to the value of take action a on state s under a policy π .

3.5. Improving Sample Efficiency: Behavior and Target Policies

The main drawback of the REINFORCE algorithm is its sample complexity. Once we roll out the policy and collect the data, we cannot reuse it after the policy has been updated. We must collect new data following the *target policy* π_θ that we want to update. In RL literature, this is referred to as *on-policy* learning. Reusing the data $\mathcal{D} \sim \pi_{\theta_{\text{old}}}$ to update the current policy π_θ would significantly improve sample efficiency¹¹. However, once we update the policy, the previously collected data is no longer valid because the policy has changed. The distribution from which the data was sampled is now $\pi_{\theta_{\text{old}}}$.

Using behavior data learned from another policy, known as a *behavior policy*, to update the current policy is referred to as *off-policy* learning in RL literature. Let's introduce a *behavior policy* in the RL objective defined in Equation (3.8) using **importance sampling** (See Mckay book, Section 29.2 [55]):

$$\begin{aligned}
 \nabla_\theta \mathcal{J}(\theta) &= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\nabla_\theta \log p_\theta(\tau) R(\tau) \right] \\
 &= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} R(\tau) \right] \\
 &= \int_{\mathcal{X}} p_\theta(\tau) \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} R(\tau) d\tau \\
 &= \int_{\mathcal{X}} \frac{p_{\theta_{\text{old}}}(\tau)}{p_{\theta_{\text{old}}}(\tau)} p_\theta(\tau) \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} R(\tau) d\tau \\
 &= \int_{\mathcal{X}} p_{\theta_{\text{old}}}(\tau) \frac{\nabla_\theta p_\theta(\tau)}{p_{\theta_{\text{old}}}(\tau)} R(\tau) d\tau \\
 &= \mathbb{E}_{\tau \sim p_{\theta_{\text{old}}}(\tau)} \left[\frac{\nabla_\theta p_\theta(\tau)}{p_{\theta_{\text{old}}}(\tau)} R(\tau) \right].
 \end{aligned} \tag{3.19}$$

We derive a new objective that is more general and reconciles both *on-policy* and *off-policy* learning in the importance weight, or importance correction ($p_\theta(\tau)/p_{\theta_{\text{old}}}(\tau)$)

$$\mathcal{J}_{\text{IS}}(\theta) = \mathbb{E}_{\tau \sim p_{\theta_{\text{old}}}(\tau)} \left[\frac{p_\theta(\tau)}{p_{\theta_{\text{old}}}(\tau)} R(\tau) \right]. \tag{3.20}$$

We can assume that the data collected from the behavior policy is not so different from the target policy, and use first order approximation to update the policy

$$\begin{aligned}
 \nabla_\theta \mathcal{J}(\theta)|_{\theta=\theta_{\text{old}}} &= \mathbb{E}_{\tau \sim p_{\theta_{\text{old}}}(\tau)} \left[\frac{\nabla_\theta p_\theta(\tau)|_{\theta=\theta_{\text{old}}}}{p_{\theta_{\text{old}}}(\tau)} R(\tau) \right] \\
 &= \mathbb{E}_{\tau \sim p_{\theta_{\text{old}}}(\tau)} \left[\nabla_\theta \log p_\theta(\tau)|_{\theta=\theta_{\text{old}}} R(\tau) \right].
 \end{aligned} \tag{3.21}$$

The problem with first order approximation. The gradient estimation it is good only

¹¹ This issue also arises when attempting to transfer behavior from one task to another using existing data.

in the immediate vicinity, because is a local approximation of the function. Hence, the step size is crucial to avoid a policy degradation, a situation where the policy is updated with a bad gradient, it is difficult to recover from this situation. Given that the data is collected by the policy, the feedback loop can be dangerous for the training stability.

3.6. Trust Region and Proximal Policy Optimization

Trust Region Policy Optimization (TRPO) [56] allows us to avoid the policy degradation given bad updates. The idea is to define a trust region in which update the policy parameter is safer and balancing the policy improvement with stability

$$\begin{aligned} \text{Surrogate loss: } \max_{\pi_{\theta}} L(\pi) &= \mathbb{E}_{\pi_{\theta_{\text{old}}}} \left[\frac{\pi_{\theta}(a | s)}{\pi_{\theta_{\text{old}}}(a | s)} A^{\pi_{\theta_{\text{old}}}}(s, a) \right] \\ \text{Constraint: } \mathbb{E}_{\pi_{\text{old}}} [D_{\text{KL}}(\pi_{\theta} || \pi_{\theta_{\text{old}}})] &\leq \epsilon. \end{aligned} \quad (3.22)$$

Increase data efficiency while avoiding step size problems in updating parameters, compared to traditional policy gradients (PG). The main idea is to improve a surrogate objective significantly while making minimal changes to the policy. These minimal changes are quantified using the KL divergence between action distributions. The trust region is the area where the new policy remains close to the old one, guarantee training stability.

Proximal Policy Optimization (PPO) [57] is about simplify TRPO in order to (i) be easier to implement avoiding solve the second order optimization in Equation (3.22), (ii) taking advantage of first order optimizer such as ADAM [58], and (iii) be more compatible with neural networks operations such as dropout that are incompatible with TRPO setting.

Let’s rename the importance weights as the probability ratio r :

$$r_t(\theta) = \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}. \quad (3.23)$$

The strategy is to keep this ratio closer to 1. We can create a trust region via clipping the ratio to force within a range $[1 - \epsilon, 1 + \epsilon]$,

$$\mathcal{L}^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right]. \quad (3.24)$$

For a walkthrough implementation that cover important details avoid in the paper and that impact significantly in the performance, review the work “*The 37 implementation details of proximal policy optimization*” (Huang, 2023 [59]).

3.7. Reinforcement Learning From Human Feedback

Reinforcement learning from human feedback (RLHF) is introducing the human within the reinforcement loop, providing the agent the necessary feedback to learn the intended behavior. The idea is learning a reward model that capture the behavior from humans, and use the model to provide the feedback in asynchronous way, which means that during agent

Algorithm 5 Proximal Policy Optimization (PPO), Actor-Critic Style

Initialize policy parameter θ , set learning rate α
Initialize value V_ϕ
for iteration = 0, 1, 2, ... N **do**
 for actor = 0, 1, 2, ... M **do**
 Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
 Compute advantage estimates $\hat{A}_0, \dots, \hat{A}_{T-1}$
 end for
 Optimize surrogate $\mathcal{L}^{\text{CLIP}}$ wrt θ (Equation 3.24), with K epochs and minibatch size $M \leq NT$
end for

training it is not require to wait the human for a respond with the reward¹²; making fully compatible and scalable in the model-free and online learning setting studied in this chapter.

A reward function elicit changes in the system behavior, that is the whole point of reinforcement learning. By trial-and-error, figuring out what is the best behavior that maximize the reward, and by that means, the intended goal. However, the reward function is a key component in the RL setting and is not always easy to design it. Outside video games with clear rules and scenarios, real world is highly complex, erratic, and difficult to simulate. Therefore, more than sometimes align the intended behavior with the reward function is a difficult task. Beyond the realm of RL, use human feedback in this setting provide a useful way to align matching distribution learning systems with the human preferences. Perhaps, the most significant example is in large language models, in which RLHF can mute the behavior of an inherent autocompletion system into a more instructional and conversational behavior [2].

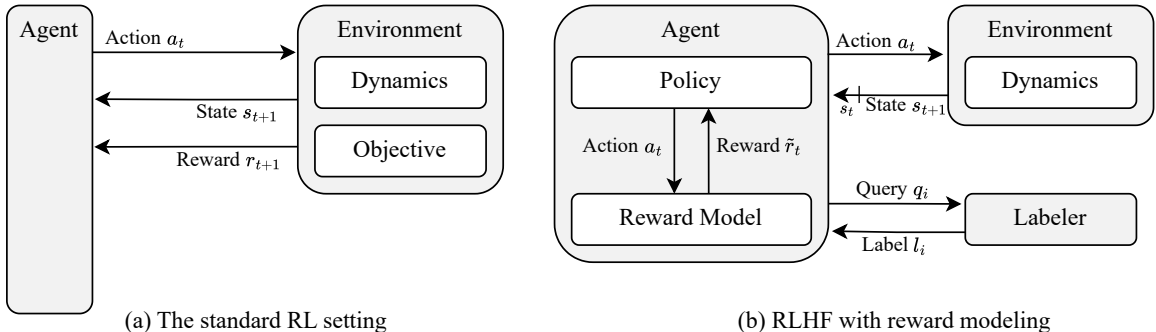


Figure 3.4: **Left:** A standard RL settings. **Right:** A RLHF setting considering reward modeling. **Source:** [A Survey of Reinforcement Learning from Human Feedback \(Kaufmann et al., 2024\) \[23\]](#). Notice how the reward model is decoupled from the environment and the relation highlighted between the reward model and an oracle (i.e. labeler) who provides a label to a given query.

Training the reward model. Translate the human feedback from a collection of trajectory samples into a reward signal. However, the human feedback can be noisy, inconsistent, and

¹² A static reward model is equivalent to extract information in a off-policy fashion. But, the reward model can update dynamically with the agent, taking new information from the on-policy trajectories.

sparse. So, instead of using the human feedback directly, it is used a reward model that learns via an utility function that design to consume different types of feedbacks. For instance, a common setting is a binary comparison between trajectories where a utility function is learned from the human preference between both using the Bradley-Terry model [60]

$$P(\tau_1 \succ \tau_2) = \frac{1}{1 + \exp(R(\tau_2) - R(\tau_1))}.$$

given a collection of trajectories \mathcal{D} where \succ means “preferred to” and $R(\tau)$ corresponds to the utility (i.e. return in the context of RL) and it could be between an intermediate feature maps that allows to the human gives feedback in a useful way but provide a scalar signal easily to digest by the agent. The reward model can be trained using supervised learning, imitation learning, or inverse reinforcement learning. The reward model can be used in different ways, such as reward shaping, reward augmentation, or reward correction

$$\max_{\phi} \prod_{i=1}^N \frac{1}{1 + \exp(R_{\phi})}. \quad (3.25)$$

Several human feedback types can distill into rewards models such as critique, comparisons, inter-temporal feedback, proxy rewards, social behavior, improvements, and natural language [23]. Specifically, for visual perception tasks, some interesting research lines are leveraging general human knowledge from large pretrained visual language models to provide feedback to the agent if it achieve success (i.e. success detectors) on the task in which is reinforced [61]. The main point is that once we have a reward model we can move from a standard RL setting to a RLHF setting, as shown in Figure 3.4. The reward model requires a labeler to encode the human preferences into the model weights, but once it is trained, the human preferences can be use asynchronous and continuously to provide feedback.

3.8. Summary

In this chapter, we have explored the foundational concepts and methodologies in reinforcement learning (RL). The core of RL is the interaction between an agent and its environment, where learning occurs through trial-and-error. The agent’s goal is to maximize cumulative rewards by taking actions based on its observations, influencing the state of the environment, and receiving rewards.

We began by introducing the Markov Decision Process (MDP), a mathematical framework that describes the interaction between the agent and the environment. An MDP is characterized by a state space, action space, transition probabilities, and reward functions. The agent aims to learn a policy that maximizes the expected return, which is the sum of discounted rewards over time.

We then delved into policy optimization methods, focusing on policy gradients, a popular approach in model-free RL. Policy gradient methods reduce RL to a problem of stochastic gradient descent, leveraging trajectories of state-action pairs to update the policy parameters. We discussed techniques such as the reward-to-go and baselines to reduce the variance of gradient estimators, thus improving learning efficiency.

In conclusion, reinforcement learning offers a powerful framework for designing intelligent agents capable of learning optimal behaviors through interaction with their environment. By understanding and implementing the principles and techniques covered in this chapter, one can develop sophisticated RL agents for a wide range of applications

4

Extending Reinforcement Learning in Diffusion Models

Once a diffusion model has been trained, it can generate samples starting solely from noise. The careful process of learning a diffusion chain that removes noise while preserving the structure of the training observations is crucial for producing high-quality samples. The model must be trained using a significant amount of observations at various noise levels managed by the noise scheduler. Ultimately, this results in a multi-step inference process that relies on noise to generate novel samples. As we explored in Chapter 2, this generation process can be modified to exert control over it and even accelerated by skipping steps in the diffusion chain, though this may come at the cost of sample quality.

How does this entire inference process relate to the reinforcement learning concepts of agent and environment discussed in Chapter 3? What would be the purpose of an agent learning a policy in the context of the generation process? How should we understand the objectives and rewards necessary for learning such a policy? These questions will be addressed in this chapter, beginning with an exploration of the intersection between reinforcement learning and pretrained diffusion models. To this end, we will base our approach on the formulation of the diffusion model as a sequential decision-making process, as established in the work *Training Diffusion Models with Reinforcement Learning* (Black et al. [1]).

Next, we will specifically examine two reward functions to contextualize them within the sample generation process of a diffusion model. In particular, we will investigate how noise state trajectories lead to a final sample and how the reward responds at these intermediate states. To do this, we will conduct an empirical analysis of reward signals, evaluating their impact on the intermediate steps of the diffusion process and deriving insights for optimizing generative outcomes.

4.1. Diffusion Model as Sequential Decision-making Process

Let's consider as a starting point a pretrained diffusion model using the DDPM methodology [15] (Section 2.1), from which we can sample via the backward process to generate samples from noise. Recall that the initial state of this Markov process is obtained by sam-

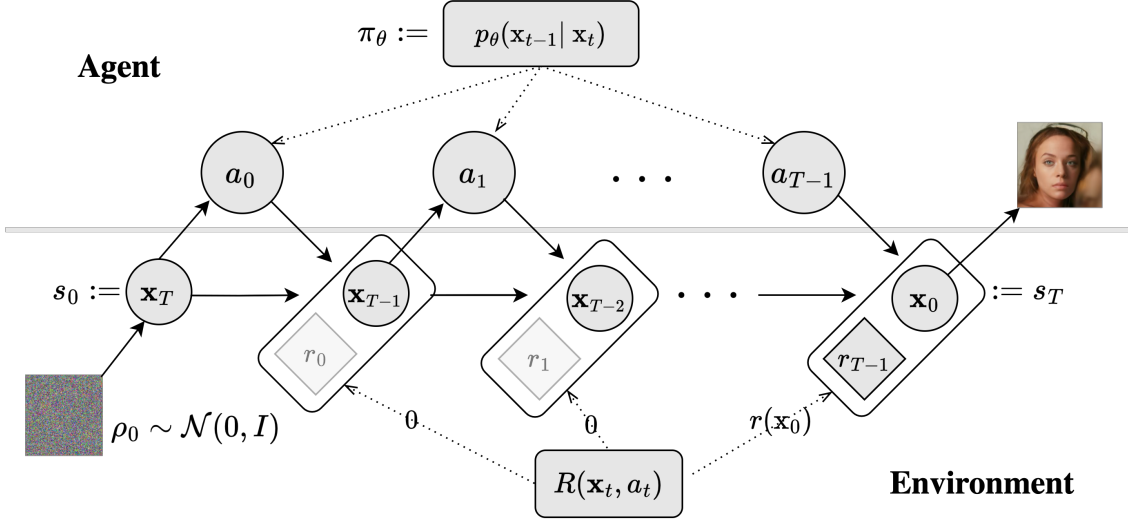


Figure 4.1: **Equivalence of the *backward process* of a diffusion model as a sequential decision-making process.** The initial state distribution of this MDP corresponds to an isotropic Gaussian, $\rho_0(s_0) \sim \mathcal{N}(0, I)$, where we assign the noise instance to the initial state $s_0 := x_T$. The agent follows a sequence of decisions a_t determined by the policy $\pi_\theta(a_t | s_t) := p_\theta(x_{T-t-1} | x_{T-t})$, moving from a noisy state x_{T-t} to a less noisy one x_{T-t-1} until it reaches to the sample x_0 , illustrated as the terminal state s_T in the diagram. This process together with the sample x_0 generates the whole denoising trajectory $\tau = \{x_T, x_{T-1}, x_{T-2}, \dots, x_0\}$, which is associated with a reward. In the case of DDPO, the reward is considered only for the final sample $r(x_0)$.

pling from an isotropic Gaussian distribution $x_T \sim \mathcal{N}(0, I)$, corresponding to the initial noise in the backward process, as illustrated at the beginning of Figure 4.1, where the sample generation starts. A neural network p_θ is then responsible for the denoising process, estimating either x_{t-1} directly or indirectly by estimating the noise $\hat{\epsilon}_t$ used at each step t to corrupt the structure. In either case, once the process is completed, it produces a trajectory of states from the initial noise to the final sample $\tau = \{x_T, x_{T-1}, \dots, x_1, x_0\}$.

Figure 4.1 also illustrates the equivalence of the backward process to a sequential decision-making process, where we establish a correspondence with reinforcement learning concepts to define the MDP described in Section 3.1. The initial state distribution ρ_0 is the simple prior distribution $s_0 \sim \mathcal{N}(0, I)$, from which we learned a process or sampler to transform it into our target distribution $q(x_0)$, a process modeled and learned through DDPM (Section 2.1). Recall that we assume a pre-trained diffusion model as our starting point. Therefore, in this MDP, the initial state is the Gaussian noise $s_0 := x_T$, and the diffusion process acts as the policy controlling the agent’s actions, $\pi_\theta(a_t, s_t) := p_\theta(x_{T-t-1} | x_{T-t})$. In this context, the policy describes how an agent—*through its denoising actions* a_t —moves from a noisy state to a less noisy one (i.e. $a_t : x_{T-t} \rightarrow x_{T-t-1}$) until reaching the final sample x_0 , or the terminal state s_T .

Next, we formalize the MDP: δ_t represents a Dirac delta where the entire probability mass is concentrated at time t , and c refers to a signal that conditions the diffusion model, such as a label or prompt. This formulation is general, and it’s possible to set $c = \emptyset$ to specify an

unconditional model, as shown in Figure 4.1.

$$\begin{aligned}
s_t &= (c, T - t, \mathbf{x}_{T-t}) & \pi(a_t | s_t) &= p_\theta(\mathbf{x}_{T-t-1} | \mathbf{x}_{T-t}, c) & P(s_{t+1} | s_t, a_t) &= (\delta_c, \delta_{T-t-1}, \delta_{\mathbf{x}_{T-t-1}}) \\
a_t &= \mathbf{x}_{T-t-1} & \rho_0(s_0) &= (p(c), \delta_T, \mathcal{N}(0, \mathbf{I})) & R(s_t, a_t) &= \begin{cases} r(\mathbf{x}_{T-t}, c) & \text{if } t = T \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

The purpose of establishing the MDP is to align the pretrained diffusion model with downstream tasks represented by a reward model. In this RL framework, we can directly optimize the diffusion model parameters θ via policy gradient estimation to maximize any arbitrary scalar reward signal over the sample \mathbf{x}_0 . This approach is known as Denoising Diffusion Policy Optimization (DDPO) [1]. In other words, the agent learns how to denoise trajectories to maximize the expected reward using the following denoising diffusion RL objective (DDRL),

$$\mathcal{J}_{\text{DDRL}}(\theta) = \mathbb{E}_{c \sim p(c), \mathbf{x}_0 \sim p_\theta(\mathbf{x}_0 | c)}[r(\mathbf{x}_0, c)]. \quad (4.1)$$

A relevant aspect of this formulation is that the reward $R(s_t, a_t)$ only provides information about the final sample \mathbf{x}_0 , giving zero reward to each non-terminal state, or \mathbf{x}_t where $t \neq 0$ as it is depicted in Figure 4.1. Additionally, DDPO-based methods propose two ways to compute the gradients: (i) via a *score function* method, denoted as DDPO_{SF}, also known as the REINFORCE method derived in Section 3.3,

$$(\text{DDPO}_{\text{SF}}) \quad \nabla_\theta \mathcal{J} = \mathbb{E}_{\mathbf{x}_{T:0} \sim p_\theta} \left[\sum_{t=0}^T \nabla_\theta \log p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t) r(\mathbf{x}_0) \right]. \quad (4.2)$$

and (ii) using *importance sampling* to optimize a surrogate objective [56, 57]. This second approach, denoted DDPO_{IS} in Equation (4.3), is more data efficient because is allowing to update a policy multiple times with the same set of trajectories collected. Given this reasons, in this work any time we use or refer to DDPO without specifying, it is DDPO with importance sampling¹³.

$$(\text{DDPO}_{\text{IS}}) \quad \nabla_\theta \mathcal{J} = \mathbb{E}_{\mathbf{x}_{T:0} \sim p_{\theta_{\text{old}}}} \left[\sum_{t=0}^T \frac{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)}{p_{\theta_{\text{old}}}(\mathbf{x}_{t-1} | \mathbf{x}_t)} \nabla_\theta \log p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t) r(\mathbf{x}_0) \right]. \quad (4.3)$$

There are some points to consider when implement Equation (4.3):

1. For training compatibility with automatic differentiation softwares such as PyTorch or JAX, we transform the RL objective into a loss function \mathcal{L}_{IS} to minimize the negative of the objective,

$$\mathcal{L}_{\text{IS}}(\theta) = \mathbb{E}_{\mathbf{x}_{T:0} \sim p_{\theta_{\text{old}}}} \left[- \left(\sum_{t=0}^T \frac{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)}{p_{\theta_{\text{old}}}(\mathbf{x}_{t-1} | \mathbf{x}_t)} \log p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t) \right) r(\mathbf{x}_0) \right].$$

¹³ Also known as PPO algorithm [57], see Section 3.6.

2. For numerical stability, the probability ratio r_t is computed as follows:

$$r_t(\theta) = \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{p_{\theta_{\text{old}}}(\mathbf{x}_{t-1}|\mathbf{x}_t)}$$

$$r_t(\theta) = \exp(\log p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) - \log p_{\theta_{\text{old}}}(\mathbf{x}_{t-1}|\mathbf{x}_t)).$$

3. For training stability, proximal policy optimization (PPO) implements the above objective by clipping the ratios to avoid large changes in the policy parameters (see Section 3.6),

$$\mathcal{L}_{\text{IS}}^{\text{CLIP}}(\theta) = \sum_{t=0}^T \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right]. \quad (4.4)$$

4. Note that in the previous point, the raw rewards $r(\mathbf{x}_0)$ are not used directly. Instead, \hat{A} , a standardized version of the rewards within the batch, is applied. This standardization helps prevent the gradient estimation from being skewed by the arbitrary magnitude of the reward function.

5. Finally, $\sum_{t=0}^T \log p_\theta$ is straightforward to compute, given that p_θ is a conditional Gaussian distribution with its mean parameterized by θ . Using an automatic differentiation framework, we can efficiently compute and accumulate $\sum_{t=0}^T \nabla_\theta \log p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)$.

Reward Finetuning using DDPO. Figure 4.2 illustrates the “post-training” process¹⁴ for the diffusion model based on a reward signal, defined by a reward model R , referred to as *reward finetuning*. All previous components are integrated so that the diffusion model becomes an agent that learns to denoise trajectories in order to maximize the expected reward. In the diagram, we use only the forward indexing of the MDP from $t = 0, \dots, T$, to avoid confusion with the reverse indexing of the diffusion backward process steps.

Since this problem is approached through model-free reinforcement learning—*where we do not aim to learn a model of the environment*—the goal is to maximize the policy (the generation process) based solely on the experience of generating samples. This will allow us to obtain samples that, on average, have a higher reward and are aligned with our objectives. Therefore, the process is divided into two main stages:

1. **Dataset Collection:** Collect a dataset \mathcal{D}^{π_θ} of samples using the diffusion model as policy rollouts. This involves recording the trajectories of states and obtaining the rewards associated with the final sample \mathbf{x}_0 using a reward model R .
2. **Reward Finetuning:** use the dataset \mathcal{D}^{π_θ} to estimate gradients based on the collected samples and use them to update the model parameters.

The diagram also shows the case where we decide to reuse the collected samples multiple times instead of using them just once for a policy update (see diamond shape). It’s important to consider that the dataset is inherently non-stationary since it comes from a policy whose parameters are being continuously updated. When reusing samples, the dataset comes from an older policy $\pi_{\theta_{\text{old}}}$, which can lead to policy degradation if the gradient estimate is not valid

¹⁴In this work, “post-training” and “adjustment” process are used synonymously to refer to the model’s parameter update process during finetuning.

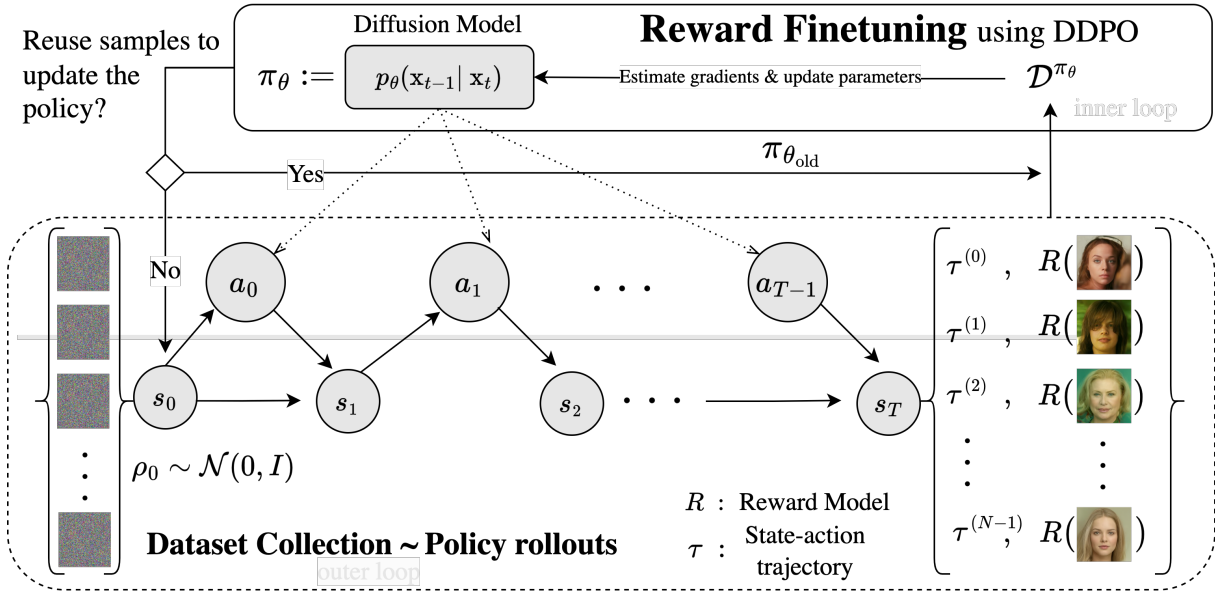


Figure 4.2: **Reward finetuning diagram using DDPO.** The finetuning process occurs in two stages: (i) collecting a dataset \mathcal{D}^{π_θ} of samples by using the diffusion model as policy rollouts, and (ii) using \mathcal{D}^{π_θ} to estimate the gradient of the parameters $\nabla_{\theta} \mathcal{J}$, with objectives such as DDPO_{SF} , and applying a first-order optimization algorithm (e.g., gradient ascent) to update the model. Given that the dataset originates from a non-stationary distribution (changing policy), if we want to reuse the samples $\mathcal{D}^{\pi_{\theta_{\text{old}}}}$ to adjust the parameters more than once, we need to use the DDPO_{IS} objective.

for an *on-policy* setup. For this reason, it’s crucial to use DDPO_{IS} in these cases to avoid the issues discussed in Section 3.5.

Lastly, the reward model has so far been considered as a given, available for evaluating the samples x_0 and obtaining the rewards necessary to align the model’s sample generation with a specific downstream task. The acquisition and role of the reward model follow the principles outlined in Section 3.7 on RLHF, but also apply to any function. In the next section, we will further explore the behavior of the reward and its mechanics within the sample generation process. To study these mechanics, it is essential to instantiate a pretrained diffusion model and a concrete reward model, allowing us to illustratively examine the key components driving the reward finetuning process explained in this section.

4.2. Empirical Analysis on Reward Trajectory Dynamics

Is the reward behavior informative during sample trajectories? In the previous section, we observed that DDPO relies solely on the reward from the final sample. In this section, we explore the potential of the reward signal from intermediate diffusion states, which could provide valuable insights for extending the DDPO framework to utilize the reward signal throughout the entire sample trajectory. This is crucial when applying the reward function in the context of diffusion models, as it should effectively guide the agent to maximize the reward. We present an empirical analysis of the reward signal dynamics during the sample generation process. To do so, we use the unconditional pretrained diffusion model

[google/ddpm-celebahq-256](#) [15], trained on the CelebA-HQ dataset [62], to compile a set of trajectory samples, denoted as \mathcal{S}_o .

The 1000 observations in \mathcal{S}_0 comprises a summary of six intermediate states extracted from each trajectory. We sampled from the model using a 40-step subsequence τ of the diffusion chain using the `DDIMScheduler` [16], speeding up the sampling process over the original diffusion chain of $T = 1000$ steps, as we explained in Section 2.7. To reduce the computational cost, only six intermediate state of the subsequence τ are saved, and the equivalence of this summary states $x_{\tilde{t}}$ with the original DDPM steps are given in the following tuples: $(\tilde{t} = 0, t = 1000)$, $(\tilde{t} = 1, t = 975)$, $(\tilde{t} = 2, t = 725)$, $(\tilde{t} = 3, t = 475)$, $(\tilde{t} = 4, t = 225)$, $(\tilde{t} = 5, t = 0)$. Therefore, the trajectories are describe starting from the Gaussian noise, then jumping to the step 975, 725, until the final sample $x_{t=0}$, or $x_{\tilde{t}=5}$.

The analysis include two different reward functions. The first one is the LAION aesthetic predictor [24], a multilayer perceptron (MLP) that assigns a scalar value from 1 to 10 to indicate the aesthetic quality of an image, and it was trained based on human preferences. The second reward function is the image size after JPEG compression algorithm, which is used to define two downstream tasks: compressibility and incompressibility. For compressibility, we want to maximize the negative size of the image after compression, which is equivalent to minimizing the size of the image after compression. The opposite is to maximize the size of the image after compression, which we refer to as incompressibility. The election of these reward functions were based to match the experimental pipeline use in the work that introduces DDPO [1].

Computing the reward over each trajectory on \mathcal{S}_0 allows us to inspect the reward behavior over the sample trajectories, as shown in the top row of Figure 4.3. For the aesthetic quality (left), it is possible to observe that from the step $t = 725$ ahead, the reward signal starts to increase in average (orange line) and a higher variance in the reward signal is observe as the trajectory approach to the final sample x_0 . During approximately the first 25% of the denoising process, the signal is concentrate with minor variation between a 3.5 aesthetic score. That is close to the aesthetic score that we obtain when we apply the LAION aesthetic predictor to a purely Gaussian noise. At the end, we achieve a 5.11 average (orange line) aesthetic score, a 6.22 and 3.86 corresponding to the best (green line) and worst (red line) aesthetic score.

For the image size after JPEG compression (right), the reward signal is more stable over the trajectory, and the variance is lower than the aesthetic quality reward signal. The average image size of the final samples after JPEG compression is 17.26 kilobytes (orange line), and the best (green line) and worst (red line) filesizes are 34.38 and 4.07 kilobytes, respectively. The results show that the reward signal start to be some informative on intermediate states from the step $t = 475$ ahead, but with a lower variance than the aesthetic quality reward signal. The reason is that the noise is difficult to compress, and the JPEG compression can start to have effects on the image size reduction when the semantic of the image start to appear and the entropy is reduce.

The top row of Figure 4.3 shows the rewards computed directly over the intermediate states, i.e. structure plus noise. However, there is an issue in doing this, and could be the reason

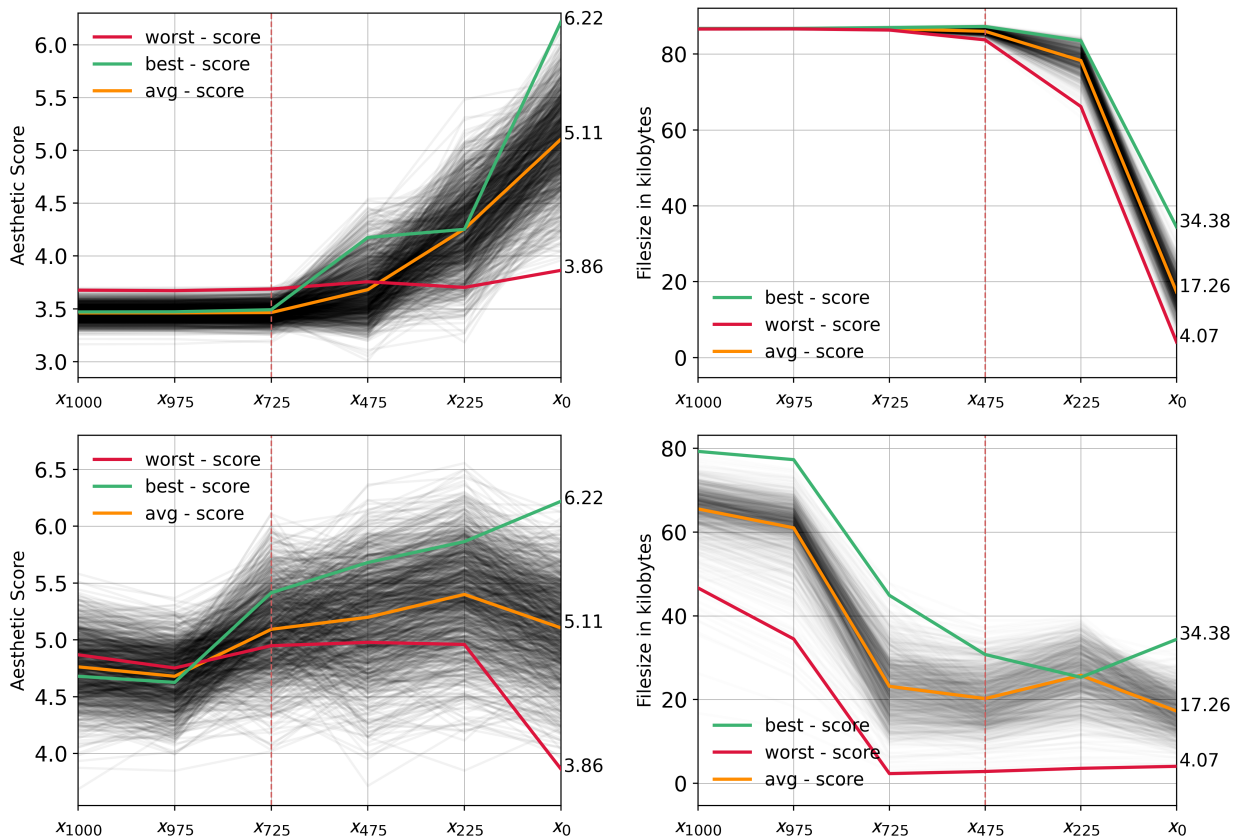


Figure 4.3: **Visualizing reward signal during sample trajectories.** **Left:** Evolution of the aesthetic score as reward signal over the six states x_t , summarizing each of the 1000 trajectories (\uparrow better). **Right:** Image size in kbs after JPEG compression, providing another form of reward signal for the same set of trajectories (\downarrow better). **Top:** Rewards computed over the noisy intermediate states x_t . **Bottom:** Rewards computed over the denoised states \tilde{x}_t .

of why is a good idea to only compute the reward on the final samples x_0 , as is illustrated in the markov decision process formulation for a diffusion a model in Figure 4.1. The reward function to control the model is designed to maximize the reward in the final sample, which is a state that can be directly guide by the human preferences. Of course that we can provide human preferences between noisy states, but nothing guarantees that the denoising trajectories will be consistent in the following states. Similar with the JPEG compression, the goal is to generate images of lower size, and this endeavour has nothing to do with the noise involved in the sample generation process. Therefore, we have a communicational gap between the intermediate states and the reward signal, a similar problem that occur using classifier guidance with a model that is not robust to the noise to guide the generative process [43].

Denoised sample trajectories. We will use the denoised observations $\tilde{x}_{t \rightarrow 0}$ for the initial noise and each intermediate step t of the sample trajectories, as referred in the DDIM Section 2.7:

$$\tilde{x}_{t \rightarrow 0} = f_{\theta}^{(t)} = \frac{x_t - \sqrt{1 - \alpha_t} \epsilon_{\theta}^{(t)}(x_t)}{\sqrt{\alpha_t}}. \quad (4.5)$$

As it can be seen in Figure 4.4, the denoised samples $\tilde{x}_{t \rightarrow 0}$ are fairly similar to the final sample.

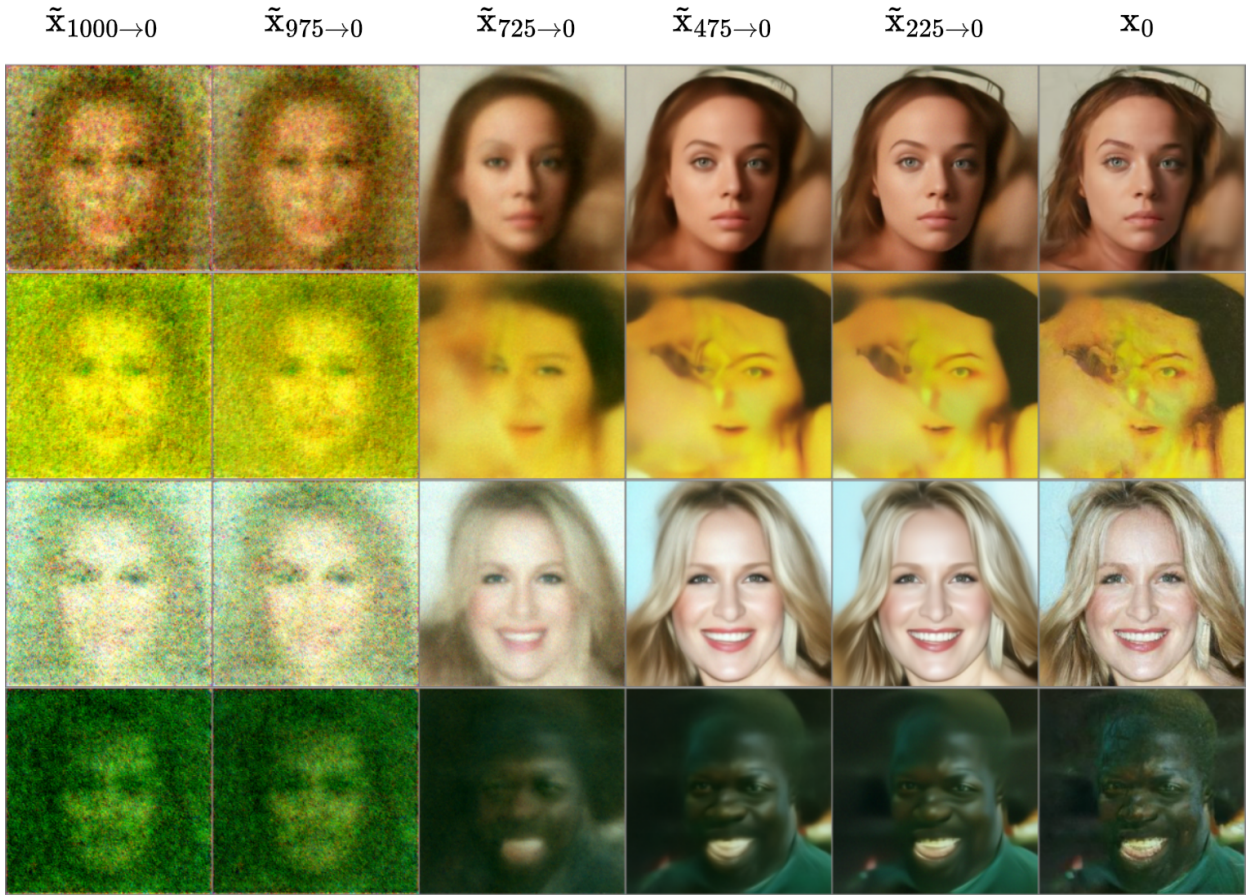


Figure 4.4: **Denoised sample trajectories.** Each row, from **top-to-bottom**, represents the best (6.22) and worst (3.86) aesthetic scores, and the highest (34.38) and lowest (4.07) filesizes in kilobytes after JPEG compression for the final samples x_0 in \mathcal{S}_o . Each column, from **left-to-right**, summarizes the states for each denoised sample’s trajectory using Equation 4.5. The rows correspond to the trajectories highlighted in the green and red lines of Figure 4.3.

The denoised intermediate state $\tilde{x}_{725 \rightarrow 0}$ almost captures the essence of the final sample x_0 , and $\tilde{x}_{1000 \rightarrow 0}$ is a highly informative latent encode of the high level features such image pose, gender, hair style and colors of the final image. Notice that in the same point in Figure 4.3 (bottom-left), we have an increasing dispersion in the aesthetic score trajectory distribution from the start. This is a clear indication that the reward signal is more informative in the denoised samples than in the noisy samples. The same behavior is observed for the image size after JPEG compression (bottom-right), where the variance is higher in the denoised samples than in the noisy samples. That is because the JPEG compression knows in advance which trajectories contain features that can be compressed or not.

4.3. Experiments

We aim to reproduce the implementation outlined in *Training Diffusion Models with Reinforcement Learning* [1] within a more streamlined setting to facilitate experimentation, while still capturing the complexity of text-to-image models for experimenting with reward functions. Our baseline will be the generative capabilities of the pretrained [google/ddpm-](#)

[celebahq-256](#) and [google/ddpm-church-256](#) models, both trained using the DDPM methodology described in Section [15]. These are unconditional models that generate 256×256 pixel RGB images of human faces and churches, respectively. Both models operate in the pixel space, unlike latent diffusion models such as Stable Diffusion [28], in which the denoising process occurs in the latent space modelled by a variational autoencoder instead.

Evaluation Dataset. We use the final samples of each trajectory in \mathcal{S}_0 —described in Section 4.2—as a baseline for assessing the generative performance of the pretrained model on downstream tasks. In the following chapter, we evaluate the experiments using the same seeds to generate samples with finetuned checkpoints, and report the mean and standard error of the rewards across the evaluation samples. These metrics serve as the quantitative evaluation criteria in this work, enabling us to compare the performance of the finetuned models against the pretrained one.

In our experiments, we employed three downstream tasks as outlined in (Black, 2023 [1]): JPEG compressibility and incompressibility, and aesthetic quality. The first two tasks are defined by the size of images after applying a JPEG compression algorithm, serving as the reward function. For *compressibility*, we want to maximize the negative size of the image after compression. This is equivalent to minimizing the size of the image after compression. The opposite is to maximize the size of the image after compression, which we refer to as *incompressibility*. For *aesthetic quality*, we utilized the LAION-Aesthetic Predictor V2 model [24], a multilayer perceptron that assigns a scalar value from 1 to 10 to indicate the aesthetic quality of an image.

These tasks effectively demonstrate the flexibility of using reinforcement learning to learn new downstream objectives. Supervised learning finetuning often struggles to encode tasks like compressibility and incompressibility into a loss function, whereas RL can optimize these tasks directly through a reward function as shown in Figure 4.2. Additionally, the LAION aesthetic model, trained on human preferences, exemplifies how human feedback can be incorporated to align sample generation with desired outcomes. This is a key advantage of using RL with generative models [2].

5

Results on Reward Finetuning using DDPO

This chapter employs the previously explained DDPO method to finetune pretrained models based on reward functions (Figure 4.2). We present results obtained from three image generation tasks, utilizing two different pretrained models: [google/ddpm-celebahq-256](#), which specializes in generating celebrity-like faces and is used as an example throughout this work, and [google/ddpm-church-256](#), which specializes in church generation.

The reward functions used are JPEG compressibility and incompressibility, as well as aesthetic quality using the LAION-Aesthetic Predictor V2 model [24] as a reward function. These three reward functions are presented in the work of Black et al. [1] and serve as a reference for comparing the results obtained in this study.

Additionally, we design and implement a novel reward function called **OVER50** for the case of face generation. This function aims to increase the proportion of faces of people over 50 years old in a model that initially has a very low frequency of generating images with these characteristics. To achieve this objective, we use the logits from the predictions of an age classifier for human faces [25]. This approach demonstrates the flexibility of reinforcement learning for finetuning models while also pushing its limits in tasks where supervised finetuning might be more effective. However, the RL approach offers the advantage of using synthetic data generated by the model itself, rather than incurring the cost of collecting facial images of people with these characteristics.

Table 5.1 provides a comprehensive summary of the performance of models finetuned with DDPO in image generation tasks, comparing them to the initial performance of pretrained models. The results demonstrate that DDPO finetuned models consistently outperform their initial counterparts across all tasks—*an expected outcome considering the reinforcement learning maximization process*—in terms of average reward over evaluation samples. These improvements hold true even when accounting for estimator variability, which is presented alongside the average for each task.

Downstream Task	Baseline	DDPO
google/ddpm-celebahq-256		
Aesthetic Score (higher is better)	5.11 \pm 0.01	5.58 \pm 0.01
Compressibility (lower is better)	17.26 \pm 0.15	6.01 \pm 0.13
Incompressibility (higher is better)	17.26 \pm 0.15	21.6 \pm 0.12
Over 50 years old (higher is better)	-7.72 \pm 0.17	7.39 \pm 0.16
google/ddpm-church-256		
Aesthetic Score (higher is better)	4.77 \pm 0.01	5.13 \pm 0.01
Compressibility (lower is better)	29.57 \pm 0.29	10.62 \pm 0.18
Incompressibility (higher is better)	29.57 \pm 0.29	50.21 \pm 0.34

Table 5.1: **Mean and standard error for each downstream task across two pretrained models.** All samples were generated using the same initial noise to ensure a fair comparison. **Baseline** refers to the generative capabilities of the pretrained model, as represented by \mathcal{S}_0 (see Section 4.2). **DDPO** displays results from the finetuned models using DDPO with importance sampling (see Section 4.1).

The code developed for finetuning the models and obtaining the presented results is available in the accompanying [code repository](#). Detailed logging information for each experiment can be found in Table A.1 of the appendix. Additionally, model checkpoints have been uploaded to [Hugging Face](#) to facilitate reproducibility and allow for further exploration of the experimental results. Let’s begin by analyzing in detail the results of the face generation model adapted to each of the tasks.

5.1. Reward Finetuning on Face Generation

A set of images generated under the same initial noise by the pretrained model and the finetuned models are provided in Figure 5.1 for an overview and quality comparison. In most cases, the general semantics of the faces are preserved, still identifying the same subject depicted in the original sample. However, DDPO induces modifications in some high—*and* low—level features to achieve the reward objective.

For instance, in the second sample of the first row (from left to right), the pretrained model generates an image of a woman with blue eyes, while the compressibility panel shows a man wearing blue headwear. In terms of high-level features, both images maintain a neutral facial expression, but the apparent gender shifts from female to male. Additionally, low-level features, such as pixel colors, are darker in the compressibility version, which also induces high-level changes like a transition to darker skin tones.

In the aesthetic quality panel, we observe that the samples are less diverse compared to those in other panels. Additional comparative samples like these are available in Appendix B. Next, we will explore in detail each of the experiments that involved adapting the model to the downstream tasks shown in the panels of Figure 5.1.

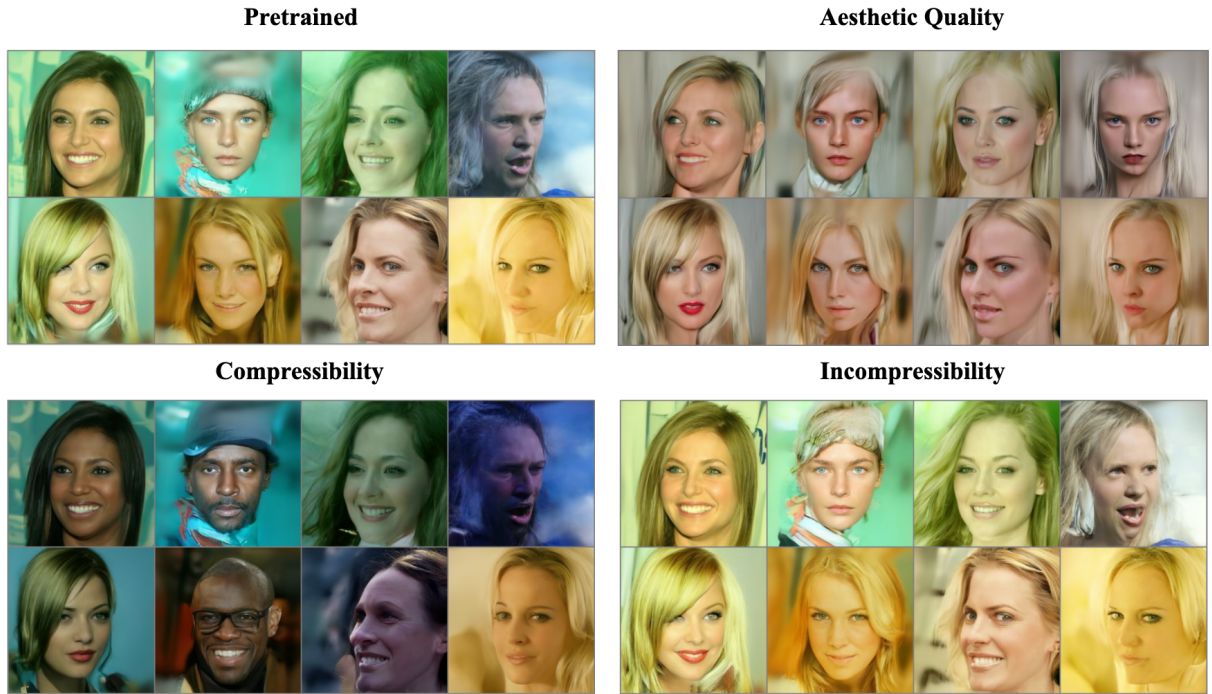


Figure 5.1: **Qualitative comparison of the effects of DDPO finetuning versus the pretrained model.** The top-left panel shows samples from the DDPM pretrained model [google/ddpm-celebahq-256](#). The other panels display samples generated from the same initial noise, but using models finetuned with DDPO for different reward functions: aesthetic quality (top right), JPEG compressibility (bottom left), and incompressibility (bottom right). Additional samples are provided in Appendix B.

5.1.1. JPEG Compressibility

Emergent effects in face generation using JPEG compressibility as reward. Certain effects emerge as the entropy of the images is reduced, allowing for greater compression and reduced file size. Some of the recurring effects during the adjustment process include the elimination of details such as hair definition, reduction in lighting, simplification of image backgrounds, appearance and intensification of shadows, increased intensity around the contours of the eyes, and enhanced facial depth, among others.

In Figure 5.2, four panels display images generated by the pretrained model on left, compared to those generated by the finetuned model with DDPO to maximize compressibility on the right. The effect of *melanotropism* is observed, with an increased frequency of faces with darker skin tones, as shown in the top panels. In some cases, more serious facial expressions, along with other described effects, give a more *realistic appearance*, as seen in the bottom panels. Instances of *transmasculinity* are also common in the images. The hypothesis for this phenomenon suggests that a combination of hair detail loss and shading produces short hairstyles, either through cropped hair or an emphasis on short hair, steering the generation process towards regions more likely to produce masculine features, as illustrated in the top-left and bottom-right panels of Figure 5.2.

Table 5.1 reports the file sizes of images generated by the pretrained model and those generated by the model finetuned with DDPO. The pretrained model generates images with an

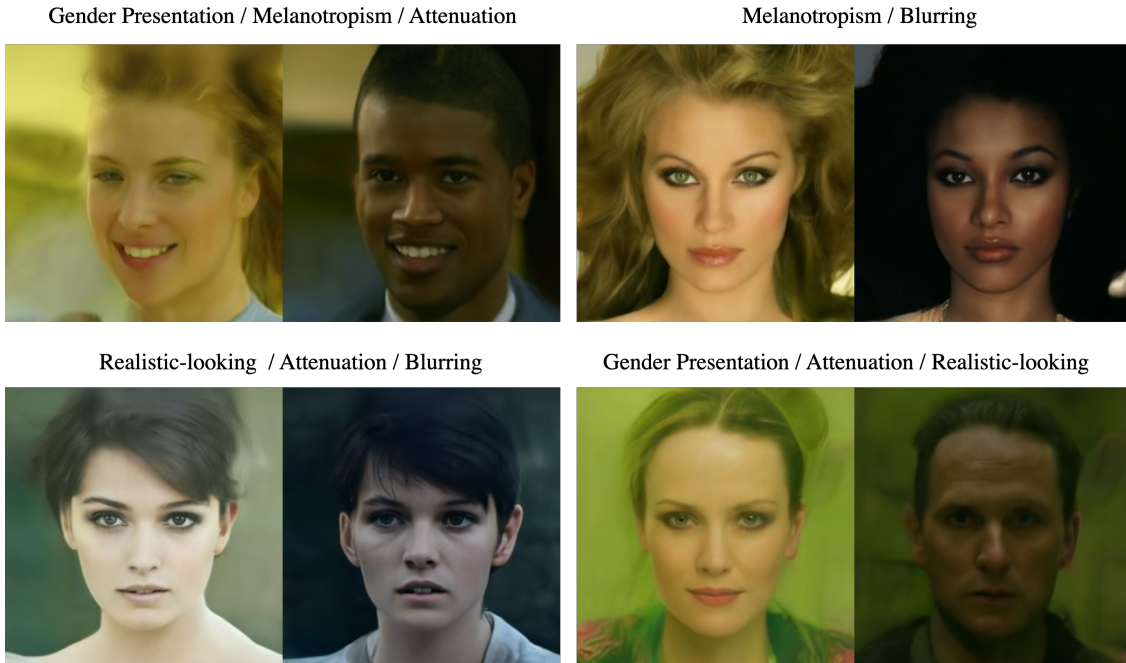


Figure 5.2: **Emergent effects in face generation using JPEG compressibility as a reward.** Each panel compares a pretrained model sample (left) with its finetuned version (right). Notable changes include melanotropism, realistic facial expressions, and altered gender presentation due to hair detail loss and shading effects.

average size of 17.26 kilobytes, while the model finetuned with DDPO to maximize compressibility reduces the file size to an average of 6.01 kilobytes, representing almost a $3\times$ reduction. Additionally, the reward curve for the samples generated during the post-training process is presented, showing how the model specializes in generating smaller images on average (Figure 5.3, blue), and how the dispersion of rewards in the samples collected at each epoch decreases as the model adapts to the task of maximizing compressibility. Finally, Figure 5.4 shows the evolution of an image during the finetuning process, where smooth transitions occur between each image due to the low learning rate used (see Appendix A), despite the occurrence of the transmasculinity phenomenon.

5.1.2. JPEG Incompressibility

Emergent effects in face generation using JPEG incompressibility as reward. The effects produced by maximizing the image size using DDPO are, in some ways, the opposite of those observed when maximizing compressibility. In this case, generating images with larger file sizes is achieved through the addition of details and overall increase in brightness. Some recurring effects during the finetuning process include adding details such as hair volumization & definition, lightening of hair tones, increasing the complexity of image backgrounds, exploiting certain artifacts in the image, removing shadows, appearance or intensifying of makeup.

The results of maximizing image size after JPEG compression—*essentially making the generated images harder to compress*—are presented in Table 5.1. The pretrained model generates

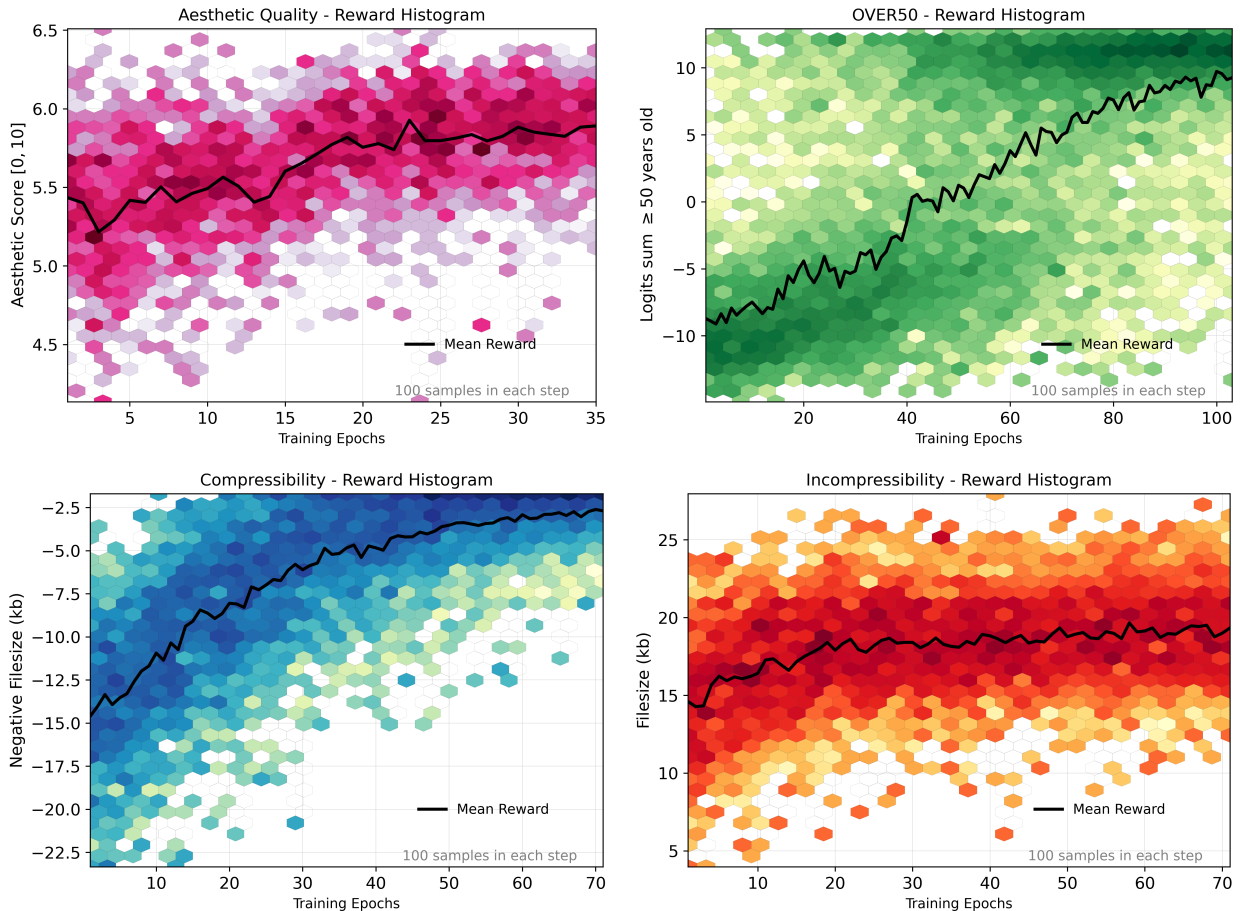


Figure 5.3: **Learning curves from DDPO finetuning on face generation tasks.** The evolution of the mean reward (black line) and the reward distribution (hexbin) are shown for each *downstream task*. The reward estimates were computed in each step using 100 samples from the model. The top-left panel shows the learning curve for the LAION aesthetic quality reward (aesthetic score), the top-right panel for the OVER50 reward (sum of logits for relevant classes), the bottom-left panel for JPEG compressibility (negative file size in kB), and the bottom-right for JPEG incompressibility (file size in kB).

images with an average size of 17.26 kilobytes, while the DDPO model, finetuned to maximize incompressibility, increases the file size to an average of 21.6 kilobytes, a $1.25\times$ increase. Additionally, the reward curve for the samples generated during the finetuning process is shown, indicating how the model specializes in generating larger images on average (Figure 5.3, red). However, while the dispersion of rewards in the samples collected at each adjustment step decreases, incompressibility is less effective than compressibility. Figure 5.6 shows the evolution of an image during the finetuning process, highlighting effects such as increased lighting, reduced shadows, greater hair definition, make-up intensity in the lipstick, and the preservation of image details. Notably, elements like earrings and background artifacts, which emerge during the post-training process, become more exploitable as the model learn to emphasize features that are difficult to compress.

It is interesting to compare the learning dynamics of both rewards using the JPEG compression algorithm. While they represent two sides of the same coin, achieving similar magnitudes



Figure 5.4: **Sample transformation when optimizing the pretrained [google/ddpm-celebahq-256](#) model using JPEG compressibility as the reward function.** The top-left image shows a sample generated by the pretrained model. Moving from left to right and top to bottom, the sample is regenerated from the same initial noise after each update of the model’s parameters. The final result of the finetuning process is shown in the bottom-right image.

for both is not equally straightforward. Once achieves a $3\times$ reduction, while the other only a $1.25\times$ increase. The learning dynamics for the incompressibility reward are inherently more challenging than for compressibility. This is reflected in the reward curves and sample histograms during post-training, as mentioned earlier (Figure 5.3, bottom row).

Upon further reflection, this makes sense given the model’s capabilities and the nature of the task. Adding more information demands greater “generative capacity” to incorporate visual semantics and other features into the image that remain intact during the compression process. The limitation of our generative model is that it is constrained to generating human faces rather than more complex objects or scenes. In contrast, reducing the file size can always be achieved by degrading the model’s generative capacity—*removing information is easier than adding or creating it.*

5.1.3. Aesthetic Quality

Improving aesthetic quality using RLHF. Enhancing the aesthetic quality of an image is an objective that can be achieved using reinforcement learning. However, aesthetic quality is highly subjective, depending on personal preferences and cultural or tempo-



Figure 5.5: **Emergent effects in face generation using JPEG incompressibility as a reward.** Each panel compares a pretrained model sample (left) with its finetuned version (right). Notable changes include increased hair volume, hair definition, skin tone lightening, and overall increased illumination with reduced shadows.

ral influences. In situations where a clear reward function is not easily formulated, human preferences can guide the training of a generative model. Section 3.7 explains how human preferences can be captured in a reward model and used as an oracle to generate the reward signal for images, thus guiding the generation process to maximize the desired attribute.

The pretrained [google/ddpm-celebahq-256](#) was finetuned using the LAION- Aesthetic Predictor V2 model [24] to enhance the aesthetic quality of generated images, reproducing the downstream task from the reference study with DDPO [1]. This predictor, trained on human preferences with scores ranging from 1 to 10, where 10 represents the highest aesthetic quality¹⁵. Post-training began with generating a set of images to evaluate their aesthetic scores. Subsequently, the model was finetuned with DDPO to maximize aesthetic quality. The results, presented in Table 5.1, show that the pretrained model had an aesthetic score of 5.11, while the DDPO-finetuned model achieved a score of 5.58 on the evaluation set. Training dynamics are illustrated by the average reward trajectory of the samples used for finetuning, demonstrating the model’s increasing specialization toward its goal (Figure 5.3, purple). Each iteration produced a new set of images that, on average, displayed higher aesthetic scores and lower reward dispersion compared to previous ones, providing strong evidence that the reinforcement learning objective of maximizing the reward was successfully achieved.

A significant difference compared to parameter tuning in previous tasks is the requirement for a more complex training dynamic to achieve higher rewards. In previous tasks, a learning rate

¹⁵ On this [website](#), you can find reference images was used to assess the model’s initial capability



Figure 5.6: **Sample transformation when optimizing the pretrained [google/ddpm-celebahq-256](#) model using JPEG incompressibility as the reward function.** The top-left image shows a sample generated by the pretrained model. Moving from left to right and top to bottom, the sample is regenerated from the same initial noise after each update of the model’s parameters. The final result of the finetuning process is shown in the bottom-right image.

of 9×10^{-8} was used, but maximizing aesthetic quality **did not show an increase in rewards** with the same computational budget and a learning rate of 7×10^{-8} . On the other hand, increasing the learning rate too high (e.g., 1×10^{-5}) resulted in degraded generated images. Therefore, we implemented a linear warm-up for the learning rate, followed by a half-cosine scheduler to control and adjust the learning rate during training. This started with a learning rate of 9×10^{-8} , linearly increasing to a peak of 3.74×10^{-5} within the first 25% of the training steps. During the remaining 75% of the training, the half-cycle cosine scheduler reduced the learning rate to 9×10^{-9} (see Figure 5.7). This training dynamic allowed the use of a higher learning rate without destroying the model’s generative capabilities. Consequently, the modifications resulted in more substantial semantic variations in the images to maximize the aesthetic score without compromising generative capacity, as seen in the transition from a pretrained model sample to the final checkpoint in Figure 5.9. This was achieved in fewer epochs than those used in previous tasks (about half), as you can see in the learning curves of Figure 5.3.

Emergent effects in face generation using the LAION aesthetic predictor as a reward. The effects observed in images generated by the DDPO-finetuned model to maximize the “aesthetic quality” of generated faces reflect much of the human preferences on which the LAION predictor was trained. The generated images tend to be predominantly

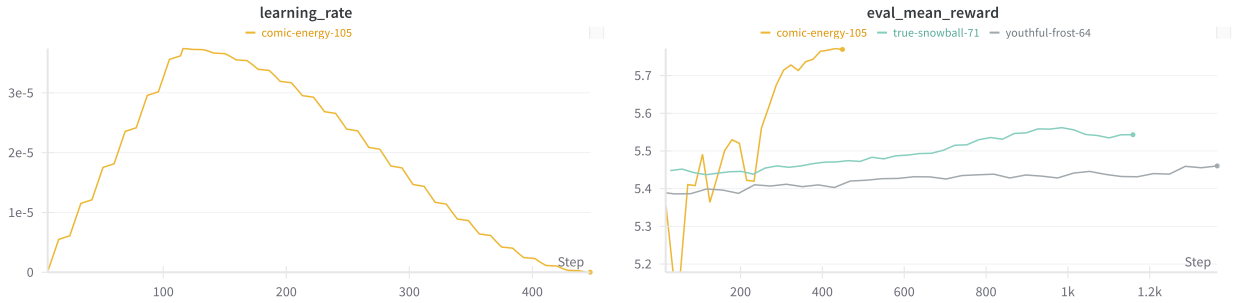


Figure 5.7: **Aesthetic quality training dynamics.** **Left:** The learning rate scheduler with a linear warm-up reaching a peak learning rate of 3.64×10^{-5} at the first quarter of training, followed by a half-cosine decay for the remaining three-quarters. **Right:** Mean aesthetic score on the evaluation set during training. The yellow line corresponds to training with the learning rate schedule described on the left, while the other lines represent training with fixed lower learning rates of 7×10^{-8} and 9×10^{-8} . Aesthetic quality requires a more complex training dynamic to achieve higher rewards.

female and appear younger. Figure 5.8 showcases these emergent effects at the right side of each of the four panels, contrasting them with versions produced by the pretrained model on the left. The second row panels displays two instances of the rejuvenation effect, with the bottom-right panel also undergoing a gender change. Warmer colors are also predominant, a phenomenon expected when compared to example images from the study introducing DDPO [1]. The hypothesis behind this warmth and sketched appearance effect is due to the preference for portraits or illustrations, which tend to be better evaluated and achieve higher aesthetic scores, thus inducing these attributes during the finetuning process. Another emergent effect is the intensity of the gaze, achieved not only through intensified eye makeup but also the *camera position* that captures the face, resembling a magazine cover image. Figure 5.9 provides evidence of the transition from a sample of the pretrained model to the final checkpoint, where many of the described effects emerge, such as blondification, profiling, and rejuvenation.

5.1.4. OVER50

Generating faces of older people. A new task is incorporated to demonstrate the flexibility of reinforcement learning in defining objectives. The idea is to generate faces of people over 50 years old using images generated by a pretrained model as a starting point. The frequency of faces of people in this age group is 6.1%, as shown in the distribution on the left in Figure 5.10.

It is important to note that this task is better suited to be solved through traditional finetuning, using the same cost function employed by the pretrained model we discussed in Section 2.3, applied to a set of collected images of older people’s faces. However, the aim here is to demonstrate that RL can achieve this goal using only synthetic images generated by the same model. Although this is not the most optimal approach, it serves as an interesting exercise to explore how even a model with a low probability of generating synthetic data with the desired attribute, such as faces of people aged 50 or older, can still produce results using the same methodological framework used in previously discussed tasks.

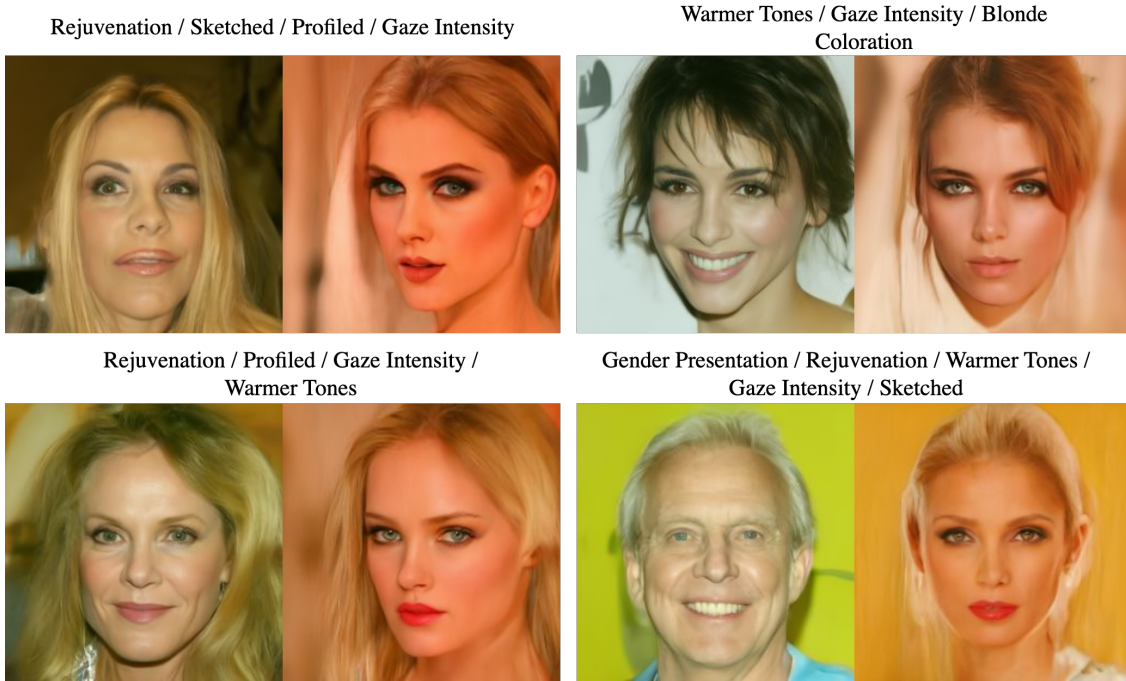


Figure 5.8: **Emergent effects in face generation using LAION aesthetic predictor as a reward.** Each panel compares a pretrained model sample (left) with its finetuned version (right). Notable changes are increased prevalence of female faces, a younger appearance, warmer tones, and more intense gazes, reflecting aesthetic preferences learned by the LAION predictor.

Using a classifier to design a reward function. A classifier is used to detect age from facial images, trained on the FairFace dataset [25, 63]. The model gives the probability that the face belong to age classes 0–2, 3–9, 10–19, 20–29, 30–39, 40–49, 50–59, 60–69, and more than 70. The reward design involves simply summing the logits of the age classes of 50 years or older, and by using logits, this goal is achieved while also being more numerically stable than using the final probabilities of each class. This approach rewards the diffusion model for generating faces of older people and disincentivizes the generation of samples with a low sum of logits in the classes of interest. We are implicitly conditioning the model to generate faces of older people by not providing direct age information as a label¹⁶.

The results are presented in Table 5.1, where the pretrained model reports a logit sum of -7.72 —*the sum of the logits of the relevant classes is considered*—and the model finetuned with DDPO achieves a logit sum of 7.39. It is observed that the sample trajectories increase the average reward while their dispersion decreases, indicating that the model is specializing in its objective (Figure 5.3, green). The age distribution of the finetuned model is shown on the right in Figure 5.10. An increase in the proportion of faces of people aged 50 or older is noted, from 6.1% in the baseline to 78.7% in the finetuned samples. Finally, the transition of a sample from the pretrained model to the final checkpoint is provided as evidence in Figure 5.11.

¹⁶ An interesting experiment would be to adapt classifier-free guidance training [44] to make the conditioning explicit while still using RL.



Figure 5.9: **Sample transformation when optimizing the pretrained `google/ddpm-celebahq-256` model using LAION aesthetic score as the reward function.** The top-left image shows a sample generated by the pretrained model. Moving from left to right and top to bottom, the sample is regenerated from the same initial noise after each update of the model’s parameters. The final result of the finetuning process is shown in the bottom-right image.

5.2. Beyond Face Generation

In the previous sections, we demonstrated the effectiveness of the DDPO method on a smaller pretrained model—114 *million parameters versus the 860 million parameters used in the reference work (Stable Diffusion 1.4 [1])*—to generate images of faces, optimizing for four different downstream tasks. To verify the robustness of the method, experiments were conducted using another pretrained model with a completely different visual semantics, focusing not on face generation but on images of buildings, specifically trained on churches from a subset of the **Large-scale Scene UNderstanding** challenge dataset, also known as LSUN [64]. The base model, `google/ddpm-church-256`, trained using the same DDPM methodology [15] outlined in Algorithm 1, was finetuned for the first three downstream tasks, excluding OVER50 for obvious reasons.

Figure 5.12 offers a visual comparison between four samples from the pretrained model and their versions generated by the finetuned models across the respective tasks. The images are generated from the same initial noise to facilitate comparison and highlight the effects induced by the reward functions. It is noteworthy that the definition of the churches is not always of high quality; in the first image of the pretrained panel, a building is visible that might be a church, but it is not immediately obvious. One hypothesis for why this model

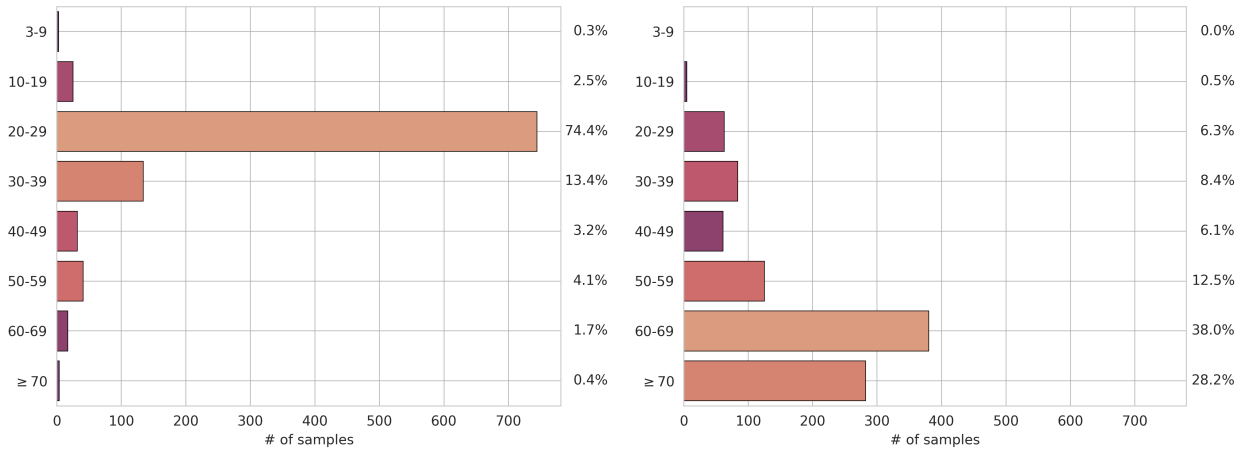


Figure 5.10: **Face age distribution using the ViT age classifier [25] prediction classes on DDPM samples (left) and DDPO finetuned model samples (right).** Finetuning with DDPO using the OVER50 reward function, which maximizes the logits sum for age classes 50 – 59, 60 – 69, and ≥ 70 , increases the proportion of faces over 50 years old from 6.1% in the baseline to 78.7% in the finetuned samples.

generates lower-fidelity images is the complexity of details and variations in scene images, such as those of churches, compared to other concepts like the elements needed to generate faces. This likely results in the model not capturing details as effectively, thereby reducing its generative capacity. The remaining three images from the pretrained model samples exhibit higher levels of definition, clearly showing churches of different styles.

The results for the compressibility and incompressibility reward tasks are shown in Table 5.1, where the average size of images generated by the pretrained model after JPEG compression is reported to be 29.57 kB. These images are generally larger than those generated by the pretrained model for celebrity faces, due to the greater number of elements and details in scene images like those of churches, making compression more challenging. In the case of the compressibility reward, the average image size generated by the model finetuned with DDPO is 10.62 kB, while for the incompressibility task, the average size is 50.21 kB. This represents a 64% reduction in image size when optimizing for compressibility and a 70% increase in image size when optimizing for incompressibility, relative to the model’s initial generative capabilities.

Analyzing the effects of the reward functions as seen in the samples in Figure 5.12, the primary emerging effect when maximizing compressibility is the prevalence of night skies and overall darker images. Notably, even as the images darken, light sources such as the moon or artificial illumination emerges, as seen in the first row of images. The effects of optimizing the incompressibility task include an overall increase in image illumination and the generation of elements like trees, grass, or bushes with numerous leaves. In this regard, the two samples in the first column of the incompressibility examples show signs of over-optimization, as the effect degrades the visual semantics of the reference churches. Transitions from pretrained to finetuned models are presented in Figure 5.13; the first row provides another example of how the darkening effect in the skies emerges, but also of how artificial illumination emanates from the building itself. Conversely, the second row shows the addition of strong green tones



Figure 5.11: **Sample transformation when optimizing the pretrained `google/ddpm-celebahq-256` model using OVER50 as the reward function.** The top-left image shows a sample generated by the pretrained model. Moving from left to right and top to bottom, the sample is regenerated from the same initial noise after each update of the model’s parameters. The final result of the finetuning process is shown in the bottom-right image.

resembling grass and the transformation of distant mountains into something more akin to trees. The cloudy sky begins to detail more but eventually results in an excess that diminishes the sense of density and realism. A similar degradation occurs in the building, which becomes less realistic in the final two images.

Regarding aesthetic quality in Figure 5.12, there is a generalized saturation of details, with trees appearing in 3 out of the 4 images. For example, in the second image of the first row, we can see several trees and bushes surrounding the church. Another detail visible in this image, as well as in the others in the group, are the lines on the building’s walls, which give the impression that the material is stone. This could be because images of buildings with such details tend to be rated higher in aesthetic quality by the LAION aesthetic predictor. The fourth image is more grandiose compared to its original version, with more detailed and numerous clouds, the church dome replaced by a tower, and the building protruding from the left side of the original image replaced by a tree with branches. Additionally, the interplay of light and shadow gives the impression that the image was taken at sunset. In terms of quan-

titative results, Table 5.1 reports an average aesthetic score of 4.77 for the pretrained LSUN model, which increases to 5.13 after optimizing for aesthetic quality. Finally, a transition image between the pretrained model and checkpoints during parameter tuning is presented in the third row of Figure 5.13. This transition suggests converting the church in the figure into an image where the background contains houses and other buildings. The hypothesis for this effect is that many images containing churches also include a view of the city, as seen in the first pretrained model sample in Figure 5.12, which may explain why the transition results in an image saturated with elements that blur the distinction of individual buildings but collectively resemble a city or cluster of buildings.

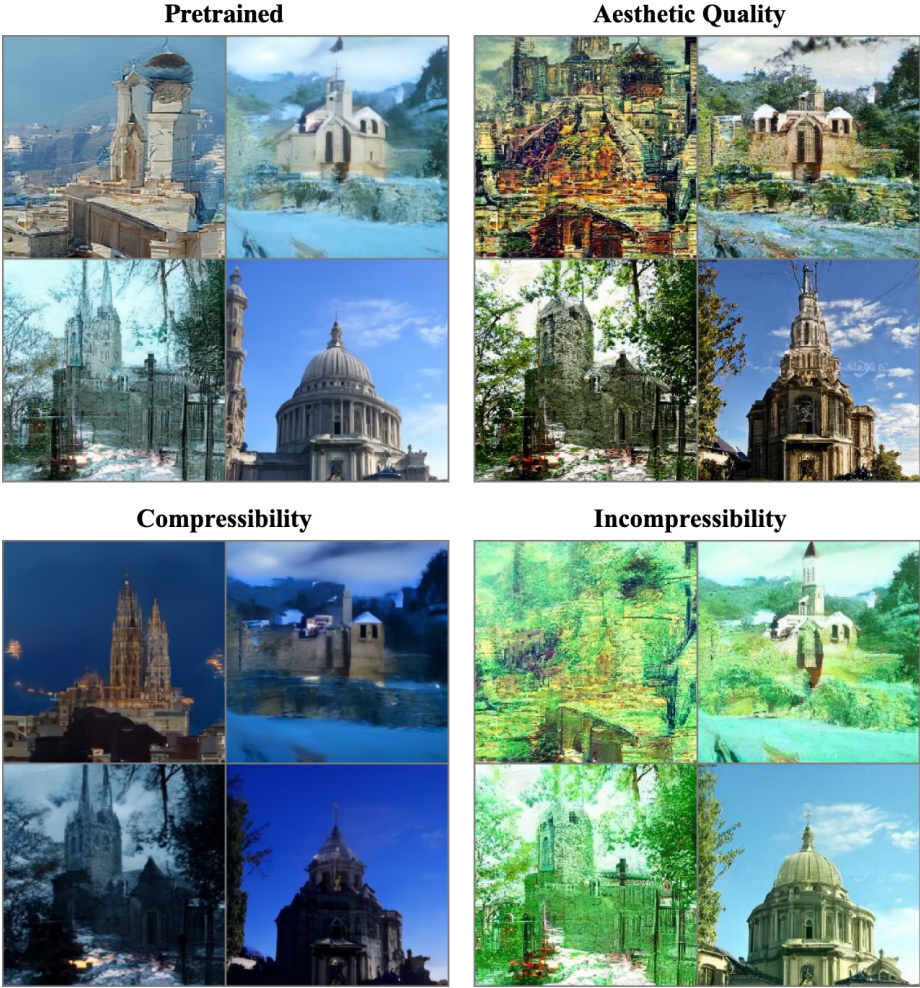


Figure 5.12: **Qualitative comparison of the effects of DDPO finetuning versus the pretrained model.** The top-left panel shows samples from the DDPM pretrained model [google/ddpm-church-256](https://huggingface.co/google/ddpm-church-256). The other panels display samples generated from the same initial noise, but using models finetuned with DDPO for different reward functions: aesthetic quality (top right), JPEG compressibility (bottom left), and incompressibility (bottom right). Additional samples are provided in Appendix D



Figure 5.13: **Sample transformation when optimizing the pretrained [google/ddpm-church-256](#) model using the reward functions: JPEG compressibility (top) and incompressibility (middle), and LAION aesthetic score (bottom).** First column shows the sample generated by the pretrained model. Moving from left to right the sample is regenerated from the same initial noise after each update of the model’s parameters. Final result of the finetuning process is shown in the last column.

5.3. Discussion & Limitations

Overoptimization and diversity samples. Despite the benefits of optimizing diffusion models using reinforcement learning, reward overoptimization remains a significant challenge. This issue arises when the model excessively exploits the reward function [65], leading to a lack of diversity in the generated samples. In severe cases, this can degrade image semantics, resulting in a model that fails to achieve practical utility.

To understand and visualize reward overoptimization in the context of this work, we extract CLIP [42] features from two sets of images that share the same initial seed and noise. One set is generated by the [google/ddpm-celebahq-256](#) (as mentioned in Section 4.2), and the other set is generated using a DDPO checkpoint [alkzar90/aesthetic-celebahq-256](#) optimized for aesthetic quality. We then project these image features into a 2 dimensional embedding space using t-SNE [66] to visualize the samples.

As shown in Figure 5.14, we observe that the samples optimized for aesthetic quality cluster near the sample with the highest aesthetic score from the DDPM set (illustrated in Figure 4.4). This occurs because the model reinforces samples with higher aesthetic quality, generating more of these samples until the model concentrates on a very specific high-reward region, ultimately collapsing into a single mode.

Broader impact. Research into finetuned diffusion models with reinforcement learning offers significant advantages in gaining precise control over image generation, unlocking various practical applications. These models can be tailored to produce highly specific outputs, allo-

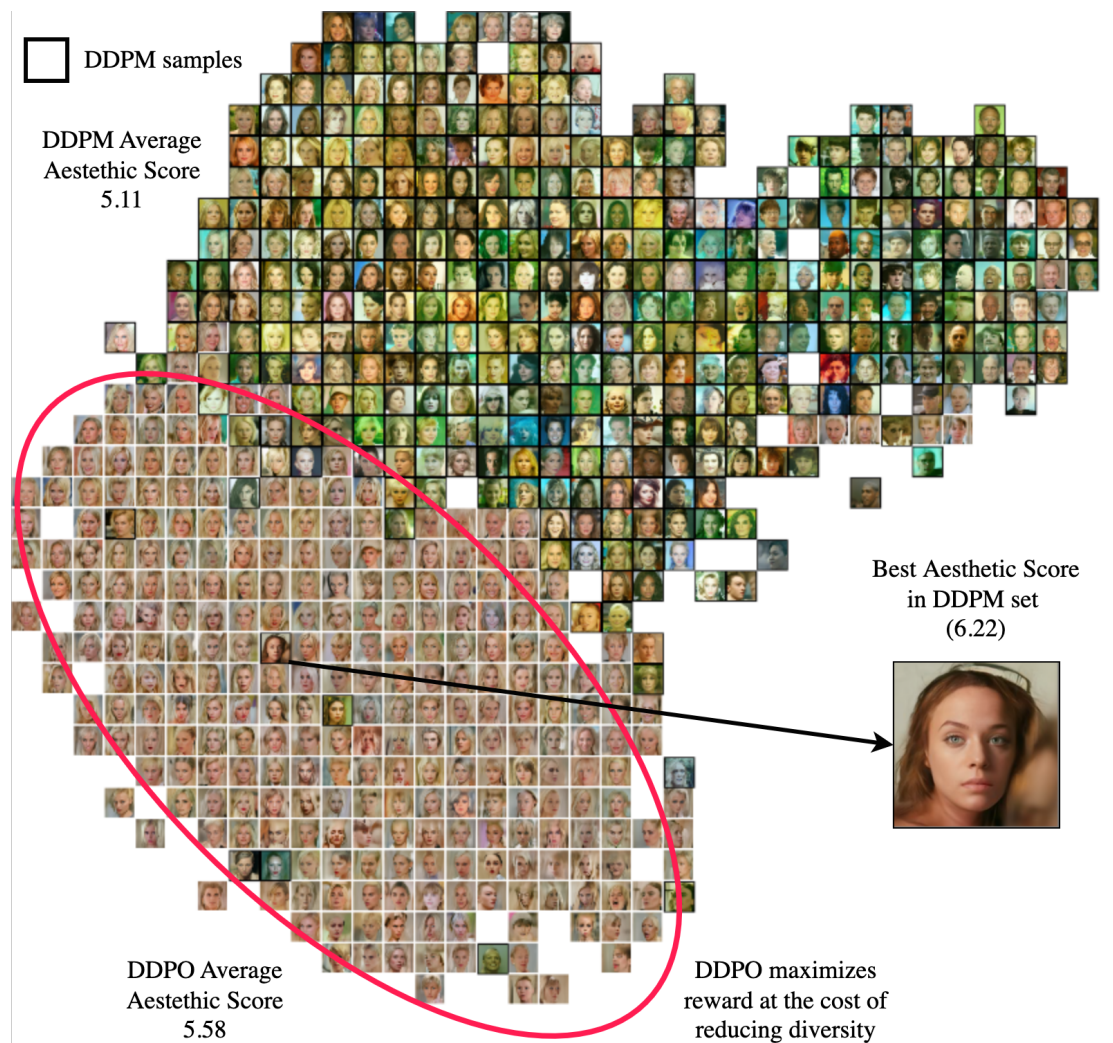


Figure 5.14: **A CLIP feature embedding space coexisting DDPM and DDPO Samples.** Both sets of samples were generated from the same initial noise. Notably, samples optimized for aesthetic quality cluster near the highest aesthetic score sample in the DDPM set (Figure 4.4), illustrating a clear *mode collapse effect*.

wing for the generation of images that meet exact requirements in fields such as entertainment, virtual reality, medical imaging, and design. For instance, in the healthcare industry, these models could aid in creating accurate simulations for surgical planning or training, providing a safe and controlled environment for practice. In the creative industries, they can enhance the ability to generate unique and high-quality visual content, facilitating the development of realistic video games or films. The capacity to finetune these models with RL also means they can adapt to new styles or features as needed, offering businesses and researchers flexible tools that evolve with their requirements.

However, alongside these advantages, there are critical ethical concerns related to the use of diffusion models, particularly regarding bias and fairness. When trained on datasets containing biases, such as those related to gender, geography, or ethnicity, these models risk perpetuating and amplifying these biases in their outputs. It doesn't require much attention to notice that the mode collapse effect, previously demonstrated, is a clear example of pre-

ference bias affecting the reward model. Samples with higher rewards are concentrated in profiles of women with white skin and blond hair (see more samples in Figure B.4). While reinforcement learning with human feedback (RLHF) techniques can be used to align the model with operator preferences and mitigate this issue, they can also exacerbate the problem or lead to unexpected consequences by aligning with subjective objectives or preferences that may not be representative.

5.4. Future Work

Avoid mode collapse. One consequence of the overoptimization problem discussed previously is the mode collapse effect, which leads to a lack of diversity in the samples. Mode collapse is a common issue in generative models. Can we develop a method to avoid this effect in the context of diffusion models? An interesting research direction is to gain control over the diffusion chain process to influence and block certain features during reward finetuning. Identifying and managing local features in images to unlock alternative high-reward areas can help prevent mode collapse or at least reduce its impact without compromising sample diversity, effectively controlling the trade-off between overoptimization and diverse samples.

Measure diversity. Exploring the use of Vendi Score [67, 68], a metric inspired by statistical ecology to evaluate the diversity of samples, could provide insights into how to algorithmically improve sample generation [69] and mitigate the mode collapse problem.

A benchmark dataset for evaluation. Evaluation is difficult and not straightforward for assessing the effectiveness of a method. Does the method generalize to other models trained on different datasets? How can we evaluate the performance of the model in a more robust way? How many downstream tasks provide robustness in the generalization of a method? These questions must be addressed with equal effort as in the application of RLHF for large language models. An immediate effort is to define a set of downstream tasks that cover a wide range of objectives and preferences, on both smaller and larger models. This includes providing access not only to the checkpoints used to report results but also to the samples and reward evaluations.

Exploit the intermediate states rewards. Empirical analysis of the reward signal during the sample generation process shows that the reward signal is more informative in denoised samples than in noisy ones (Section 4.2). Can we take advantage of intermediate states to explore the space more efficiently? Introducing an agent that can build intermediate denoised modifications worth exploring could increase the data generated in a useful way for DDPO or alternative preference algorithms.

6

Conclusion

The primary goal of this work was to build the foundational knowledge necessary to understand how to use reinforcement learning (RL) to control the generation of samples from a diffusion model by maximizing a reward function. We began by establishing the background needed to comprehend what diffusion models are and how they operate, starting with a review of one of the seminal papers in this field, DDPM [15]. Following this, we conducted an experiment using an alternative approach to RL for controlling sample generation in a pretrained model, involving guidance from an external classifier based on CLIP embeddings (Section 2.6.1).

We briefly explored how to improve sampling efficiency in diffusion models by employing an implicit model, (DDIM [16]). With DDIM, once the initial noise for the image to be generated is fixed, the generation trajectory becomes deterministic. This approach offers two key benefits: (i) it allows us to estimate the denoised version of any intermediate state in the image generation process (Figure 4.4), and (ii) it leverages sample consistency, which is nearly reversible, enabling controlled image editing. This concept was tested and demonstrated in Figure 2.8.

In the realm of reinforcement learning, we approached the problem by solving the Markov Decision Process (MDP) through direct policy optimization. We introduced the necessary mathematical tools to achieve this, specifically gradient estimation via score function. The classical formulation of this in RL is the REINFORCE algorithm, where the objective is to maximize the expected return of the trajectory-generating process (Section 3.3). By employing an arbitrary scalar reward function, we can estimate the policy gradient and use it to update the parameters, reinforcing behaviors aligned with the desired outcomes encoded in the reward function. This culminated in the use of the PPO algorithm, which is more data-efficient than REINFORCE, supports batch policy updates, and is a first-order optimization algorithm (Section 3.6). This was the crucial piece needed to control sample generation in a diffusion model.

Once the foundational background was established, we delved into the intersection of RL and diffusion models, where sample generation becomes a sequential decision-making process, and each action corresponds to selecting the next state in the diffusion chain (Section 4.1). This perspective allowed us to formulate the MDP and, given any reward function we wish to maximize, apply RL tools to find the policy that maximizes the expected reward—*where an*

agent moves from a noisy state to a cleaner one, generating samples that enhance the desired attribute specified by the reward. We found that the implementation of policy gradients in the context of diffusion models enables us to implement an algorithm to solve this MDP, known as Denoised Diffusion Policy Optimization (DDPO), introduced in the reference work *Training Diffusion Models with Reinforcement Learning* [1].

Next, we conducted an empirical study of the reward signal during the generation process, observing that the reward signal is highly noisy for intermediate states. However, when applying the reward signal to denoised versions of these states, the signal became more stable and less noisy. This observation suggests a potential improvement for the DDPO algorithm by exploring the possibility of using the reward signal on denoised intermediate states (Section 4.2).

Armed with this knowledge, we implemented DDPO and presented the results of experiments conducted using pretrained models based on the DDPM methodology. In the results chapter, we detail these experiments, which utilized pretrained diffusion models such as [google/ddpm-celebaahq-256](#) and [google/ddpm-church-256](#). DDPO was implemented and optimized based on reward functions used in the reference work, such as JPEG compressibility (5.1.1), incompressibility (5.1.2), and the LAION Aesthetic Score (5.1.3), yielding promising results (Table 5.1). The latter reward function exemplifies the potential of incorporating human feedback into the generation loop to control generative models through RL using human feedback (Section 3.7).

Additionally, we designed a novel reward function based on a human face age classifier [25] and conducted an experiment aimed at altering the proportion of famous faces generated, specifically to predominantly produce images of individuals over 50 years old. This reward function, named **OVER50** (5.1.4), was used to showcase the flexibility of RL in controlling sample generation in a diffusion model and its capacity to design reward functions that capture diverse objectives.

Despite these advancements, challenges remain in the application of these methodologies. Issues such as over-optimization and the resulting loss of diversity in the samples (Figure 5.14), as well as the need to strengthen evaluation benchmarks for more effective comparison with other methods, persist. These challenges present opportunities for further exploration in a rapidly evolving field that intersects diffusion models, reinforcement learning, and human-computer interaction.

Bibliography

- [1] Black, K., Janner, M., Du, Y., Kostrikov, I., y Levine, S., “Training diffusion models with reinforcement learning,” arXiv preprint arXiv:2305.13301, 2023.
- [2] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Aspell, A., Welinder, P., Christiano, P., Leike, J., y Lowe, R., “Training language models to follow instructions with human feedback,” 2022.
- [3] Lee, K., Liu, H., Ryu, M., Watkins, O., Du, Y., Boutilier, C., Abbeel, P., Ghavamzadeh, M., y Gu, S. S., “Aligning text-to-image models using human feedback,” 2023.
- [4] Fan, Y., Watkins, O., Du, Y., Liu, H., Ryu, M., Boutilier, C., Abbeel, P., Ghavamzadeh, M., Lee, K., y Lee, K., “Dpok: Reinforcement learning for fine-tuning text-to-image diffusion models,” 2023, <https://arxiv.org/abs/2305.16381>.
- [5] Deng, F., Wang, Q., Wei, W., Grundmann, M., y Hou, T., “Prdp: Proximal reward difference prediction for large-scale reward finetuning of diffusion models,” 2024, <https://arxiv.org/abs/2402.08714>.
- [6] Du, Y. Q., Human-Centric Reward Design. PhD thesis, UC Berkeley, 2023.
- [7] Pommeranz, A., Broekens, J., Wiggers, P., Brinkman, W.-P., y Jonker, C. M., “Designing interfaces for explicit preference elicitation: a user-centered investigation of preference representation and elicitation process,” *User Modeling and User-Adapted Interaction*, vol. 22, no. 4, pp. 357–397, 2012.
- [8] Arzate Cruz, C. y Igarashi, T., “A survey on interactive reinforcement learning: Design principles and open challenges,” en *Proceedings of the 2020 ACM Designing Interactive Systems Conference, DIS '20, (New York, NY, USA)*, p. 1195–1209, Association for Computing Machinery, 2020, [doi:10.1145/3357236.3395525](https://doi.org/10.1145/3357236.3395525).
- [9] Wan, Z., Wang, X., Liu, C., Alam, S., Zheng, Y., Liu, J., Qu, Z., Yan, S., Zhu, Y., Zhang, Q., Chowdhury, M., y Zhang, M., “Efficient large language models: A survey,” 2024, <https://arxiv.org/abs/2312.03863>.
- [10] Karras, T., Aittala, M., Aila, T., y Laine, S., “Elucidating the design space of diffusion-based generative models,” 2022, <https://arxiv.org/abs/2206.00364>.
- [11] Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., y Poole, B., “Score-based generative modeling through stochastic differential equations,” 2021, <https://arxiv.org/abs/2011.13456>.
- [12] Salimans, T. y Ho, J., “Progressive distillation for fast sampling of diffusion models,” *ArXiv*, vol. abs/2202.00512, 2022.
- [13] Watson, D., Ho, J., Norouzi, M., y Chan, W., “Learning to efficiently sample from

- diffusion probabilistic models,” 2021, <https://arxiv.org/abs/2106.03802>.
- [14] Watson, D., Chan, W., Ho, J., y Norouzi, M., “Learning fast samplers for diffusion models by differentiating through sample quality,” 2022, <https://arxiv.org/abs/2202.05830>.
- [15] Ho, J., Jain, A., y Abbeel, P., “Denoising diffusion probabilistic models,” 2020.
- [16] Song, J., Meng, C., y Ermon, S., “Denoising diffusion implicit models,” arXiv preprint arXiv:2010.02502, 2020.
- [17] Nichol, A. y Dhariwal, P., “Improved denoising diffusion probabilistic models,” 2021.
- [18] Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., y Chen, M., “Hierarchical text-conditional image generation with clip latents,” arXiv preprint arXiv:2204.06125, 2022.
- [19] Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E. L., Ghasemipour, K., Gontijo Lopes, R., Karagol Ayan, B., Salimans, T., *et al.*, “Photorealistic text-to-image diffusion models with deep language understanding,” Advances in Neural Information Processing Systems, vol. 35, pp. 36479–36494, 2022.
- [20] Ruiz, N., Li, Y., Jampani, V., Pritch, Y., Rubinstein, M., y Aberman, K., “Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation,” 2023.
- [21] Gal, R., Alaluf, Y., Atzmon, Y., Patashnik, O., Bermano, A. H., Chechik, G., y Cohen-Or, D., “An image is worth one word: Personalizing text-to-image generation using textual inversion,” 2022.
- [22] Zhang, L. y Agrawala, M., “Adding conditional control to text-to-image diffusion models,” 2023.
- [23] Kaufmann, T., Weng, P., Bengs, V., y Hüllermeier, E., “A survey of reinforcement learning from human feedback,” arXiv preprint arXiv:2312.14925, 2023.
- [24] Schuhmann, C., “Laion-aesthetics v2,” 2022, <https://laion.ai/blog/laion-aesthetics/>.
- [25] Raw, N., “Age classification from facial photos using visual transformers trained on the fairface dataset,” 2021, <https://huggingface.co/nateraw/vit-age-classifier>.
- [26] Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., y Ganguli, S., “Deep unsupervised learning using nonequilibrium thermodynamics,” 2015.
- [27] Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., y Sutskever, I., “Zero-shot text-to-image generation,” 2021.
- [28] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., y Ommer, B., “High-resolution image synthesis with latent diffusion models,” 2022.
- [29] Singer, U., Polyak, A., Hayes, T., Yin, X., An, J., Zhang, S., Hu, Q., Yang, H., Ashual, O., Gafni, O., Parikh, D., Gupta, S., y Taigman, Y., “Make-a-video: Text-to-video generation without text-video data,” 2022.
- [30] Jing, B., Corso, G., Chang, J., Barzilay, R., y Jaakkola, T., “Torsional diffusion for molecular conformer generation,” 2023.
- [31] Yang, L., Zhang, Z., Song, Y., Hong, S., Xu, R., Zhao, Y., Zhang, W., Cui, B., y Yang, M.-H., “Diffusion models: A comprehensive survey of methods and applications,” 2024.
- [32] Song, Y. y Ermon, S., “Generative modeling by estimating gradients of the data distribution,” 2020.

- [33] Song, Y. y Ermon, S., “Improved techniques for training score-based generative models,” 2020.
- [34] Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., y Poole, B., “Score-based generative modeling through stochastic differential equations,” en International Conference on Learning Representations, 2021, <https://openreview.net/forum?id=PxTIG12RRHS>.
- [35] Luo, C., “Understanding diffusion models: A unified perspective,” 2022.
- [36] Kingma, D. P. y Welling, M., “Auto-encoding variational bayes,” arXiv preprint arXiv:1312.6114, 2013.
- [37] Baydin, A. G., Pearlmutter, B. A., Radul, A. A., y Siskind, J. M., “Automatic differentiation in machine learning: a survey,” 2018, <https://arxiv.org/abs/1502.05767>.
- [38] Weng, L., “What are diffusion models?,” lilianweng.github.io, 2021, <https://lilianweng.github.io/posts/2021-07-11-diffusion-models/>.
- [39] Dhariwal, P. y Nichol, A., “Diffusion models beat gans on image synthesis,” ArXiv, vol. abs/2105.05233, 2021.
- [40] Dieleman, S., “Guidance: a cheat code for diffusion models,” 2022, <https://benanne.github.io/2022/05/26/guidance.html>.
- [41] Murphy, K., Probabilistic Machine Learning: An Introduction. Adaptive Computation and Machine Learning series, MIT Press, 2022, <https://books.google.cl/books?id=HLlyzgEACAAJ>.
- [42] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., y Sutskever, I., “Learning transferable visual models from natural language supervision,” 2021.
- [43] Bansal, A., Chu, H.-M., Schwarzschild, A., Sengupta, S., Goldblum, M., Geiping, J., y Goldstein, T., “Universal guidance for diffusion models,” en Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 843–852, 2023.
- [44] Ho, J., “Classifier-free diffusion guidance,” ArXiv, vol. abs/2207.12598, 2022.
- [45] Sutton, R. S. y Barto, A. G., Reinforcement Learning: An Introduction. The MIT Press, second ed., 2018, <http://incompleteideas.net/book/the-book-2nd.html>.
- [46] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., y Riedmiller, M., “Playing atari with deep reinforcement learning,” 2013.
- [47] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., *et al.*, “Mastering the game of go with deep neural networks and tree search,” nature, vol. 529, no. 7587, pp. 484–489, 2016.
- [48] Schulman, J., “Optimizing expectations: From deep reinforcement learning to stochastic computation graphs,” 2016.
- [49] Sutton, R. S., McAllester, D., Singh, S., y Mansour, Y., “Policy gradient methods for reinforcement learning with function approximation,” Advances in neural information processing systems, vol. 12, 1999.
- [50] Mohamed, S., Rosca, M., Figurnov, M., y Mnih, A., “Monte carlo gradient estimation

- in machine learning,” 2020.
- [51] Williams, R. J., “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine learning*, vol. 8, pp. 229–256, 1992.
- [52] Abbeel, P., “L3 policy gradients and advantage estimation.”, <https://youtu.be/AKbX1Zvo7r8?t=1501>.
- [53] Greensmith, E., Bartlett, P., y Baxter, J., “Variance reduction techniques for gradient estimates in reinforcement learning,” en *Advances in Neural Information Processing Systems* (Dietterich, T., Becker, S., y Ghahramani, Z., eds.), vol. 14, MIT Press, 2001, https://proceedings.neurips.cc/paper_files/paper/2001/file/584b98aac2dddf59ee2cf19ca4ccb75e-Paper.pdf.
- [54] Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Husenot, L., Geist, M., Pietquin, O., Michalski, M., Gelly, S., y Bachem, O., “What matters in on-policy reinforcement learning? a large-scale empirical study,” 2020, <https://arxiv.org/abs/2006.05990>.
- [55] MacKay, D. J. C., *Information Theory, Inference & Learning Algorithms*. USA: Cambridge University Press, 2002.
- [56] Schulman, J., Levine, S., Abbeel, P., Jordan, M., y Moritz, P., “Trust region policy optimization,” en *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.
- [57] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., y Klimov, O., “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [58] Kingma, D. P. y Ba, J., “Adam: A method for stochastic optimization,” 2017, <https://arxiv.org/abs/1412.6980>.
- [59] Huang, S., Dossa, R. F. J., Raffin, A., Kanervisto, A., y Wang, W., “The 37 implementation details of proximal policy optimization,” en *The ICLR Blog Track 2023*, 2022, <https://elib.dlr.de/191986/>.
- [60] Bradley, R. A. y Terry, M. E., “Rank analysis of incomplete block designs: I. the method of paired comparisons,” *Biometrika*, vol. 39, no. 3/4, pp. 324–345, 1952, <http://www.jstor.org/stable/2334029> (visitado el 2024-07-07).
- [61] Du, Y., Konyushkova, K., Denil, M., Raju, A., Landon, J., Hill, F., de Freitas, N., y Cabi, S., “Vision-language models as success detectors,” 2023, <https://arxiv.org/abs/2303.07280>.
- [62] Karras, T., Aila, T., Laine, S., y Lehtinen, J., “Progressive growing of gans for improved quality, stability, and variation,” *arXiv preprint arXiv:1710.10196*, 2017.
- [63] Kärkkäinen, K. y Joo, J., “Fairface: Face attribute dataset for balanced race, gender, and age,” 2019, <https://arxiv.org/abs/1908.04913>.
- [64] Wang, L., Guo, S., Huang, W., Xiong, Y., y Qiao, Y., “Knowledge guided disambiguation for large-scale scene classification with multi-resolution cnns,” *IEEE Transactions on Image Processing*, vol. 26, no. 4, p. 2055–2068, 2017, [doi:10.1109/tip.2017.2675339](https://doi.org/10.1109/tip.2017.2675339).
- [65] Gao, L., Schulman, J., y Hilton, J., “Scaling laws for reward model overoptimization,” en *International Conference on Machine Learning*, pp. 10835–10866, PMLR, 2023.
- [66] Van der Maaten, L. y Hinton, G., “Visualizing data using t-sne.” *Journal of machine*

learning research, vol. 9, no. 11, 2008.

- [67] Friedman, D. y Dieng, A. B., “The vendi score: A diversity evaluation metric for machine learning,” 2023, <https://arxiv.org/abs/2210.02410>.
- [68] Pasarkar, A. P. y Dieng, A. B., “Cousins of the vendi score: A family of similarity-based diversity metrics for science and machine learning,” 2024, <https://arxiv.org/abs/2310.12952>.
- [69] Hemmat, R. A., Hall, M., Sun, A., Ross, C., Drozdal, M., y Romero-Soriano, A., “Improving geo-diversity of generated images with contextualized vendi score guidance,” 2024, <https://arxiv.org/abs/2406.04551>.

Annex

A. Implementation details

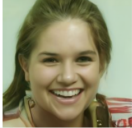

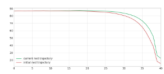
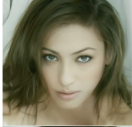
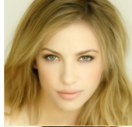
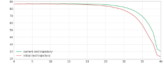
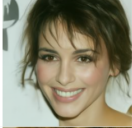
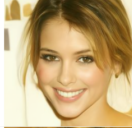
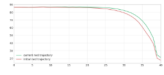
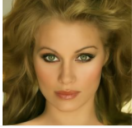
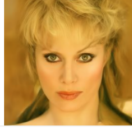
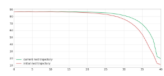
Each experiment conducted in this work is meticulously documented and accessible through their respective experiment dashboards on Weights & Biases (W&B), ensuring comprehensive logging of all relevant metrics and parameters. Additionally, the corresponding model checkpoints are stored and can be retrieved from Hugging Face model repositories, providing seamless integration for further analysis and reproducibility. The details of these experiments, including the model checkpoints and experiment dashboards, are summarized in Table A.1. The code repository used for training and evaluation is available at the following link: <https://github.com/alcazar90/ddpo-celebahq>.

Experiment	Model (Hugging Face)	W&B
google/ddpm-celebahq-256		
Aesthetic Quality	aesthetic-celebahq-256	run1/run2
Compressibility	compressibility-celebahq-256	run1/run2
Incompressibility	incompressibility-celebahq-256	run1/run2
OVER50	over50-celebahq-256	run1/run2/run3/run4/run5/run6
google/ddpm-church-256		
Aesthetic Quality	aesthetic-church-256	run1/run2
Compressibility	compressibility-church-256	run1/run2/run3
Incompressibility	incompressibility-church-256	run1/run2/run3

Table A.1: **Experiment details** with corresponding model checkpoints on Hugging Face and experiment dashboards on Weight & Biases, including logging information. Multiple runs indicate that the experiment continued training from the previous run, using the last saved checkpoint.

Tracking a Development Set. To monitor the model’s performance during training, a development set is used to evaluate performance on a held-out set of images. This development set consists of 64 images from the pretrained model, which are not used to update the parameters and are regenerated during each evaluation phase. The development set is evaluated in each iteration, providing a comparison between the original image, the current image, both rewards, and a plot showing the reward curves during sample generation. The table in Figure A.1 illustrates the tracking of the development set during training, with results logged to the experiment dashboard on W&B.

runs.summary["eval_table"] ⚙️

	original_samples	current_samples	current_final_reward	original_final_reward	diff_reward	reward_trajectory
39			22.346	15.289	7.057	
40			20.127	12.463	7.664	
41			21.036	17.407	3.629	
42			19.386	10.906	8.48	

☰ ☰ ☰ - < < 39 - 42 of 64 > >

Export as CSV Columns... Reset table

Figure A.1: **Table for Monitoring Development Set Progress.** This table compares images generated by the pretrained model with those refined using DDPO. It displays rewards for each image, their differences, and a graph illustrating reward curves throughout the diffusion model’s generation process.

Hyperparameters. In Table A.2 there is a summary of the hyperparameters used for finetuning the DDPM models on the JPEG Compressibility, Incompressibility, and Aesthetic Quality tasks using DDPO.

	Compressibility	Incompressibility	Aesthetic
Diffusion			
Denoising steps (T)	40	40	40
DDIMScheduler	True	True	True
Optimization			
Number of epochs	72	72	35
Half-cycle cosine scheduler	False	False	True
Initial learning rate	9e-8	9e-8	9e-8
Peak learning rate	9e-8	9e-8	3.75e-6
Warmup steps	-	-	25 %
Optimizer	AdamW	AdamW	AdamW
Weight decay	1e-4	1e-4	1e-4
β_1	0.9	0.9	0.9
β_2	0.999	0.999	0.999
ϵ	1e-8	1e-8	1e-8
Gradient clip norm	1.0	1.0	1.0
DDPO			
Batch size	10	10	10
Samples per iteration	100	100	100
Number of inner iterations	1	1	1
Gradient updates per iteration	10	10	10
Clip range probability ratio	1e-4	1e-4	1e-4
Clip advantages	4.5	4.5	10

Table A.2: Hyperparameters for finetuning [google/ddpm-celebahq-256](#) on JPEG Compressibility, Incompressibility, and Aesthetic Quality tasks using DDPO.

B. Additional Samples: Celebrity faces by DDPO



Figure B.1: $256x \times 256$ celebrity face samples generated by the pretrained model [google/ddpm-celebahq-256](#).



Figure B.2: 256×256 celebrity face samples generated by the DDPO finetuned model [alkzar90/ddpo-compressibility-celebahq-256](#), optimized for JPEG compressibility.

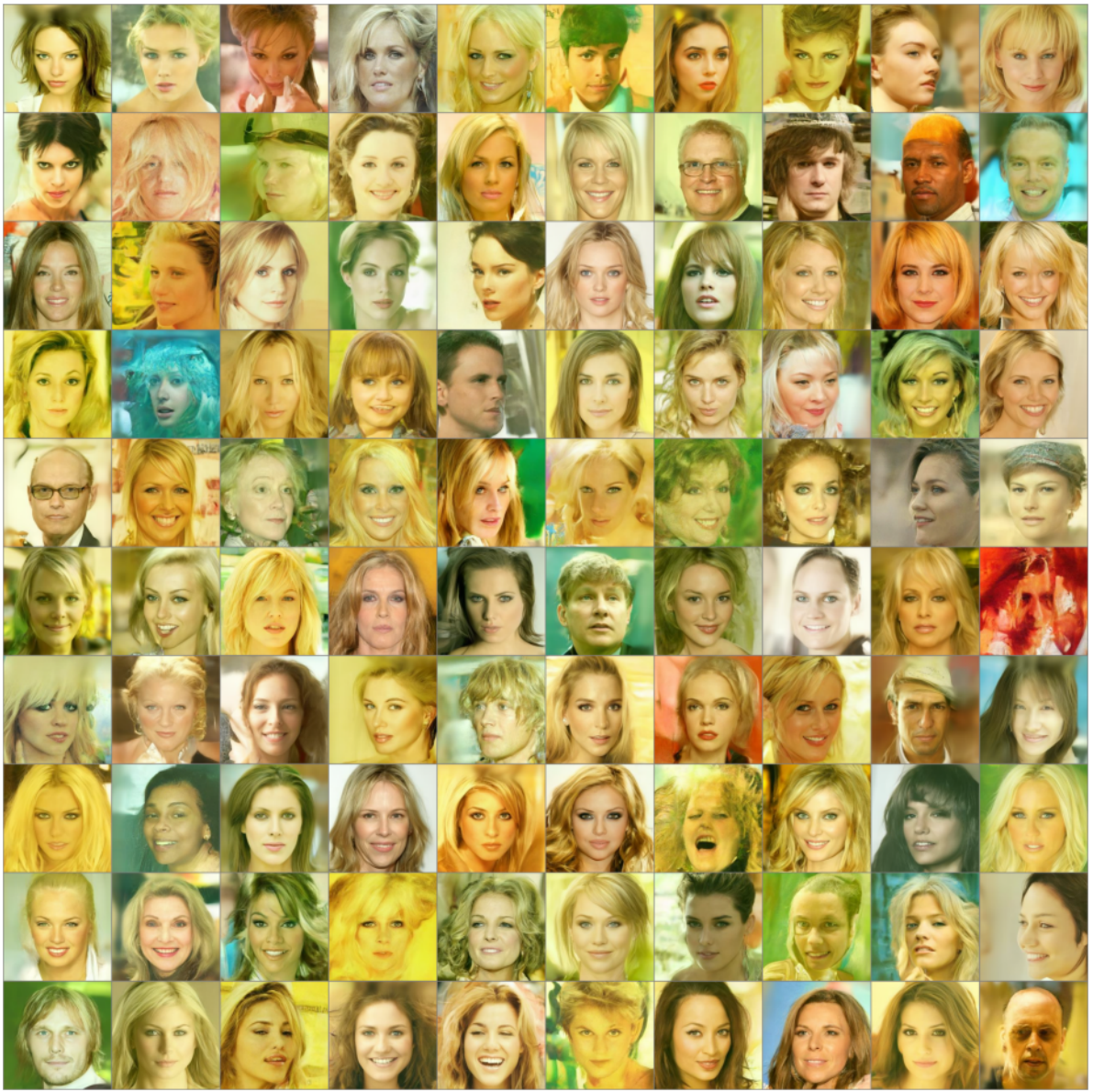


Figure B.3: 256×256 celebrity face samples generated by the DDPO finetuned model [alkzar90/ddpo-incompressibility-celebahq-256](#), optimized for JPEG incompressibility.



Figure B.4: 256×256 celebrity face samples generated by the DDPO finetuned model [alkzar90/ddpo-aesthetic-celebahq-256](#), optimized for aesthetic quality.

C. Additional Transitions: from DDPM to DDPO



Figure C.1: **Example 1 of JPEG compressibility transformation during model updates**, starting with a pretrained DDPM model and optimized with DDPO to maximize image file size reduction after JPEG compression.

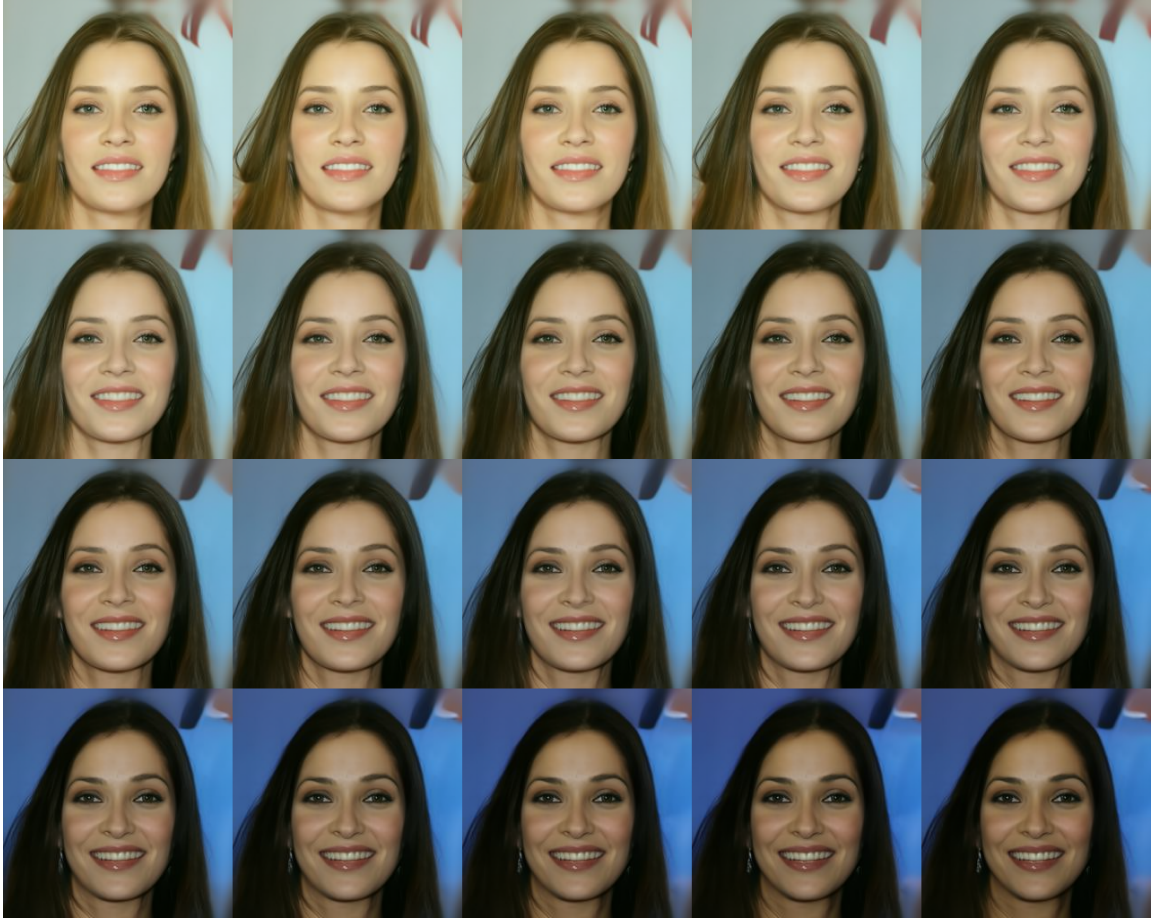


Figure C.2: **Example 2 of JPEG compressibility transformation during model updates**, starting with a pretrained DDPM model and optimized with DDPO to maximize image file size reduction after JPEG compression.

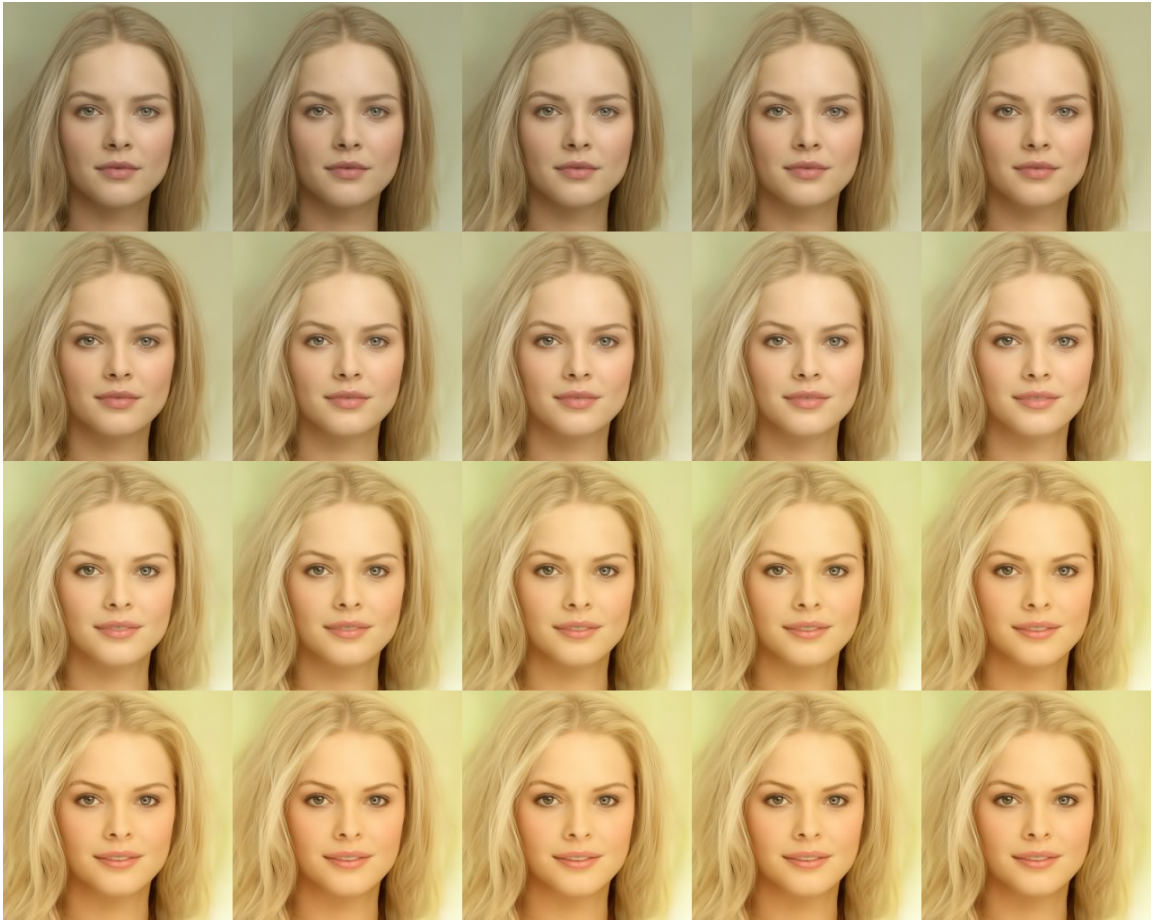


Figure C.3: **Example 1 of JPEG incompressibility transformation during model updates**, starting with a pretrained DDPM model and optimized with DDPO to maximize image file size after JPEG compression.



Figure C.4: **Example 2 of JPEG incompressibility transformation during model updates**, starting with a pretrained DDPM model and optimized with DDPO to maximize image file size after JPEG compression.



Figure C.5: **Example 1 of aesthetic quality transformation during model updates**, starting with a pretrained DDPM model and optimized with DDPO to maximize aesthetic quality.



Figure C.6: **Example 2 of aesthetic quality transformation during model updates**, starting with a pretrained DDPM model and optimized with DDPO to maximize aesthetic quality.



Figure C.7: **Example 1 of OVER50 transformation during model updates**, starting with a pretrained DDPM model and optimized with DDPO to maximize the sum of logits for classes ≥ 50 years old using the ViT Age classifier.



Figure C.8: **Example 2 of OVER50 transformation during model updates**, starting with a pretrained DDPM model and optimized with DDPO to maximize the sum of logits for classes ≥ 50 years old using the ViT Age classifier.

D. Additional Samples: Church images by DDPO



Figure D.1: 256×256 church samples generated by the pretrained model [google/ddpm-church-256](#).



Figure D.2: 256×256 church samples generated by the DDPO finetuned model [alkzar90/ddpo-compressibility-church-256](#), optimized by JPEG compressibility.



Figure D.3: 256×256 church samples generated by the DDPO finetuned model [alkzar90/ddpo-incompressibility-church-256](#), optimized for JPEG incompressibility.



Figure D.4: 256×256 church samples generated by the DDPO finetuned model [alkzar90/ddpo-aesthetic-church-256](#), optimized for aesthetic quality.