

Personalization of Text-to-Image Diffusion Models

A Comparative Study of DreamBooth, LoRA, Textual Inversion,
Custom Diffusion, Latent Consistency Distillation, and DDPO

Meghana Nanuvala

Indiana University

April 2026

Using HuggingFace Diffusers | Stable Diffusion v1.4/v1.5 | NVIDIA H100 80 GB

IU Quartz HPC Cluster

Contents

1	Abstract	3
2	Introduction	3
3	Background: How Diffusion Models Work	4
3.1	Forward Process — Gradually Adding Noise	4
3.2	Reverse Process — Predicting and Removing Noise	4
3.3	Latent Diffusion — Operating in Compressed Space	4
3.4	Text Conditioning via Cross-Attention	4
3.5	Overview Diagram	5
4	Evaluation Metrics	5
4.1	CLIP — Contrastive Language–Image Pre-training	5
4.2	DINOv2 — Self-Supervised Vision Transformer	6
4.3	Why Three Metrics?	6
5	Methods	7
5.1	DreamBooth — Full Model Fine-Tuning	7
5.2	LoRA — Low-Rank Adaptation	8
5.3	Textual Inversion — Embedding Optimization	8
5.4	Latent Consistency Distillation (LCM)	9
5.5	Custom Diffusion — Selective Cross-Attention Fine-Tuning	10
5.6	DDPO — Reinforcement Learning for Diffusion	11
5.7	Method Summary	12
6	Experimental Setup	12
7	Results	14
7.1	DreamBooth Results	14
7.2	LoRA Results	15
7.3	Textual Inversion Results	16
7.4	Custom Diffusion Results	17
7.5	LCM Distillation Results	18
7.6	DDPO Results	19
7.7	Latency Comparison	20
7.8	Metric Trends	21
8	Analysis and Discussion	22
8.1	Cross-Method Comparison	22
8.2	Key Findings by Method	22
8.3	Fidelity–Efficiency Trade-off	23
8.4	When to Use Each Method	23

9 Conclusion	23
Disclaimer	24

1 Abstract

Text-to-image diffusion models such as Stable Diffusion have demonstrated remarkable capabilities in generating high-quality images from textual descriptions. However, personalizing these models to generate images of specific subjects, styles, or concepts remains an active area of research. This report, conducted as part of an independent study under the guidance of Professor Hasan, presents a comprehensive comparative study of six personalization methods: **DreamBooth** (full model fine-tuning), **LoRA** (Low-Rank Adaptation), **Textual Inversion** (embedding optimization), **Custom Diffusion** (selective cross-attention fine-tuning), **Latent Consistency Distillation** (LCM), and **Reinforcement Learning training with DDPO** (Denoising Diffusion Policy Optimization).

I conduct systematic experiments using the HuggingFace Diffusers library on Stable Diffusion v1.5, evaluating each method under varying hyperparameter settings to understand their capabilities, trade-offs, and optimal configurations. For DreamBooth, I vary the prior-preservation loss weight $\lambda \in \{0.25, 0.50, 0.75, 1.0\}$. For LoRA, I examine the effect of rank $r \in \{4, 8, 16\}$ on style transfer quality. For Textual Inversion, I study the impact of embedding dimensionality via the number of vectors $\in \{1, 2, 4, 8\}$. For Custom Diffusion, I compare full cross-attention vs. K/V-only fine-tuning. For LCM, I compare L2 vs. Huber distillation loss. For DDPO, I compare full U-Net RL fine-tuning vs. LoRA under an aesthetic reward function. All experiments are evaluated using CLIP-T (text-image alignment), CLIP-I (image-image similarity), DINO-I (structural similarity via DINOv2), and where applicable, aesthetic scores and inference latency.

The findings reveal a clear trade-off spectrum between *fidelity* and *efficiency*: DreamBooth achieves the highest subject fidelity at the cost of ~ 3.4 GB storage, LoRA provides an excellent balance with ~ 3 MB adapters, and Textual Inversion offers the most lightweight personalization at ~ 3 –24 KB. Custom Diffusion’s selective cross-attention fine-tuning (~ 75 MB) enables multi-concept composition, with the K/V-only variant outperforming full cross-attention training. LCM distillation achieves a $\sim 27\times$ inference speedup (108 ms vs. 2,913 ms) by reducing denoising steps from 30 to 4. DDPO demonstrates that RL-based optimization with an aesthetic reward function can improve generation quality beyond reconstruction-based training, with the LoRA variant (aesthetic score: 6.28) outperforming full U-Net fine-tuning (6.05).

2 Introduction

Diffusion models [1, 2] have emerged as the dominant paradigm for high-fidelity image synthesis. Models such as Stable Diffusion [2] operate in a compressed latent space and leverage CLIP-based text conditioning to generate images from natural language prompts. While these models exhibit broad generative capabilities, they lack the ability to faithfully reproduce specific subjects, artistic styles, or user-defined concepts without additional adaptation.

Personalization refers to the task of adapting a pre-trained diffusion model so that it can generate images of a particular subject or in a particular style, given only a small number of reference images (~ 3 –5). This problem is challenging because the model must learn the target concept without overfitting to the few training samples, while preserving its ability to generalize to novel prompts and compositions. Several approaches have been proposed, each occupying a different point on the trade-off between fidelity, efficiency, and flexibility. In this report, I follow the HuggingFace Diffusers training pipelines [8] to systematically evaluate six such methods under controlled experimental conditions on Stable Diffusion v1.5 using an NVIDIA H100 80 GB GPU on the IU Quartz HPC cluster.

3 Background: How Diffusion Models Work

Since every personalization method in this report builds on top of a pre-trained diffusion model, I briefly review the core mechanism before discussing individual methods.

3.1 Forward Process — Gradually Adding Noise

A diffusion model defines a *forward* (noising) process that progressively corrupts a clean data sample x_0 over T timesteps by adding Gaussian noise according to a fixed schedule $\{\alpha_t, \sigma_t\}_{t=1}^T$:

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

At each step, a small amount of Gaussian noise is added. By the final timestep T , the data is nearly indistinguishable from pure random noise: $x_T \approx \mathcal{N}(0, \mathbf{I})$.

3.2 Reverse Process — Predicting and Removing Noise

The generative power of a diffusion model lies in learning to *reverse* this corruption. A neural network ϵ_θ (in Stable Diffusion, a U-Net) is trained to predict the noise ϵ that was added at each timestep. The training objective is a simple mean-squared-error *reconstruction loss*:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{x_0, \epsilon \sim \mathcal{N}(0, \mathbf{I}), t} [\|\epsilon - \epsilon_\theta(x_t, t)\|^2] \quad (2)$$

At inference, the process starts from pure noise x_T and iteratively denoises: at each step t , the U-Net predicts the noise component $\hat{\epsilon} = \epsilon_\theta(x_t, t)$, which is subtracted (scaled by the schedule) to yield a cleaner x_{t-1} . After all T steps, the result is a clean sample x_0 .

3.3 Latent Diffusion — Operating in Compressed Space

Running the diffusion process directly on high-resolution pixels is computationally expensive. Stable Diffusion [2] addresses this with a *Latent Diffusion Model* (LDM):

1. A pre-trained VAE (Variational Autoencoder) *encoder* compresses an image $x \in \mathbb{R}^{H \times W \times 3}$ into a lower-dimensional latent $z = \mathcal{E}(x) \in \mathbb{R}^{h \times w \times c}$ (typically $64 \times 64 \times 4$).
2. The forward and reverse diffusion processes operate entirely in this latent space on z .
3. After denoising, the VAE *decoder* maps the clean latent back to pixel space: $\hat{x} = \mathcal{D}(z_0)$.

This reduces computation by roughly $48\times$ compared to pixel-space diffusion while preserving perceptual quality.

3.4 Text Conditioning via Cross-Attention

To make generation controllable by text, a frozen CLIP text encoder converts the user’s prompt into a sequence of token embeddings c . These embeddings are injected into the U-Net through *cross-attention* layers, where the latent features form the Query and the text embeddings form the Key and Value:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V, \quad Q = W_Q z, \quad K = W_K c, \quad V = W_V c \quad (3)$$

Intuitively, although the user drives generation with a text prompt, the attention roles are assigned from the *model’s* perspective: each spatial patch in the latent acts as a **Query** (“I am a region in the

image; what should I look like?”) and searches over the text tokens, which serve as **Keys** (“which words are relevant to this location?”) and **Values** (“what visual content should flow back?”). In other words, the image features *query* the text for guidance, not the other way around. This is precisely why Custom Diffusion targets W_K and W_V : modifying the Keys changes *where* the model looks in the text for each spatial position, and modifying the Values changes *what information* is returned, together rewiring how a new concept maps from text to visual features.

All personalization methods in this report modify one or more components of this pipeline: the text embeddings (c), the cross-attention weights (W_K, W_V), or the entire U-Net (ϵ_θ).

3.5 Overview Diagram

Figure 1 illustrates the complete pipeline: forward noising, reverse denoising with the U-Net noise predictor, latent-space compression via the VAE, and text conditioning through cross-attention.

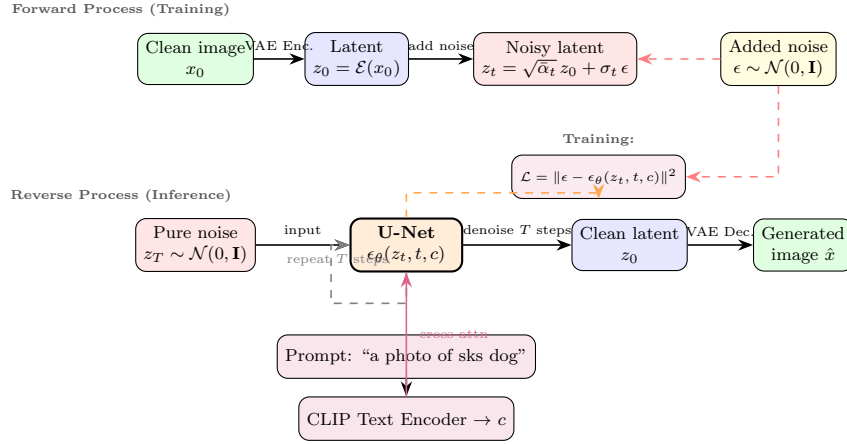


Figure 1: Overview of latent diffusion. **Top:** the forward process adds noise to a VAE-encoded latent. **Bottom:** the reverse process starts from pure noise and iteratively denoises using a U-Net conditioned on text via cross-attention. During training, the U-Net learns to predict the added noise ϵ ; during inference, this prediction is subtracted at each step to recover a clean image.

4 Evaluation Metrics

Evaluating personalized generation requires measuring both how well the output follows the text prompt and how faithfully it preserves the target subject’s appearance. I rely on two foundation models, CLIP and DINOv2, to define three complementary metrics.

4.1 CLIP — Contrastive Language–Image Pre-training

CLIP (Contrastive Language–Image Pre-training) [10] is a vision-language model developed by OpenAI that learns a shared embedding space for images and text. It was trained on ~ 400 million (image, caption) pairs collected from the internet using a *contrastive objective*: matching image–text pairs are pulled together in embedding space while non-matching pairs are pushed apart. The result is a model whose image encoder and text encoder produce directly comparable feature vectors, enabling zero-shot transfer to many downstream tasks.

CLIP is central to Stable Diffusion itself: the frozen CLIP text encoder provides the conditioning signal injected via cross-attention, so using CLIP features for evaluation is a natural choice. I derive two metrics from it:

- **CLIP-T (Text–Image Alignment):** Cosine similarity between the CLIP *text* embedding of the prompt and the CLIP *image* embedding of the generated image. It measures how well the generated image matches the textual description (“Does the image depict what was asked?”). Higher (↑) indicates better prompt adherence. Typical range for personalized outputs: 0.20–0.35.
- **CLIP-I (Image–Image Similarity):** Cosine similarity between the CLIP *image* embeddings of the generated image and the reference (input) images. It captures high-level visual similarity (texture, colour palette, overall appearance) between the generated output and the original subject. Higher (↑) indicates the generated image is visually closer to the reference. Typical range: 0.70–0.90.

Because CLIP was optimized for broad semantic alignment rather than fine-grained visual detail, it can assign high similarity scores even when structural aspects (pose, shape, proportions) differ. This motivates a complementary structural metric.

4.2 DINOv2 — Self-Supervised Vision Transformer

DINOv2 (Self-Distillation with **NO** labels, version 2) [11] is a family of Vision Transformers trained by Meta AI using self-supervised learning on a curated dataset of ~ 142 million images. Unlike CLIP, DINOv2 is trained *without any text supervision*: it learns visual features purely from images by combining a self-distillation objective (the student network must match the output of an exponential-moving-average teacher) with masked image modelling.

This purely visual training produces features that are highly sensitive to *structural* properties such as object shape, part layout, pose, and fine-grained identity, which CLIP’s language-driven training tends to abstract away. I use the ViT-S/14 variant (small Vision Transformer with 14×14 patch size) and define:

- **DINO-I (Structural Similarity):** Cosine similarity between DINOv2 CLS-token features of the generated and reference images. It answers the question “Does the generated subject have the same *structure* and *identity* as the reference?”, capturing differences in shape, pose, and spatial layout that CLIP may miss. Higher (↑) indicates a better structural match.

4.3 Why Three Metrics?

No single metric captures all dimensions of generation quality. **CLIP-T** measures *text faithfulness* (did the model follow the prompt?), **CLIP-I** measures *visual similarity* (does the output look like the reference?), and **DINO-I** measures *structural identity* (does it preserve the subject’s shape and form?). A method that scores high on CLIP-T but low on DINO-I is generating prompt-relevant images that do not faithfully preserve the subject’s identity, a common issue in under-trained embeddings. Conversely, high DINO-I with low CLIP-T signals overfitting to the reference at the expense of prompt flexibility. Together, the three metrics provide a balanced view of the personalization–generalization trade-off.

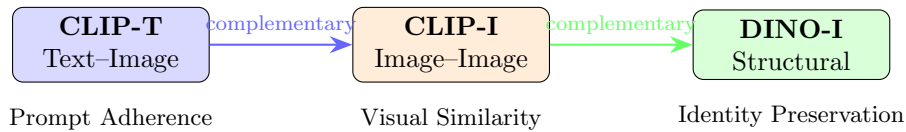


Figure 2: Three complementary evaluation metrics capturing text alignment, visual similarity, and structural identity.

5 Methods

This section describes all six personalization approaches, covering each method’s core idea, training objective, and architecture.

5.1 DreamBooth — Full Model Fine-Tuning

DreamBooth [3] personalizes a diffusion model by fine-tuning *all* parameters of the U-Net denoiser (~ 860 M) to bind a specific subject to a rare text token (e.g., “**sks**”). Given 3–5 reference images, the model learns to generate the target subject in novel contexts. The key innovation is the **prior-preservation loss**, which regularizes training by jointly optimizing on subject-specific instance images and class-level images generated by the original model. This prevents *language drift*—the phenomenon where fine-tuning causes the model to forget the general meaning of the class noun.

Architecture. The training pipeline follows a dual-path structure through the standard Stable Diffusion architecture. In the *instance path*, the subject images are paired with the prompt “a photo of **sks** dog” and processed through the frozen CLIP text encoder to obtain conditioning embeddings c_{inst} . These embeddings, together with the noised latent representations, are fed into the U-Net, which predicts the added noise ϵ_θ . The U-Net itself comprises a downsampling encoder, a middle block with self-attention and cross-attention layers, and an upsampling decoder with skip connections—all ~ 860 M parameters are updated during training.

In the *prior-preservation path*, the original (pre-training) model first generates ~ 100 generic class images (e.g., “a photo of a dog”) before training begins. During each training step, one of these class images is processed through the same pipeline to compute the prior-preservation loss. The two losses are combined as:

$$\mathcal{L} = \underbrace{\mathbb{E} [\|\epsilon - \epsilon_\theta(\alpha_t x + \sigma_t \epsilon, c_{\text{inst}})\|^2]}_{\text{Reconstruction loss (subject fidelity)}} + \lambda \cdot \underbrace{\mathbb{E} [\|\epsilon - \epsilon_\theta(\alpha_t x_{\text{pr}} + \sigma_t \epsilon, c_{\text{pr}})\|^2]}_{\text{Prior-preservation loss (class preservation)}} \quad (4)$$

The first term is a *reconstruction loss*: it trains the U-Net to denoise the subject images so that the reconstructed output faithfully matches the original reference, thereby binding the rare token to the subject’s visual identity. Here c_{inst} is the instance prompt embedding, c_{pr} is the class prompt, x_{pr} are generated class images, and λ controls prior-preservation strength. The noise scheduler samples a random timestep t and adds Gaussian noise scaled by α_t and σ_t to the clean latent, and the U-Net learns to predict this noise conditioned on the text embedding injected via cross-attention layers.

Because DreamBooth updates the entire U-Net (all encoder, decoder, self-attention, and cross-attention weights), it achieves the highest subject fidelity among parameter-efficient methods, but produces the largest checkpoint (~ 3.4 GB). The choice of the rare token “**sks**” is deliberate: it avoids collision with existing vocabulary, so the model can associate it exclusively with the target subject without interfering with pre-existing text–image mappings. Figure 3 illustrates this dual-path training pipeline.

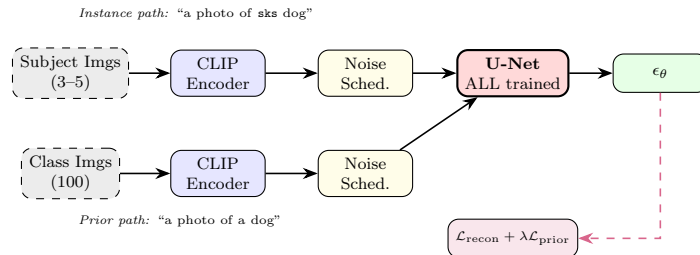


Figure 3: DreamBooth architecture. The instance path learns the subject; the prior-preservation path prevents forgetting the class. Red = trained, Blue = frozen.

5.2 LoRA — Low-Rank Adaptation

LoRA [4] freezes the pre-trained weights $W \in \mathbb{R}^{d \times k}$ and injects trainable low-rank decomposition matrices $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$ into each attention layer, where $r \ll \min(d, k)$. This achieves parameter-efficient fine-tuning with $\sim 0.1\text{--}0.7\%$ of the original parameters. At inference, adapters merge into the base weights ($W' = W + BA$) with zero latency overhead.

Architecture. The core insight of LoRA is that the weight updates ΔW needed for fine-tuning occupy a much lower-dimensional subspace than the full weight matrices. Instead of computing a full $d \times k$ update, LoRA decomposes it as $\Delta W = BA$, where $A \in \mathbb{R}^{r \times k}$ first projects the input down to rank r and $B \in \mathbb{R}^{d \times r}$ projects it back up. For each attention layer in the U-Net, LoRA inserts these adapter pairs in parallel with the frozen pretrained weight matrices for the query (W_Q), key (W_K), value (W_V), and output (W_O) projections. During the forward pass, input x flows through both the frozen path and the adapter path simultaneously:

$$h = Wx + BAx, \quad \text{where } \Delta W = BA, \quad r \cdot (d + k) \text{ params per layer vs } d \cdot k \text{ for full} \quad (5)$$

For Stable Diffusion v1.5 with $d = k = 768$ in the attention layers, a rank $r = 4$ adapter requires only $r(d + k) = 4 \times 1536 = 6,144$ parameters per layer, a $96\times$ reduction compared to $768^2 = 589,824$ for a full weight update. Matrix A is initialized with a random Gaussian distribution and B is initialized to zero, so the adapter output $BAx = 0$ at the start of training, ensuring the model begins from the original pretrained behavior.

At inference time, the adapter matrices can be algebraically merged into the base weights: $W' = W + BA$. This eliminates the parallel path entirely, so the resulting model has *identical architecture and latency* to the original—only the weight values differ. This mergeability also enables **composability**: multiple LoRA adapters can be combined additively ($W + B_1A_1 + B_2A_2$), allowing users to mix styles (e.g., “Naruto style” + “watercolor”) without retraining. The final checkpoint stores only the A and B matrices ($\sim 3\text{MB}$), making LoRA highly practical for sharing and deploying multiple personalized models on top of a single base model. Figure 4 illustrates the parallel adapter path within a single attention layer.

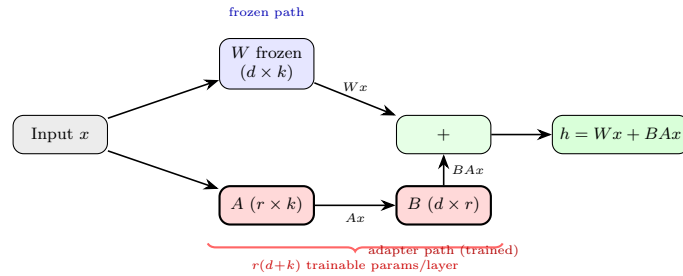


Figure 4: LoRA architecture within one attention layer. The frozen weight W and the low-rank adapter BA operate in parallel; outputs are summed. At inference, $W' = W + BA$ merges the paths.

5.3 Textual Inversion — Embedding Optimization

Textual Inversion [5] takes a minimalist approach: it learns a new text embedding vector v^* for a pseudo-token (e.g., `<sks-cat>`) while keeping the entire model frozen. This is the most parameter-efficient method, modifying only 768 parameters per embedding vector. With N vectors, the token expands to $[v_1, \dots, v_N]$ for richer representation.

Architecture. Textual Inversion operates entirely within the text conditioning pathway of the Stable Diffusion architecture. A new pseudo-token `<sks-cat>` is added to the CLIP tokenizer vocabulary, and its embedding vector $v^* \in \mathbb{R}^{768}$ is initialized either randomly or from a semantically related word (e.g., “cat”). During the forward pass, the prompt “a photo of `<sks-cat>`” is tokenized,

and the new embedding is inserted into the token sequence. This sequence passes through the *frozen* CLIP text encoder (~ 123 M parameters), which produces contextual conditioning embeddings via its 12-layer transformer. These embeddings are then injected into the *frozen* U-Net (~ 860 M parameters) through the cross-attention layers at each resolution level.

During backpropagation, gradients flow from the denoising loss backwards through the frozen U-Net, through the frozen text encoder, and arrive at the embedding lookup table. Critically, only the entry for the new pseudo-token receives updates; all other token embeddings, all text encoder weights, and all U-Net weights remain fixed. The optimization objective is:

$$v^* = \arg \min_v \mathbb{E} [\|\epsilon - \epsilon_\theta(\alpha_t x + \sigma_t \epsilon, c_\theta(\text{prompt with } v))\|^2] \quad (6)$$

where all U-Net weights θ and text encoder weights remain frozen. With the multi-vector extension, the pseudo-token expands to N consecutive embeddings $[v_1, \dots, v_N]$ in the token sequence, effectively giving the model N “words” to describe the concept. For $N = 4$, this yields $4 \times 768 = 3,072$ trainable parameters (~ 12 KB), approximately $140,000\times$ fewer than DreamBooth.

The reason this works despite such extreme parameter efficiency is that the frozen model already possesses rich knowledge of visual concepts, textures, lighting, and composition from pretraining on billions of images. Textual Inversion does not teach the model to generate—it teaches the model a new *description* of a specific concept within its existing embedding space. The frozen model acts as a powerful implicit regularizer: since no model weights can change, the learned embedding cannot overfit to training images in the same way that weight-based methods can. This explains why Textual Inversion with 4 vectors achieved the highest CLIP-I (0.857) in our experiments. Figure 5 illustrates the gradient flow.

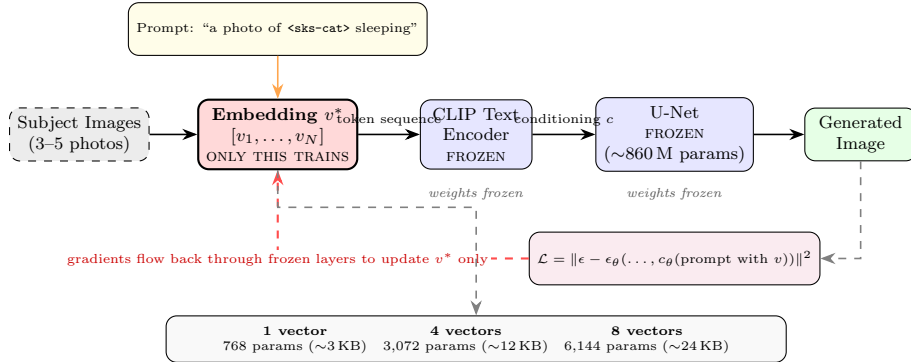


Figure 5: Textual Inversion architecture. The entire pipeline (CLIP text encoder and U-Net) is **frozen**. Only the new embedding vector(s) $v^* = [v_1, \dots, v_N]$ are trained. Gradients propagate from the denoising loss backward through all frozen layers to update only v^* , making this the most parameter-efficient method (~ 3 –24 KB).

5.4 Latent Consistency Distillation (LCM)

Latent Consistency Models [6] address a fundamentally different challenge: **inference speed**. Standard diffusion models require 20–50 sequential denoising steps to generate a single image, which limits real-time applications. LCM distills a pre-trained diffusion model (“teacher”) into a “student” that generates images in 1–4 steps, achieving a 5–50 $\times$ speedup.

Architecture. The key insight is reformulating the multi-step denoising process as solving an augmented probability flow ODE (ordinary differential equation) in the latent space. In the standard diffusion framework, each denoising step can be viewed as a numerical integration step along a trajectory from the noisy distribution to the clean data distribution. The teacher model approximates this trajectory by taking many small steps; LCM trains the student to predict the

endpoint of the trajectory *directly*, bypassing intermediate steps.

The training setup involves three models working together. The *teacher* is the frozen pre-trained Stable Diffusion model, which performs a single ODE solver step to produce an estimate $\hat{z}_{t_n}^\phi$ of what the latent should look like one step closer to clean. The *student* f_θ takes the same noisy input $z_{t_{n+1}}$ but attempts to predict the final clean output directly. The *EMA target* $f_{\theta-}$ is an exponential moving average of the student’s weights, which provides stable training targets. The consistency loss enforces that the student’s prediction at timestep t_{n+1} must match the EMA target’s prediction at timestep t_n :

$$\mathcal{L}_{\text{LCD}} = \mathbb{E} \left[d \left(f_\theta(z_{t_{n+1}}, t_{n+1}, c), f_{\theta-}(\hat{z}_{t_n}^\phi, t_n, c) \right) \right] \quad (7)$$

where $d(\cdot, \cdot)$ is a distance function—either the **Huber loss** or **LPIPS** (Learned Perceptual Image Patch Similarity) perceptual distance. The Huber loss is a hybrid of L1 and L2: it behaves like the squared error (L2) for small residuals $|r| \leq \delta$ and switches to the absolute error (L1) for large residuals, making it less sensitive to outliers than pure L2 while remaining smooth and differentiable near zero (controlled by the threshold δ , set to $c = 0.001$ in our experiments). LPIPS, by contrast, computes distance in the feature space of a pre-trained deep network (e.g., VGG or AlexNet), measuring perceptual similarity closer to human judgement than pixel-level metrics. This self-consistency constraint ensures that regardless of where along the ODE trajectory the student begins, it always maps to the same clean output—hence the name “consistency model.” The student model shares the same U-Net architecture as the teacher, but after distillation it can produce high-quality images with just 1–4 forward passes through the network, using a specially designed few-step noise schedule. The HuggingFace Diffusers library [9] supports LCM distillation from any Stable Diffusion checkpoint with LoRA-based distillation for memory efficiency. Figure 6 shows the teacher–student–EMA training setup.

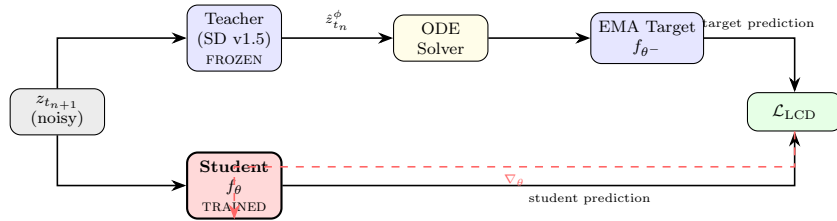


Figure 6: LCM consistency distillation. The teacher provides one-step ODE estimates; the student learns to predict the same clean output as the EMA target, enabling 1–4 step inference.

5.5 Custom Diffusion — Selective Cross-Attention Fine-Tuning

Custom Diffusion [13] occupies a middle ground between DreamBooth’s full fine-tuning and Textual Inversion’s embedding-only approach. It fine-tunes the **key (K) and value (V) projection matrices** in the U-Net’s cross-attention layers—the specific layers where text conditioning meets visual features—while keeping all other parameters frozen. Additionally, it learns a **modifier token** embedding (e.g., $\langle \mathbf{V1} \rangle$) initialized from a semantically similar word. This targets ~ 57 M parameters ($\sim 2\%$ of the model), producing a ~ 75 MB checkpoint per concept.

Architecture. The U-Net in Stable Diffusion contains two types of attention layers: *self-attention*, where spatial features attend to each other for structural coherence, and *cross-attention*, where visual features attend to text conditioning embeddings. In cross-attention, the query $Q = W_Q \cdot z$ is computed from the visual latent features z , while the key $K = W_K \cdot c$ and value $V = W_V \cdot c$ are computed from the text conditioning embedding c . The attention output is $\text{Attn}(Q, K, V) = \text{softmax}(QK^\top / \sqrt{d_k}) \cdot V$. Custom Diffusion’s insight is that **K determines where the model looks in the text** for each spatial position and **V determines what information flows back**

to the visual representation. By modifying only W_K and W_V , the method changes how the model interprets text as visual features without affecting spatial reasoning (self-attention) or the denoising backbone.

A distinctive feature is **multi-concept composition**: Custom Diffusion can jointly learn multiple concepts (e.g., a specific dog and a specific couch) by training with separate modifier tokens ($\langle V1 \rangle$, $\langle V2 \rangle$) and a shared set of K/V updates. At inference, these compose naturally: “ $\langle V1 \rangle$ dog sitting on $\langle V2 \rangle$ couch.” For regularization, it uses **real images retrieved via CLIP** (~ 200 images from LAION) rather than model-generated class images, providing more diverse regularization targets.

Training Objective. Denoising loss on concept images plus regularization with real retrieved images:

$$\mathcal{L} = \mathbb{E} [\|\epsilon - \epsilon_\theta(\alpha_t x + \sigma_t \epsilon, c_{\text{mod}})\|^2] + \lambda \cdot \mathbb{E} [\|\epsilon - \epsilon_\theta(\alpha_t x_{\text{reg}} + \sigma_t \epsilon, c_{\text{class}})\|^2] \quad (8)$$

where only the K, V projection weights and modifier token embedding receive gradients. Optionally, the Q and output projections can also be trained (`--freeze_model=crossattn`) for additional expressiveness at the cost of more trainable parameters. Figure 7 illustrates how only the cross-attention K and V projections are updated.

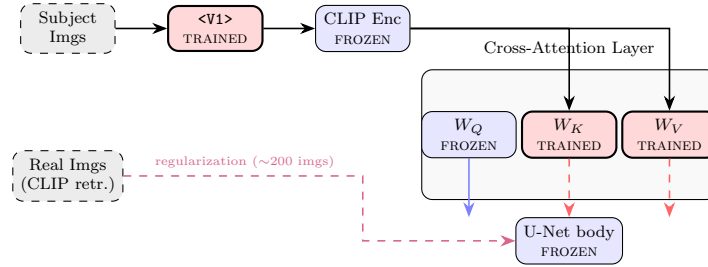


Figure 7: Custom Diffusion architecture. Only the cross-attention K and V projections plus the modifier token embedding are trained; the rest of the U-Net, CLIP encoder, and Q projections remain frozen. Real images from CLIP retrieval serve as regularization.

5.6 DDPO — Reinforcement Learning for Diffusion

DDPO [7] is fundamentally different from the other methods: instead of minimizing a reconstruction loss against training data, it optimizes an *arbitrary reward function* applied to the generated images. This enables targeting objectives that cannot be expressed as simple pixel-level or embedding-level losses, such as aesthetic quality, human preference alignment, or task-specific scores.

Architecture. DDPO reformulates the T -step denoising process as a *Markov Decision Process* (MDP). At each timestep, the **state** s_t is the current noisy latent x_t , the **action** a_t is the noise prediction $\epsilon_\theta(x_t, t, c)$, and the **transition** applies the noise scheduler to produce x_{t-1} . A complete denoising trajectory $\tau = (x_T, a_T, x_{T-1}, \dots, a_1, x_0)$ constitutes one RL episode. The **reward** $R(x_0, c)$ is computed only on the final generated image x_0 —it is *sparse*, assigned at the end of the episode.

The U-Net acts as the **policy** $\pi_\theta(a_t | s_t, c)$: given the current noisy state and the text conditioning, it outputs an action (noise prediction). DDPO uses **Proximal Policy Optimization (PPO)** [12] to update the policy. PPO computes advantages (how much better each action was compared to the average) and updates the policy using a clipped surrogate objective:

$$\mathcal{J}(\theta) = \mathbb{E}_{\tau \sim p_\theta} [R(x_0, c)], \quad \nabla_\theta \mathcal{J} \approx \mathbb{E} \left[\sum_t \min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \hat{A}_t \right) \right] \quad (9)$$

where $r_t(\theta) = p_\theta(a_t | s_t, c) / p_{\theta_{\text{old}}}(a_t | s_t, c)$ is the importance sampling ratio and $\hat{A}_t$ is the estimated advantage. The clipping mechanism prevents any single update from changing the policy too drasti-

cally, which is critical for training stability. A KL divergence penalty against the original pretrained model prevents *reward hacking*—where the model produces degenerate images that exploit the reward function without producing visually meaningful outputs.

Unlike the other methods which learn from a fixed dataset, DDPO is **on-policy**: each training iteration generates fresh trajectories with the current policy, evaluates them against the reward function, and updates. The reward function can be *any* scoring mechanism—a CLIP similarity score for text alignment, an aesthetic predictor for visual quality, or a learned reward model trained on human preference data. This flexibility makes DDPO complementary to reconstruction-based methods rather than a replacement. Figure 8 illustrates the MDP formulation.

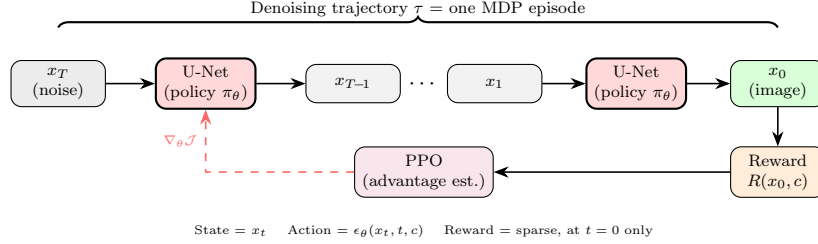


Figure 8: DDPO architecture. The denoising trajectory is treated as an MDP episode. The U-Net (policy) generates images; the reward function scores the final output; PPO distributes the reward signal back to each denoising step.

5.7 Method Summary

Table 1: Summary of what each method trains vs. freezes.

Method	What is trained	Trainable params	Storage	Key idea
DreamBooth	Entire U-Net	~860 M	~3.4 GB	Prior-preservation loss
LoRA	Low-rank adapters	~1.6–6.4 M	~3 MB	Inject $\Delta W = BA$
Textual Inv.	Embedding v^*	~768–6,144	~3–24 KB	Optimize pseudo-token
Custom Diff.	Cross-attn K, V + token	~57 M (~2%)	~75 MB	Selective fine-tuning
LCM	Student model	Full model	~3.4 GB	Consistency distillation
DDPO	U-Net (via RL)	Full model	~3.4 GB	MDP + PPO

6 Experimental Setup

All experiments use Stable Diffusion v1.5 (or v1.4 for Custom Diffusion) as the base model, run on NVIDIA H100 80 GB GPUs on the IU Quartz HPC cluster using HuggingFace Diffusers [8] and TRL [14] (for DDPO). Training and evaluation metadata are systematically tracked via an automated experiment tracker that records hyperparameters, hardware context (GPU, CUDA/PyTorch versions), and evaluation metrics into per-method CSV registries. For each method, I identify its most impactful hyperparameter and sweep over a range of values. Tables 2 and 3 summarize the configurations.

Table 2: Experimental configuration for DreamBooth, LoRA, and Textual Inversion. Common: 512×512 , seed 42, H100 80 GB.

	DreamBooth	LoRA	Textual Inversion
Task	Subject-driven (dog)	Style transfer (Naruto)	Subject-driven (cat)
Base model	SD v1.5	SD v1.5	SD v1.5
Dataset	3 dog images	naruto-blip-captions	3 cat images
Prompt	“a photo of sks dog”	BLIP captions	“a photo of ⟨sks-cat⟩”
Steps	400	15,000	800
Learning rate	5×10^{-6}	1×10^{-4}	5×10^{-4}
Batch / Grad acc.	1 / 1	1 / 4	1 / 4
Precision	fp16/bf16	bf16	bf16
Special config	100 class images, prior pres. ON	Checkpoint every 500 steps	Checkpoint every 200 steps
Variable swept	$\lambda \in \{0.25, 0.50, 0.75, 1.0\}$	Rank $r \in \{4, 8, 16\}$	Num vectors $\in \{1, 2, 4, 8\}$
Inference steps	30	30	30
Guidance scale	6.0–7.5	7.5	6.0
Metrics	CLIP-T, CLIP-I, DINO-I	CLIP-T only*	CLIP-T, CLIP-I, DINO-I

*Style transfer has no reference images for CLIP-I / DINO-I.

Table 3: Experimental configuration for Custom Diffusion, LCM, and DDPO. Common: 512×512 , seed 42, H100 80 GB.

	Custom Diffusion	LCM Distillation	DDPO
Task	Subject-driven (dog)	General acceleration	Aesthetic reward
Base model	SD v1.4 (CompVis)	SD v1.5	SD v1.5
Dataset	5 dog images	LAION-CC12M	RL-generated
Prompt	“⟨new1⟩ dog ...”	General prompts	Aesthetic prompts
Epochs / Steps	250 steps	10,000 steps	200 epochs
Learning rate	1×10^{-5}	1×10^{-6}	3×10^{-4}
Batch / Grad acc.	2 / 1	8 / 1	3 / 1 (sample: 6)
Precision	fp32 (train) / bf16 (eval)	bf16 (train) / fp16 (eval)	bf16 (train) / fp16 (eval)
Special config	Modifier token ⟨new1⟩, ckpt every 20 steps	Consistency distillation	PPO, aesthetic reward
Variable swept	<code>crossattn</code> vs. <code>crossattn_kv</code>	Loss: L2 vs. Huber	No LoRA vs. LoRA
Inference steps	30	4	50
Guidance scale	6.0	8.0	7.5
Metrics	CLIP-T, CLIP-I, DINO-I	CLIP-T, Latency [†]	Aesthetic, Latency [†] CLIP-T,

[†]LCM and DDPO generate generic images (no subject); CLIP-I/DINO-I not applicable.

Input Datasets. Subject-driven methods require a small set of reference images. **DreamBooth** and **Custom Diffusion** both use images of the same pet dog (from `diffusers/dog-example`): 3 images for DreamBooth and 5 for Custom Diffusion, plus 100 model-generated class images for prior preservation. **Textual Inversion** uses 3 images of a cat toy (from `diffusers/cat_toy_example`). **LoRA** fine-tunes on `lambdalabs/naruto-blip-captions` (~1,200 Naruto-style illustrations with BLIP captions) for style transfer. **LCM** performs consistency distillation on `LAION-CC12M`, a large-scale web-crawled image–text dataset. **DDPO** does not use a fixed training dataset; as an on-policy RL method, it trains entirely on its own generated images, scored by an aesthetic reward model at each iteration. Figure 9 shows representative input images for the subject-driven methods.

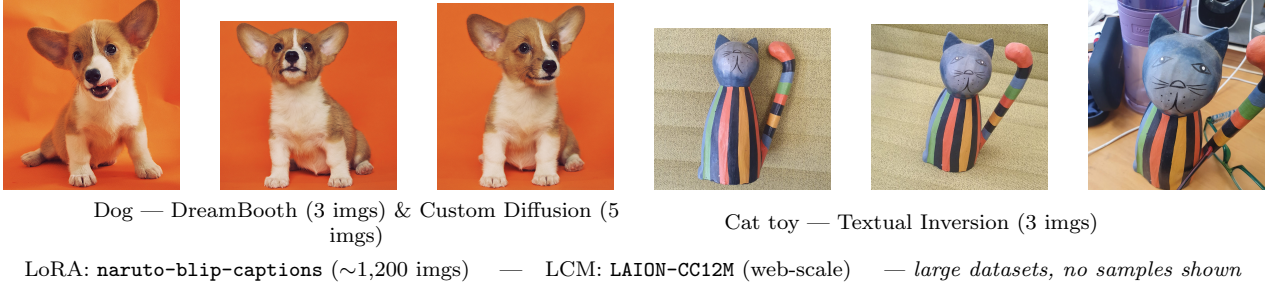


Figure 9: Input reference images for subject-driven methods. DreamBooth and Custom Diffusion use the same dog subject (3 and 5 images respectively). Textual Inversion uses a cat toy. LoRA and LCM use large-scale datasets.

Evaluation Prompts. Each method uses task-specific prompts: DreamBooth uses “sks dog in a bucket,” “sks dog sitting on a grassy field,” and “sks dog wearing a red bandana.” Textual Inversion uses “⟨sks-cat⟩ in a basket,” “on a wooden table,” and “in a garden.” LoRA uses character-style prompts (e.g., “Bill Gates with a hoodie,” “Hello Kitty with Naruto style”). Custom Diffusion uses “⟨new1⟩ dog in the park,” “⟨new1⟩ dog wearing sunglasses,” and “⟨new1⟩ dog as a cartoon.” LCM uses general prompts (“a cat sitting on a sofa,” “a car parked on a street,” etc.) since it targets inference speed rather than subject fidelity.

DDPO uses aesthetic prompts (“a beautiful portrait of a person with soft lighting,” “a cinematic landscape with mountains at sunset,” “a high quality photo of a dog in a park,” “a futuristic city at night with neon lights”) to evaluate reward-driven generation quality. The model is trained using the TRL library’s [14] DDPO implementation with an aesthetic reward function.

7 Results

7.1 DreamBooth Results

I sweep the prior-preservation weight $\lambda \in \{0.25, 0.50, 0.75, 1.0\}$ with 3 dog images, 400 training steps, and evaluate with 3 prompts. Table 4 presents the aggregate metrics.

Table 4: DreamBooth: effect of prior weight λ .

λ	CLIP-T	CLIP-I	DINO-I
0.25	0.274	0.823	0.588
0.50	0.272	0.823	0.560
0.75	0.269	0.845	0.564
1.00	0.269	0.800	0.462

Table 5: DreamBooth per-prompt metrics: $\lambda = 0.25$ (best CLIP-T/DINO-I) vs. $\lambda = 0.75$ (best CLIP-I).

Prompt	$\lambda = 0.25$			$\lambda = 0.75$		
	CLIP-T	CLIP-I	DINO-I	CLIP-T	CLIP-I	DINO-I
sks dog in a bucket	0.300	0.819	0.765	0.300	0.858	0.735
sks dog on grassy field	0.259	0.808	0.398	0.264	0.785	0.227
sks dog wearing bandana	0.262	0.841	0.601	0.243	0.893	0.731

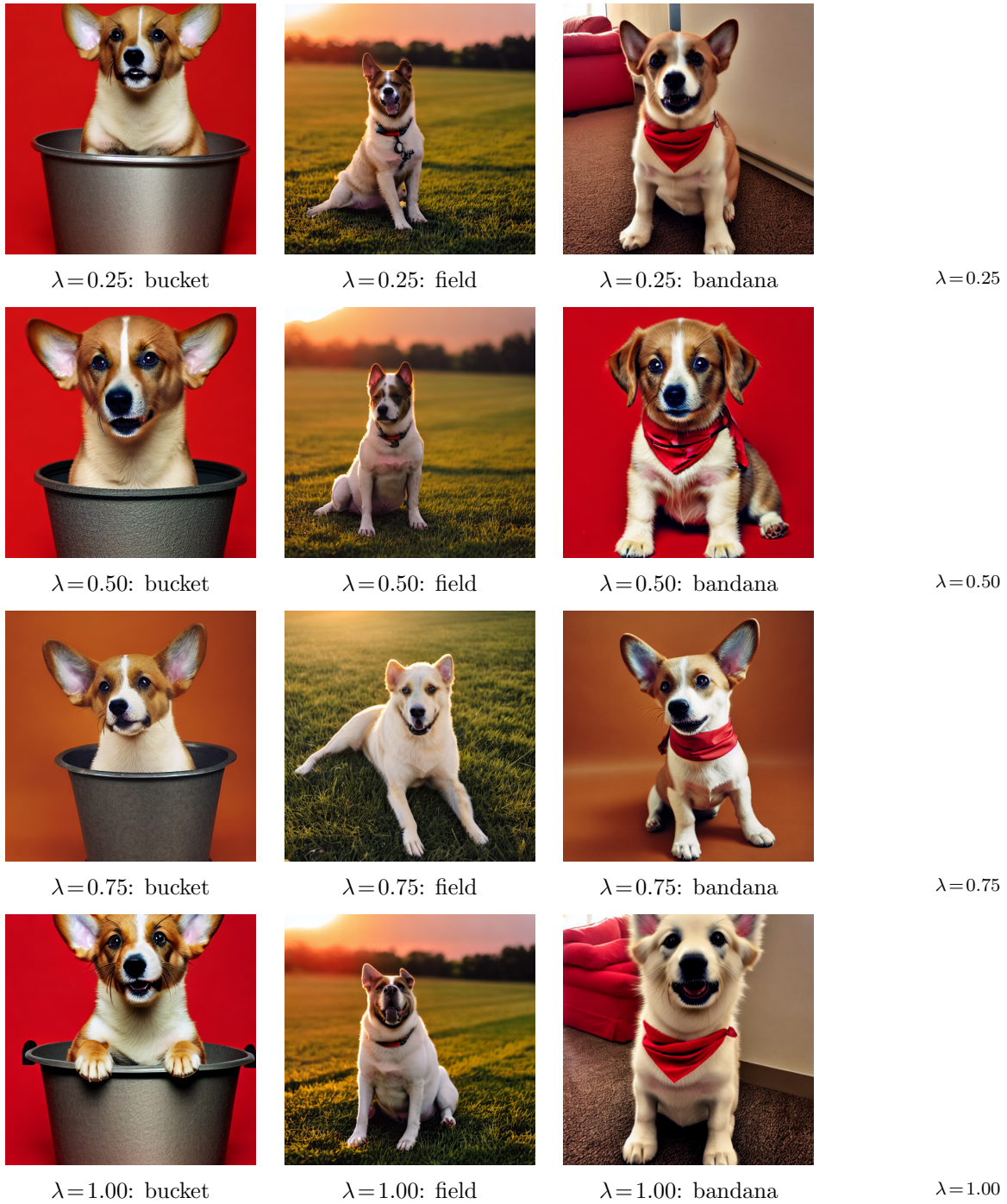


Figure 10: DreamBooth samples across $\lambda \in \{0.25, 0.50, 0.75, 1.0\}$. Lower λ preserves subject features; $\lambda = 1.0$ over-regularizes.

7.2 LoRA Results

I sweep rank $r \in \{4, 8, 16\}$ for Naruto-style transfer with 15,000 training steps. Since this is a style transfer task with no reference subject, only CLIP-T is reported. Table 6 shows per-prompt scores.

Table 6: LoRA: CLIP-T scores across ranks (style transfer – no CLIP-I/DINO-I). Best per prompt **bolded**.

Prompt	Rank 4	Rank 8	Rank 16
Bill Gates with a hoodie	0.242	0.172	0.172
John Oliver with Naruto style	0.287	0.290	0.155
Hello Kitty with Naruto style	0.314	0.277	0.253
Mickael Jackson as a ninja	0.154	0.238	0.154
Mean CLIP-T	0.249	0.244	0.183



Figure 11: LoRA Naruto-style samples. Rank 4–8 produce coherent outputs; rank 16 collapses to black on several prompts, indicating overfitting at higher rank with 15k steps.

7.3 Textual Inversion Results

I sweep the number of embedding vectors $\in \{1, 2, 4, 8\}$ with 3 cat images, 800 training steps, and evaluate with 3 prompts. Table 7 presents the aggregate metrics.

Table 7: Textual Inversion: effect of num vectors.

Vectors	CLIP-T	CLIP-I	DINO-I
1	0.241	0.798	0.628
2	0.237	0.819	0.664
4	0.271	0.857	0.687
8	0.277	0.834	0.690

Table 8: Textual Inversion per-prompt metrics: 4 vectors (best CLIP-I) vs. 8 vectors (best CLIP-T/DINO-I).

Prompt	4 Vectors			8 Vectors		
	CLIP-T	CLIP-I	DINO-I	CLIP-T	CLIP-I	DINO-I
$\langle \text{sks-cat} \rangle$ in a basket	0.256	0.859	0.657	0.239	0.855	0.735
$\langle \text{sks-cat} \rangle$ on a table	0.288	0.855	0.678	0.293	0.843	0.659
$\langle \text{sks-cat} \rangle$ in a garden	0.270	0.856	0.725	0.297	0.805	0.678

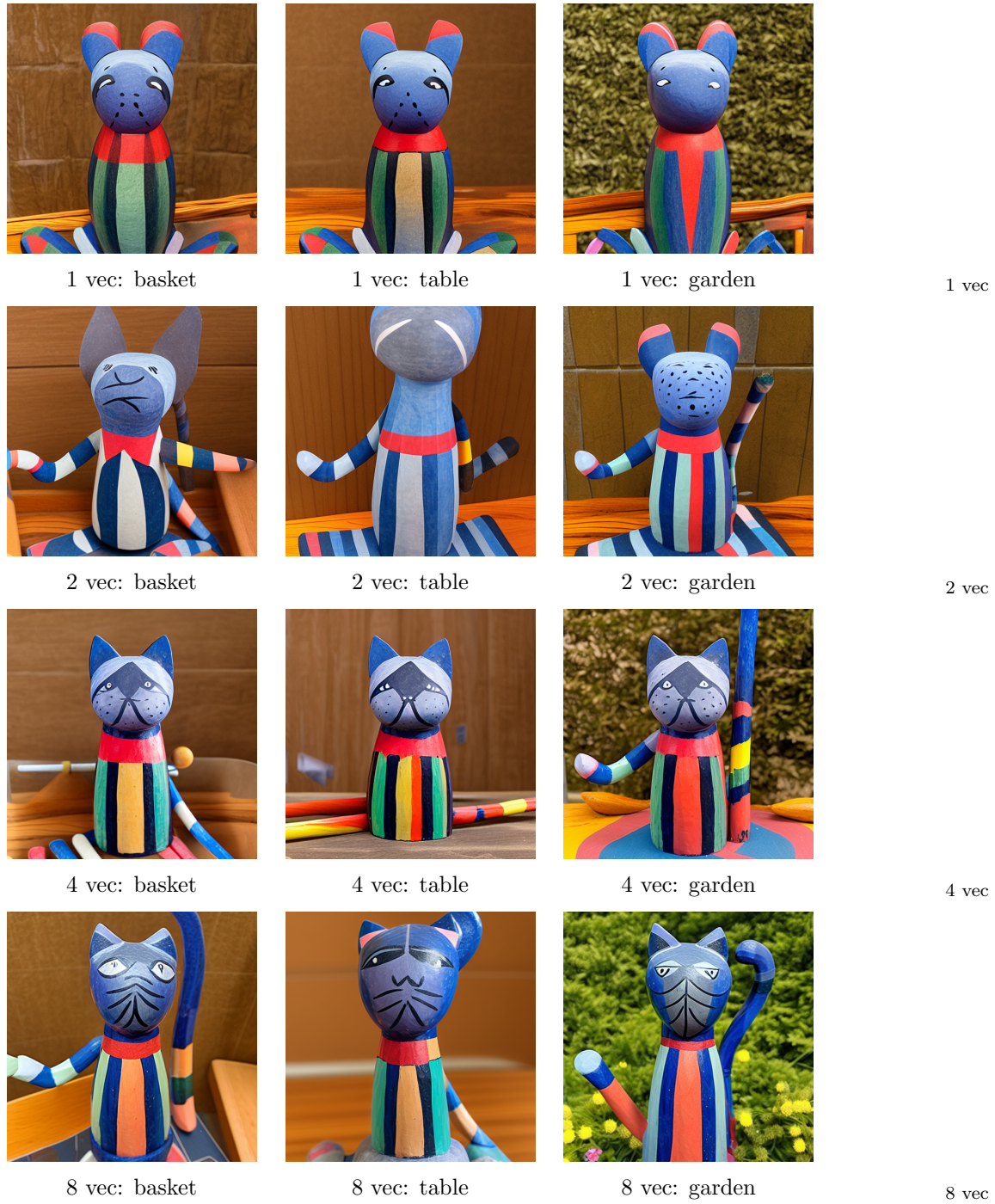


Figure 12: Textual Inversion samples across 1–8 embedding vectors. More vectors yield better subject fidelity; 1 vector has insufficient capacity.

7.4 Custom Diffusion Results

I evaluate two Custom Diffusion variants: **crossattn** (trains all cross-attention projections: Q, K, V, and output) and **crossattn_kv** (trains only the K and V projections plus the modifier token). Both use 5 dog images, 250 steps, SD v1.4, and modifier token `<new1>`. Table 9 presents the aggregate results.

Table 9: Custom Diffusion: `crossattn` vs. `crossattn_kv` (5 dog images, 250 steps, SD v1.4).

Variant	CLIP-T	CLIP-I	DINO-I	Latency (ms)
<code>crossattn</code>	0.258	0.723	0.005	645
<code>crossattn_kv</code>	0.257	0.741	0.081	644

Table 10: Custom Diffusion per-prompt metrics for both variants.

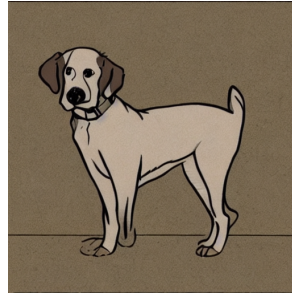
Prompt	<code>crossattn</code>			<code>crossattn_kv</code>		
	CLIP-T	CLIP-I	DINO-I	CLIP-T	CLIP-I	DINO-I
<code><new1></code> dog in the park	0.241	0.775	0.010	0.243	0.779	0.007
<code><new1></code> dog w/ sunglasses	0.287	0.714	0.070	0.287	0.759	0.229
<code><new1></code> dog as a cartoon	0.245	0.681	-0.064	0.240	0.685	0.006



kv: park



kv: sunglasses



kv: cartoon

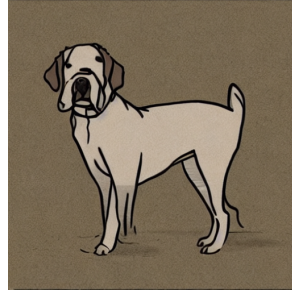
`crossattn_kv`
CLIP-T: 0.257
CLIP-I: 0.741
DINO-I: 0.081



all: park



all: sunglasses



all: cartoon

`crossattn`
CLIP-T: 0.258
CLIP-I: 0.723
DINO-I: 0.005

Figure 13: Custom Diffusion samples. Top: `crossattn_kv` (K, V only); Bottom: `crossattn` (all cross-attention). The `crossattn_kv` variant achieves better subject fidelity (higher CLIP-I and DINO-I) with fewer trainable parameters.

7.5 LCM Distillation Results

LCM is a general acceleration method that does not learn a specific subject, so CLIP-I and DINO-I (which require reference images) are not applicable. However, I evaluate **CLIP-T** to measure how well the generated images match their text prompts, alongside inference latency. I compare two distillation loss variants, L2 and Huber ($c=0.001$), both trained for 10k steps on LAION-CC12M from SD v1.5. Table 11 presents the per-prompt results.

Table 11: LCM per-prompt CLIP-T and latency: L2 vs. Huber loss (10k steps, SD v1.5, 4 inference steps, guidance 8.0).

Prompt	L2 CLIP-T	Huber CLIP-T	L2 (ms)	Huber (ms)
a cat sitting on a sofa	0.280	0.266	576 [†]	17,063 [†]
a car parked on a street	0.191	0.247	108	111
a bowl of fruit on a table	0.293	0.257	106	107
a person riding a bicycle	0.243	0.232	106	107
Mean	0.252	0.250	224	4,347
Mean (excl. warmup)	—	—	107	108

[†]First image includes GPU warmup overhead (varies across runs).

Both loss variants achieve comparable CLIP-T scores (L2: 0.252, Huber: 0.250), indicating similar text-image alignment quality. L2 shows higher variance across prompts (0.191–0.293) while Huber is more consistent (0.232–0.266). Both converge to identical steady-state latency (~ 108 ms per image) after GPU warmup; the difference in mean latency is entirely due to varying warmup overhead across runs.

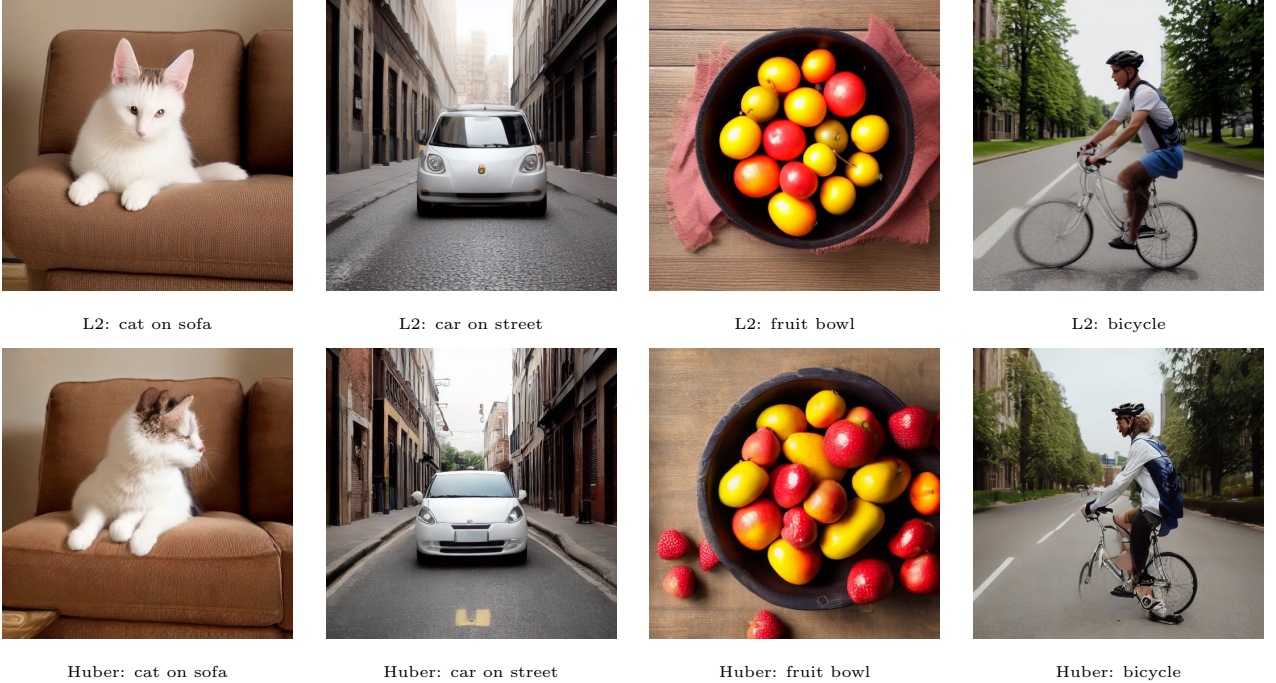


Figure 14: LCM samples generated in only 4 denoising steps. Top row: L2 loss; Bottom row: Huber loss ($c = 0.001$). Both produce recognizable images with a $\sim 27\times$ speedup over standard 30-step inference.

7.6 DDPO Results

I compare two DDPO variants: **No LoRA** (full U-Net RL fine-tuning) and **LoRA** (RL fine-tuning with low-rank adapters). Both are trained for 200 epochs with an aesthetic reward function on SD v1.5 and evaluated with 50 inference steps at guidance scale 7.5.

Table 12: DDPO per-prompt results: No-LoRA (full U-Net) vs. LoRA, 200 epochs, SD v1.5, 50 inference steps, guidance 7.5.

Prompt	Aesthetic Score		CLIP-T		Latency (ms)	
	No-LoRA	LoRA	No-LoRA	LoRA	No-LoRA	LoRA
portrait (soft lighting)	6.14	5.67	0.208	0.217	3,629 [†]	16,589 [†]
landscape (mountains)	6.32	6.57	0.252	0.251	1,216	1,190
dog in a park	5.12	6.78	0.205	0.204	1,199	1,190
futuristic city (neon)	6.61	6.09	0.260	0.258	1,202	1,181
Mean	6.05	6.28	0.231	0.232	1,812	5,038
Mean (excl. warmup)	—	—	—	—	1,206	1,187

[†]First image includes GPU warmup overhead (varies across runs).

The LoRA variant achieves a higher mean aesthetic score (6.28 vs. 6.05) despite training fewer parameters, suggesting that LoRA’s constrained parameter space acts as a regularizer that favors aesthetic coherence. Both variants produce comparable CLIP-T scores (~ 0.23), indicating similar text-image alignment. Steady-state latency is nearly identical ($\sim 1,190$ ms) since LoRA weights are merged at inference time.

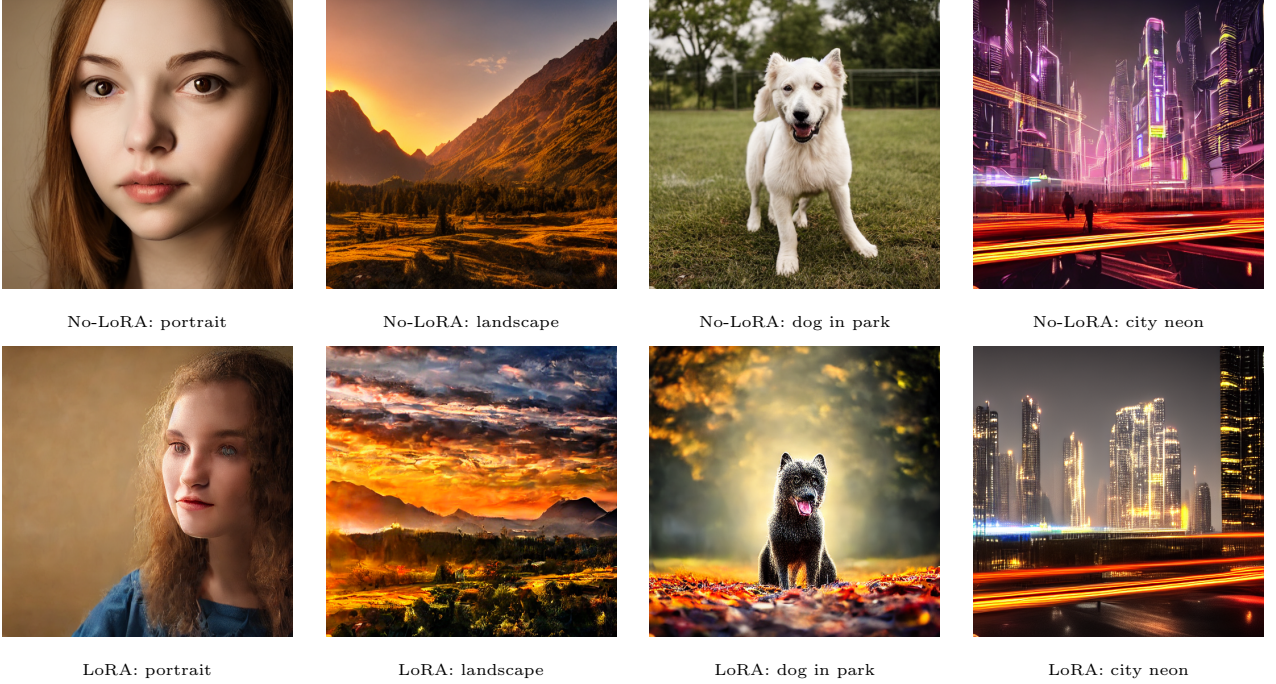


Figure 15: DDPO generated images (50 steps, guidance 7.5). Top row: No-LoRA (full U-Net RL). Bottom row: LoRA. The LoRA variant achieves higher aesthetic scores on most prompts despite training fewer parameters.

7.7 Latency Comparison

Table 13 compares inference latency across all methods on the same hardware.

Table 13: Inference latency comparison across methods (single image, 512×512 , H100 80 GB).

Method	Inference Steps	Mean Latency (ms)	Speedup vs. DB
DreamBooth ($\lambda=0.75$)	30	2,913	1.0 $\times$
DDPO (LoRA)	50	1,187*	2.5 $\times$
LoRA ($r=8$)	30	835	3.5 $\times$
Custom Diff. (kv)	30	644	4.5 $\times$
Textual Inv. (4 vec)	30	613	4.8 $\times$
LCM (L2)	4	108*	27.0$\times$

*Excluding warmup.

7.8 Metric Trends

Figure 16 visualizes how each method’s key hyperparameter affects generation quality.

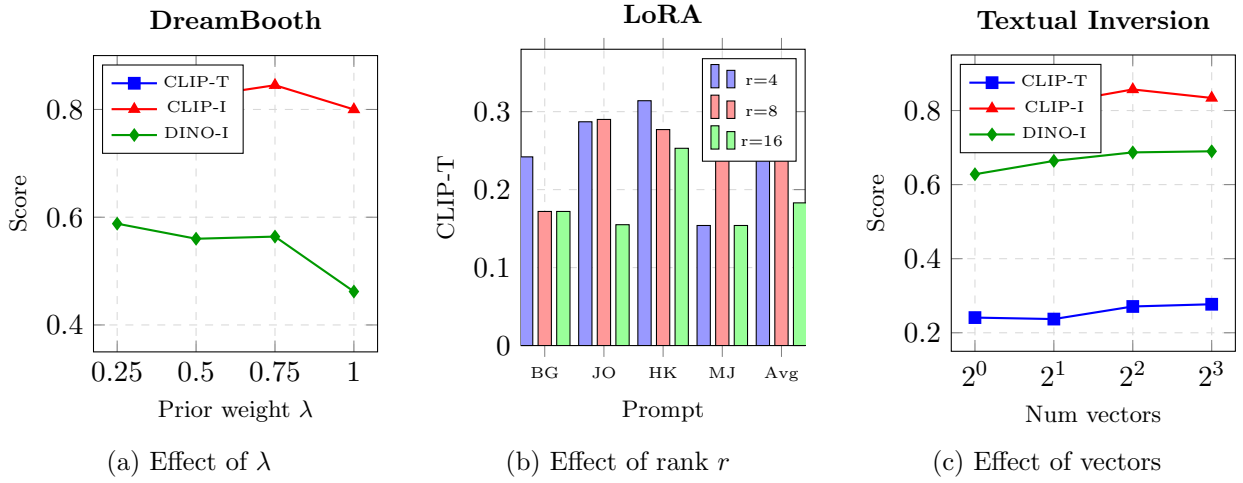


Figure 16: Metric trends for DreamBooth, LoRA, and Textual Inversion. Each plot shows how the key hyperparameter affects generation quality. LoRA only reports CLIP-T (style transfer task, no reference images).

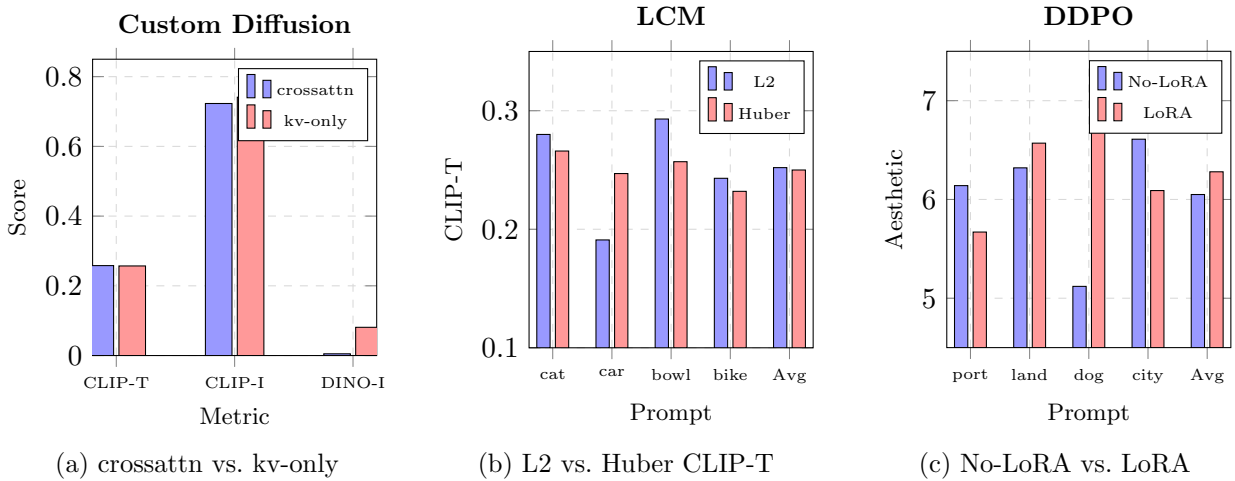


Figure 17: Metric trends for Custom Diffusion, LCM, and DDPO. CD: K/V-only outperforms full cross-attention on CLIP-I and DINO-I. LCM: L2 shows higher per-prompt variance in CLIP-T than Huber. DDPO: LoRA achieves higher mean aesthetic score than full U-Net RL.

8 Analysis and Discussion

8.1 Cross-Method Comparison

Table 14 compares all methods across key dimensions. Note that DreamBooth and Textual Inversion are evaluated on subject-driven generation (with CLIP-I/DINO-I), while LoRA is evaluated on style transfer (CLIP-T only), so direct numeric comparison between them should be interpreted with this context.

Table 14: Cross-method comparison. Best values per metric **bolded**.

Property	DreamBooth	LoRA	Text. Inv.	Custom Diff.	LCM	DDPO
Trained Params	Entire U-Net ~860M	Low-rank adapt. ~1.6–6.4M	Embedding ~768–6K	K, V + token ~57M	Student Full	U-Net (RL) Full
Storage	~3.4 GB	~3 MB	~3–24 KB	~75 MB	~3.4 GB	~3.4 GB
CLIP-T	0.274	0.249	0.277	0.258	0.252	0.232
CLIP-I	0.845	N/A	0.857	0.741	N/A [†]	N/A [†]
DINO-I	0.588	N/A	0.690	0.081	N/A [†]	N/A [†]
Aesthetic	—	—	—	—	—	6.28
Latency	2,913 ms	835 ms	613 ms	644 ms	108 ms	1,187 ms
Overhead	None	None (merged)	None	None	27× faster	None

[†]LCM and DDPO generate generic images (no subject); CLIP-I/DINO-I not applicable.

8.2 Key Findings by Method

DreamBooth. Lower λ (0.25) yields the highest CLIP-T (0.274) and DINO-I (0.588); reduced regularization lets the model capture subject-specific structural details. $\lambda = 0.75$ maximizes CLIP-I (0.845), preserving pixel-level visual characteristics. $\lambda = 1.0$ causes *over-regularization* with DINO-I dropping to 0.462, suppressing subject features in favor of generic class characteristics. The sweet spot is $\lambda \in [0.50, 0.75]$.

LoRA. Rank 4 achieves the highest mean CLIP-T (0.249) with only ~1.6M trainable parameters. Rank 8 performs comparably (0.244) and outperforms on some prompts, suggesting prompt-dependent optimal capacity. Rank 16 suffers *mode collapse*, producing black/collapsed outputs for multiple prompts (mean CLIP-T drops to 0.183). Rank 4–8 is optimal for this dataset; increasing rank beyond the effective dimensionality of the style manifold is counterproductive.

Textual Inversion. 1 vector (768 params) has insufficient capacity, yielding the lowest scores. 4 vectors achieves peak CLIP-I (0.857); 8 vectors peaks on CLIP-T (0.277) and DINO-I (0.690). The improvement from 1→4 vectors is substantial, while 4→8 shows diminishing returns. Remarkably, with only ~3–24 KB of storage, Textual Inversion achieves competitive or superior CLIP-I and DINO-I compared to DreamBooth’s ~3.4 GB.

Custom Diffusion. The `crossattn_kv` variant (K, V projections only) outperforms the full `crossattn` variant on both CLIP-I (0.741 vs. 0.723) and DINO-I (0.081 vs. 0.005), despite training fewer parameters. This suggests that training additional projections (Q, output) introduces noise that harms subject fidelity at 250 steps. The low DINO-I scores overall ($\ll 0.5$) indicate that 250 training steps are insufficient for Custom Diffusion to capture fine-grained structural identity—the method would likely benefit from longer training or more reference images. Per-prompt analysis reveals that the “sunglasses” prompt achieves the highest DINO-I (0.229 for `crossattn_kv`), suggesting the model captures subject identity better in structured scenes.

LCM Distillation. LCM achieves a dramatic ~27× latency reduction (108 ms vs. 2,913 ms for DreamBooth, excluding warmup) by reducing inference from 30 steps to just 4. Both the L2 and Huber ($c=0.001$) loss variants converge to the same steady-state latency (~108 ms); the mean-latency difference is attributable entirely to variable GPU warmup on the first call, not the loss function. Both variants also achieve comparable CLIP-T scores (L2: 0.252, Huber: 0.250), confirming that

the loss function choice does not significantly affect text-image alignment. This makes LCM the only method suitable for real-time or interactive applications.

DDPO. The LoRA variant achieves a higher mean aesthetic score (6.28 vs. 6.05 for full U-Net fine-tuning), despite training fewer parameters. This indicates that LoRA’s constrained parameter space acts as a regularizer that steers the reward optimization toward aesthetically coherent outputs rather than overfitting to the reward signal. Both variants produce comparable CLIP-T (~ 0.23) and steady-state latency ($\sim 1,190$ ms at 50 steps). The “dog in a park” prompt shows the largest improvement with LoRA (aesthetic: 6.78 vs. 5.12), while the “portrait” prompt is the only case where no-LoRA outperforms (6.14 vs. 5.67).

8.3 Fidelity–Efficiency Trade-off

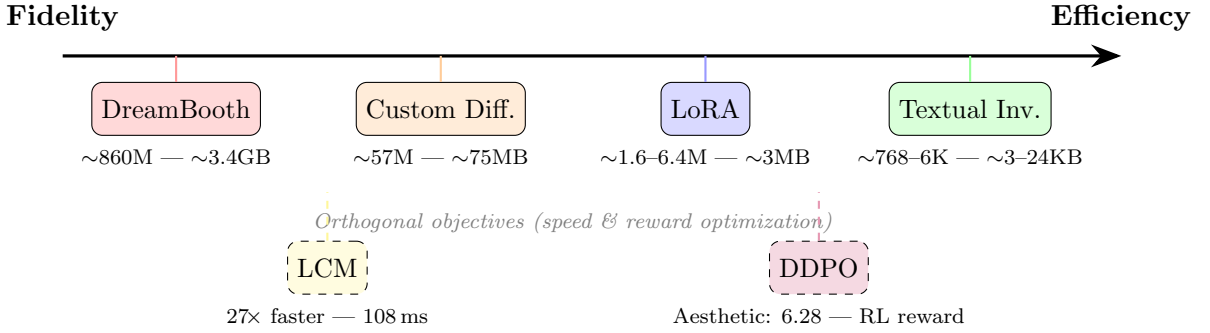


Figure 18: Fidelity–efficiency trade-off spectrum. Top row: subject-driven methods ordered by parameter efficiency. Bottom row: LCM and DDPO target orthogonal objectives (inference speed and reward optimization respectively) rather than the fidelity–efficiency axis.

8.4 When to Use Each Method

- **DreamBooth:** High-fidelity subject reconstruction when storage/compute costs are acceptable (product photography, pet portraits).
- **LoRA:** Multiple styles/concepts at scale with easy sharing, $\sim 1000\times$ smaller than DreamBooth, zero inference overhead.
- **Textual Inversion:** Minimal storage, quick experimentation, lightweight concept injection while keeping the model completely frozen.
- **Custom Diffusion:** Multi-concept composition needed: learn a dog and a couch together, then compose in one prompt. More expressive than TI, more efficient than DreamBooth.
- **LCM:** Inference speed critical: 1–4 steps instead of 20–50, enabling real-time applications.
- **DDPO:** Optimization objective beyond reconstruction loss (aesthetic scores, human preferences, task-specific rewards).

9 Conclusion

This report presents a systematic comparative study of six personalization methods for text-to-image diffusion models using HuggingFace Diffusers on Stable Diffusion v1.4/v1.5. Key findings:

1. **DreamBooth** achieves the highest subject fidelity by fine-tuning all $\sim 860\text{M}$ U-Net weights. Optimal prior-preservation weight: $\lambda \in [0.50, 0.75]$; $\lambda = 1.0$ over-regularizes (DINO-I $\downarrow 0.462$).

2. **LoRA** provides an excellent fidelity–efficiency trade-off with $\sim 1000\times$ smaller adapters. Rank 4–8 optimal for style transfer; rank 16 causes mode collapse at 15k steps.
3. **Textual Inversion** is the most parameter-efficient method (3–24 KB). With 4–8 vectors, it achieves competitive CLIP-I (0.857) and DINO-I (0.690) while keeping the entire model frozen.
4. **Custom Diffusion** enables efficient selective fine-tuning of cross-attention layers (~ 75 MB). The K/V-only variant outperforms full cross-attention training (CLIP-I: 0.741 vs. 0.723), demonstrating that training fewer, more targeted parameters can be more effective. Multi-concept composition is a unique capability not available in other methods.
5. **LCM Distillation** achieves a $\sim 27\times$ inference speedup (108 ms vs. 2,913 ms) by distilling the denoising process into 4 steps, making it the only viable method for real-time applications.
6. **DDPO** demonstrates that RL-based optimization with an aesthetic reward can improve image quality beyond what reconstruction-based training achieves. The LoRA variant (aesthetic: 6.28) outperforms full U-Net fine-tuning (6.05), suggesting that constrained parameter spaces serve as effective regularizers for reward-driven optimization.

All experiments were conducted on NVIDIA H100 80 GB GPUs on the IU Quartz HPC cluster. All training scripts, evaluation pipelines, and experiment data are publicly available at <https://github.com/meghanaNanuvala/Diffusion-Personalization/> for reproducibility.

Disclaimer

A large language model (LLM) was used as an assistive tool during the preparation of this report. Specifically, it was used to improve the clarity and readability of the writing, to help compose mathematical equations and notation, and to resolve formatting and compilation conflicts in L^AT_EX. All experimental work—including model training, evaluation, and analysis—was conducted independently by the author.

References

- [1] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *NeurIPS*, 2020.
- [2] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. *CVPR*, 2022.
- [3] N. Ruiz, Y. Li, V. Jampani, Y. Pritch, M. Rubinstein, and K. Aberman. DreamBooth: Fine tuning text-to-image diffusion models for subject-driven generation. *CVPR*, 2023. <https://arxiv.org/abs/2208.12242>.
- [4] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. LoRA: Low-rank adaptation of large language models. *ICLR*, 2022. <https://arxiv.org/abs/2106.09685>.
- [5] R. Gal, Y. Alaluf, Y. Atzmon, O. Patashnik, A. H. Bermano, G. Chechik, and D. Cohen-Or. An image is worth one word: Personalizing text-to-image generation using textual inversion. *ICLR*, 2023. <https://arxiv.org/abs/2208.01618>.
- [6] S. Luo, Y. Tan, L. Huang, J. Li, and H. Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference. *arXiv:2310.04378*, 2023.
- [7] K. Black, M. Janner, Y. Du, I. Kostrikov, and S. Levine. Training diffusion models with reinforcement learning. *arXiv:2305.13301*, 2024.

- [8] P. von Platen, S. Patil, A. Lozhkov, et al. Diffusers: State-of-the-art diffusion models. <https://github.com/huggingface/diffusers>, 2023.
- [9] HuggingFace. Latent Consistency Distillation — Diffusers Documentation. https://huggingface.co/docs/diffusers/training/lcm_distill, 2024.
- [10] A. Radford, J. W. Kim, C. Hallacy, et al. Learning transferable visual models from natural language supervision. *ICML*, 2021.
- [11] M. Oquab, T. Darcet, T. Moutakanni, et al. DINOv2: Learning robust visual features without supervision. *arXiv:2304.07193*, 2023.
- [12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.
- [13] N. Kumari, B. Zhang, R. Zhang, E. Shechtman, and J.-Y. Zhu. Multi-concept customization of text-to-image diffusion. *CVPR*, 2023. <https://arxiv.org/abs/2212.04488>.
- [14] L. von Werra, Y. Belkada, L. Tunstall, E. Beeching, T. Le Scao, et al. TRL: Transformer Reinforcement Learning. <https://github.com/huggingface/trl>, 2023.