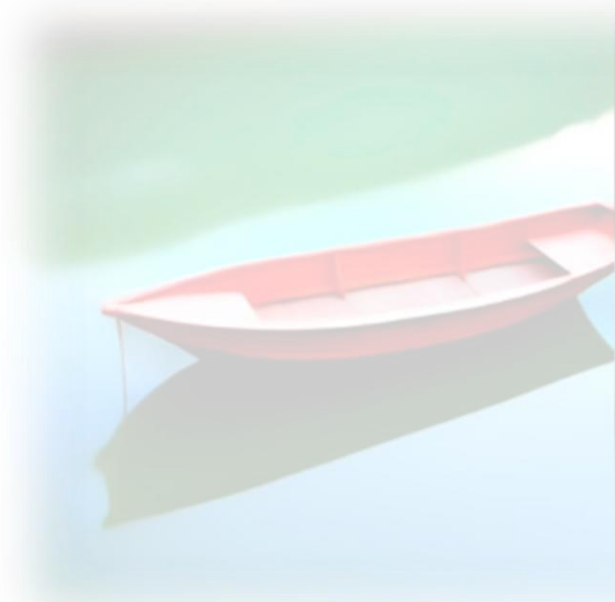




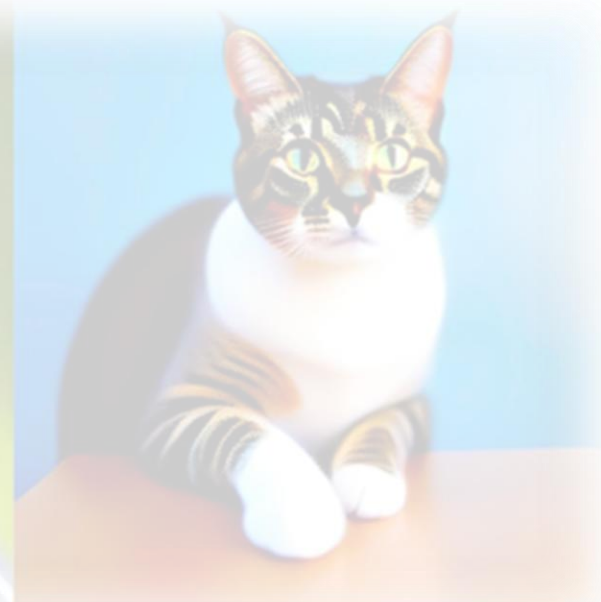
"A boat on a river"



"A burger on a plate"



"A car driving on a road"

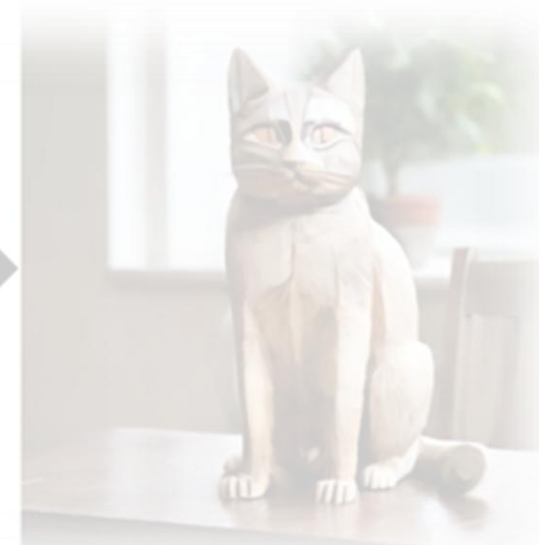
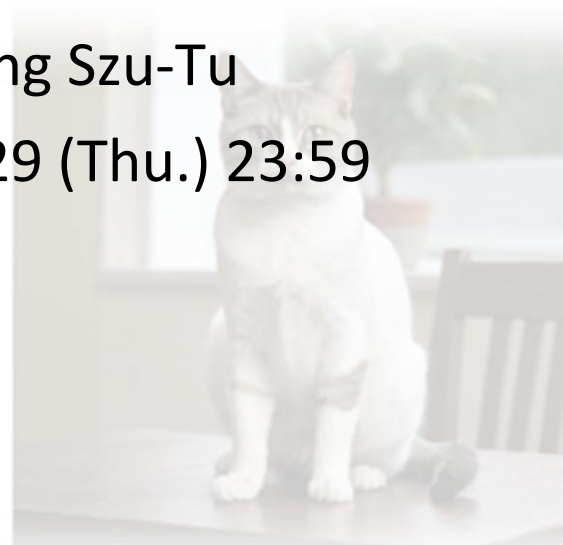


"A cat sitting on the table"

Lab3 - Distillation

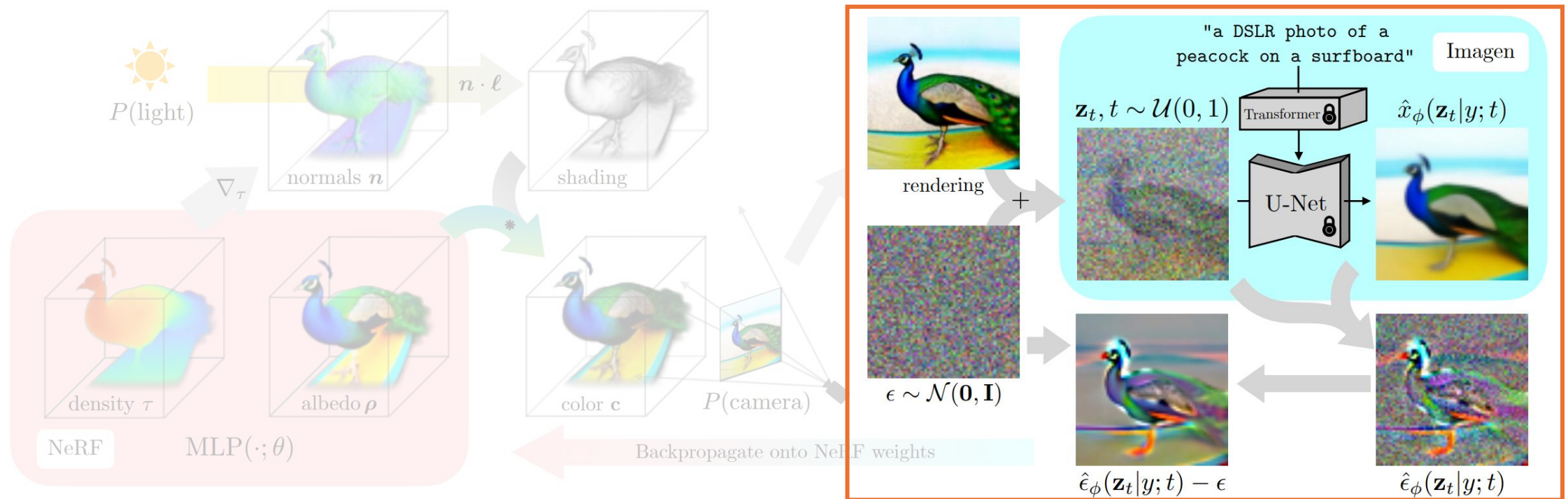


TA: Li-Zhong Szu-Tu
Due: 2026/10/29 (Thu.) 23:59



Introduction

In this assignment, we will focus on core components of **SDS**, **VSD**, and **SDI** used to obtain gradients/loss from pretrained diffusion models.

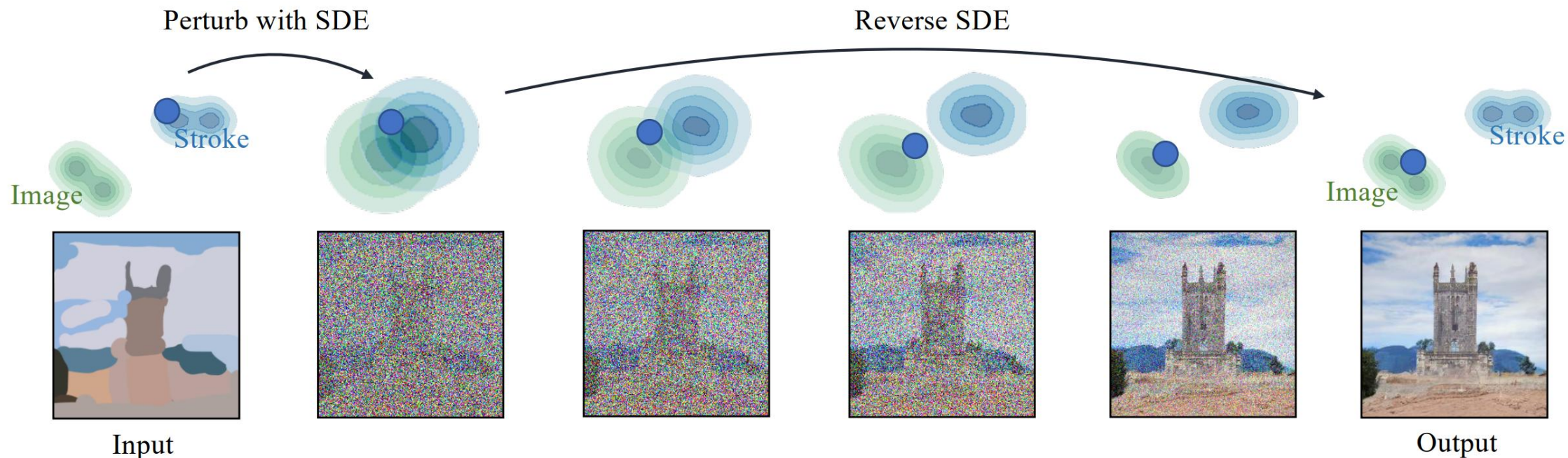


DreamFusion, Poole *et al.*, ICLR 2023

We will focus on this part!

Preliminary - SDEdit

Add partial noise to a guided input, then denoise it to realism.



SDEdit, Meng *et al.*, ICLR 2022

Preliminary - DDIM Inversion

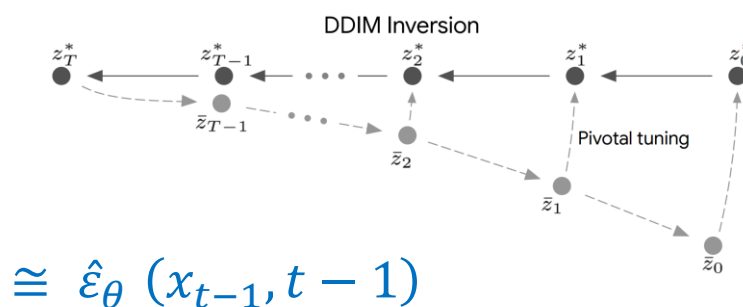
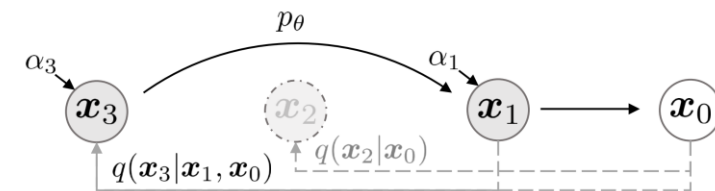
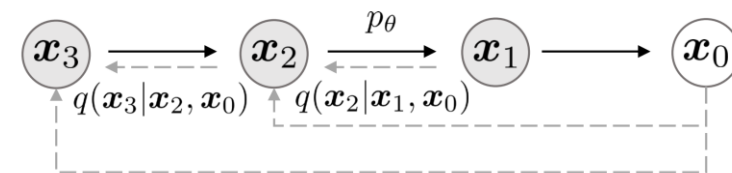
DDIM: One can generate a sample x_{t-1} from a sample x_t via:

$$x_{t-1} = \sqrt{\bar{\alpha}_{t-1}} \frac{x_t - \sqrt{1 - \bar{\alpha}_t} \hat{\epsilon}_\theta(x_t, t)}{\sqrt{\bar{\alpha}_t}} + \sqrt{1 - \bar{\alpha}_{t-1}} \hat{\epsilon}_\theta(x_t, t)$$

DDIM Inversion: Exchange samples x_{t-1} and x_t :

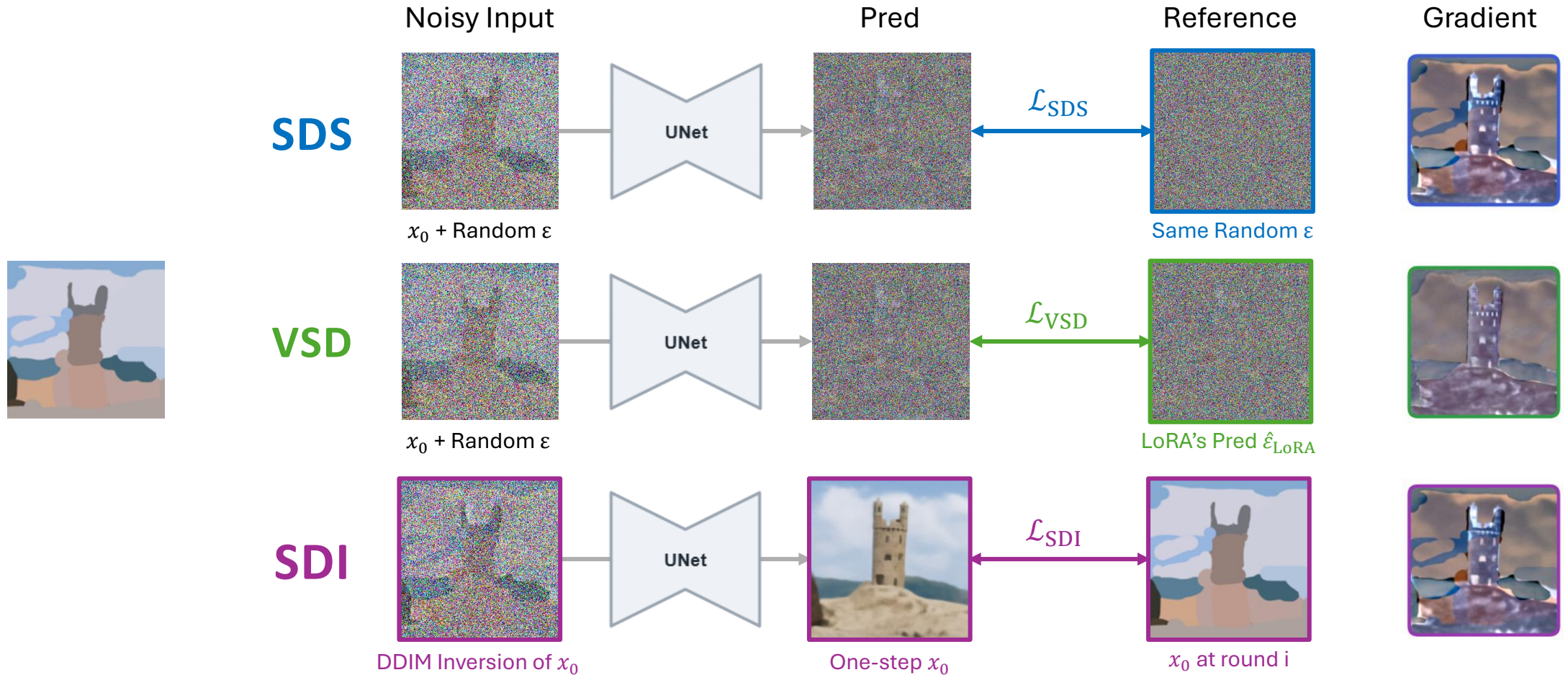
$$\Rightarrow x_t = \sqrt{\frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}} \left[x_{t-1} - \sqrt{1 - \bar{\alpha}_{t-1}} \hat{\epsilon}_\theta(x_t, t) \right] + \sqrt{1 - \bar{\alpha}_t} \hat{\epsilon}_\theta(x_t, t)$$

$$x_t = \sqrt{\frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}} x_{t-1} + \sqrt{\bar{\alpha}_t} \left(\sqrt{\frac{1}{\bar{\alpha}_t}} - 1 - \sqrt{\frac{1}{\bar{\alpha}_{t-1}}} - 1 \right) \hat{\epsilon}_\theta(x_t, t)$$



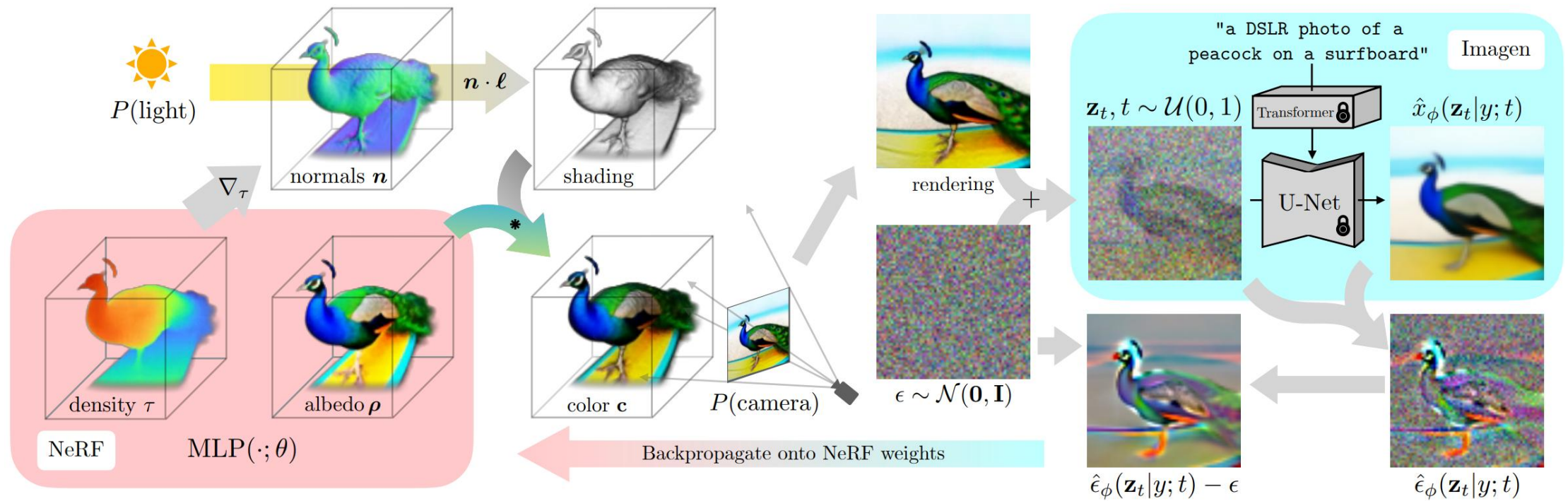
Overview

Same UNet but different noisy inputs and reference targets.



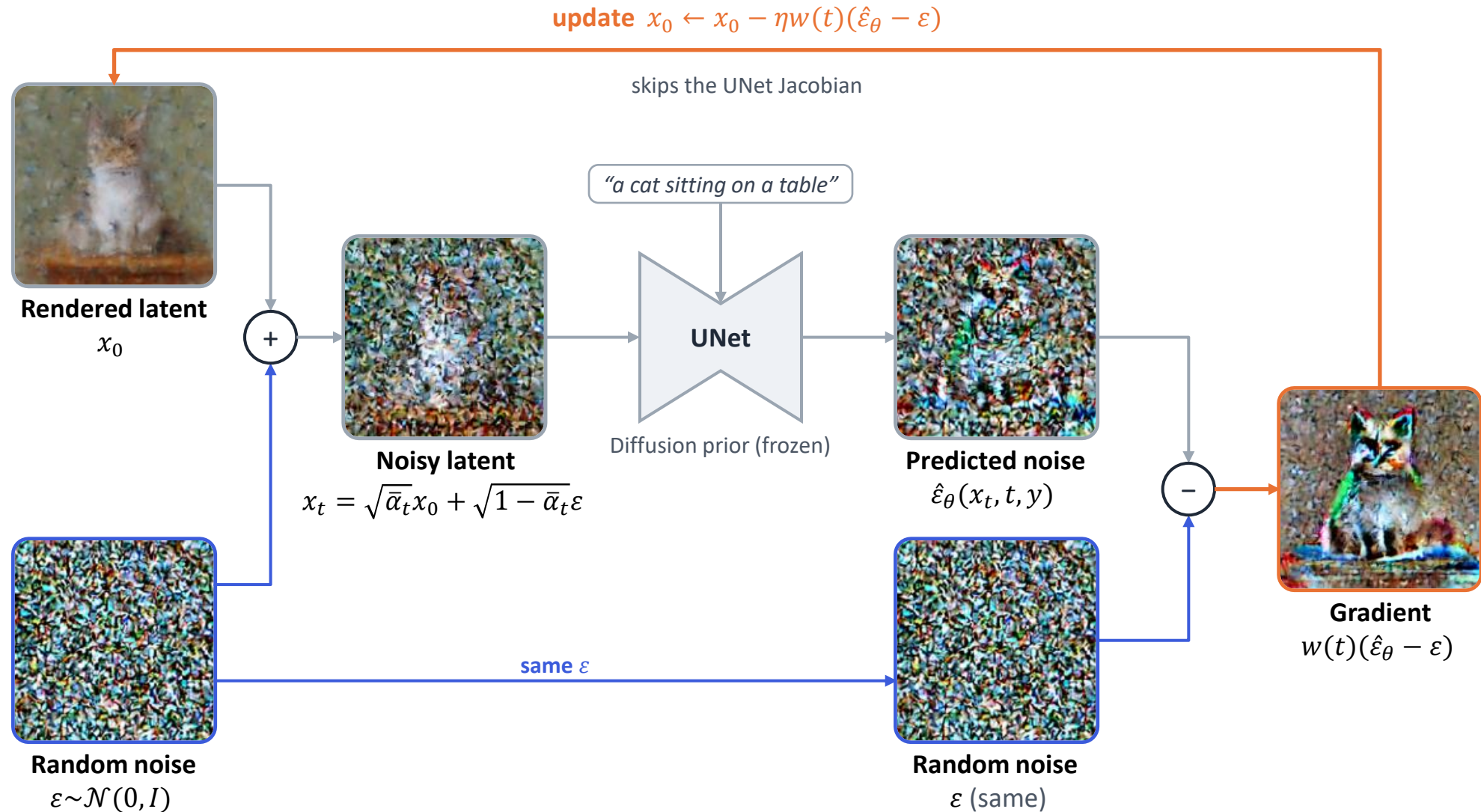
Task1 - Score Distillation Sampling (SDS)

Distill knowledge from a pretrained diffusion model to optimize $g(\theta; c)$.



DreamFusion, Poole *et al.*, ICLR 2023

Task1 - Score Distillation Sampling (SDS)



Note: For clarity, all images shown here are decoded to pixel space; the actual computation is performed in latent space.

Task1 - Score Distillation Sampling (SDS)

```
sd.py

def get_sds_loss(self, latents, text_embeddings, guidance_scale=7.5):
    """
    Score Distillation Sampling (SDS) loss.

    Reference: DreamFusion (https://arxiv.org/abs/2209.14988)

    Adds random noise to the latents at a random timestep and lets the
    diffusion model predict it. The difference between predicted and added
    noise is used as the gradient on the latents.

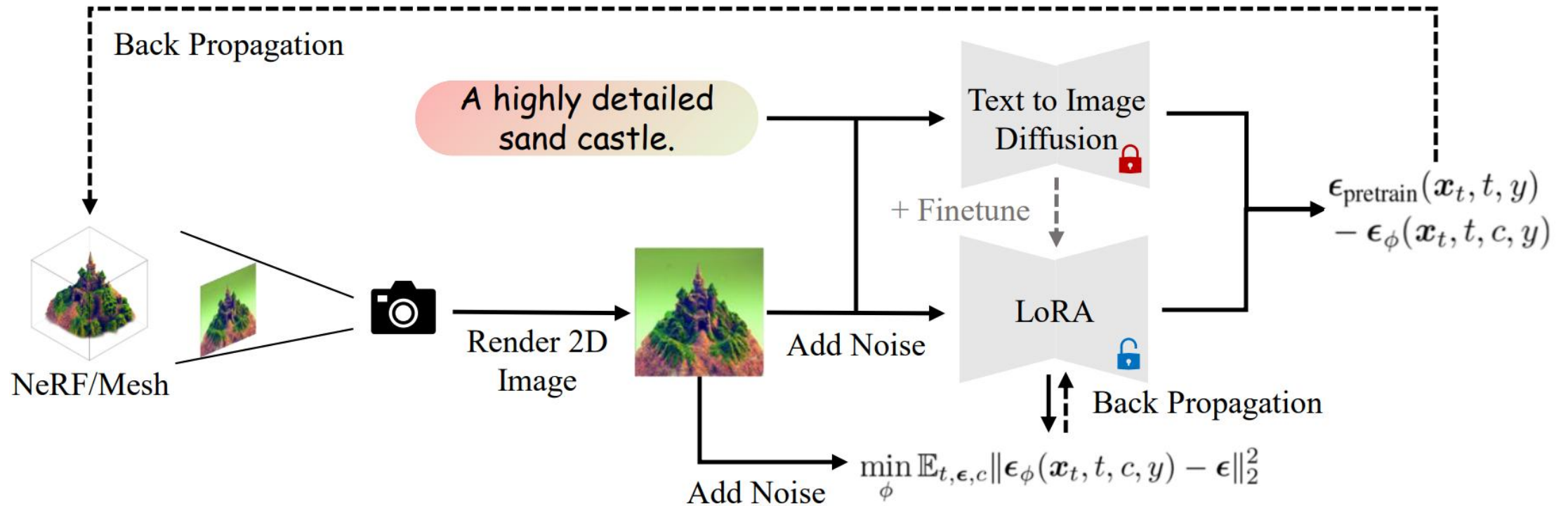
    Args:
        latents: Current optimized latents (B, 4, H, W)
        text_embeddings: Concatenated [uncond, cond] embeddings (2*B, seq_len, dim)
        guidance_scale: CFG scale

    Returns:
        loss: Scalar SDS loss
    """
    # TODO: Implement SDS Loss
    raise NotImplementedError("TODO: Implement SDS loss")
```

```
python main.py \
  --prompt "${PROMPT}" \
  --loss_type sds \
  --guidance_scale 25
```

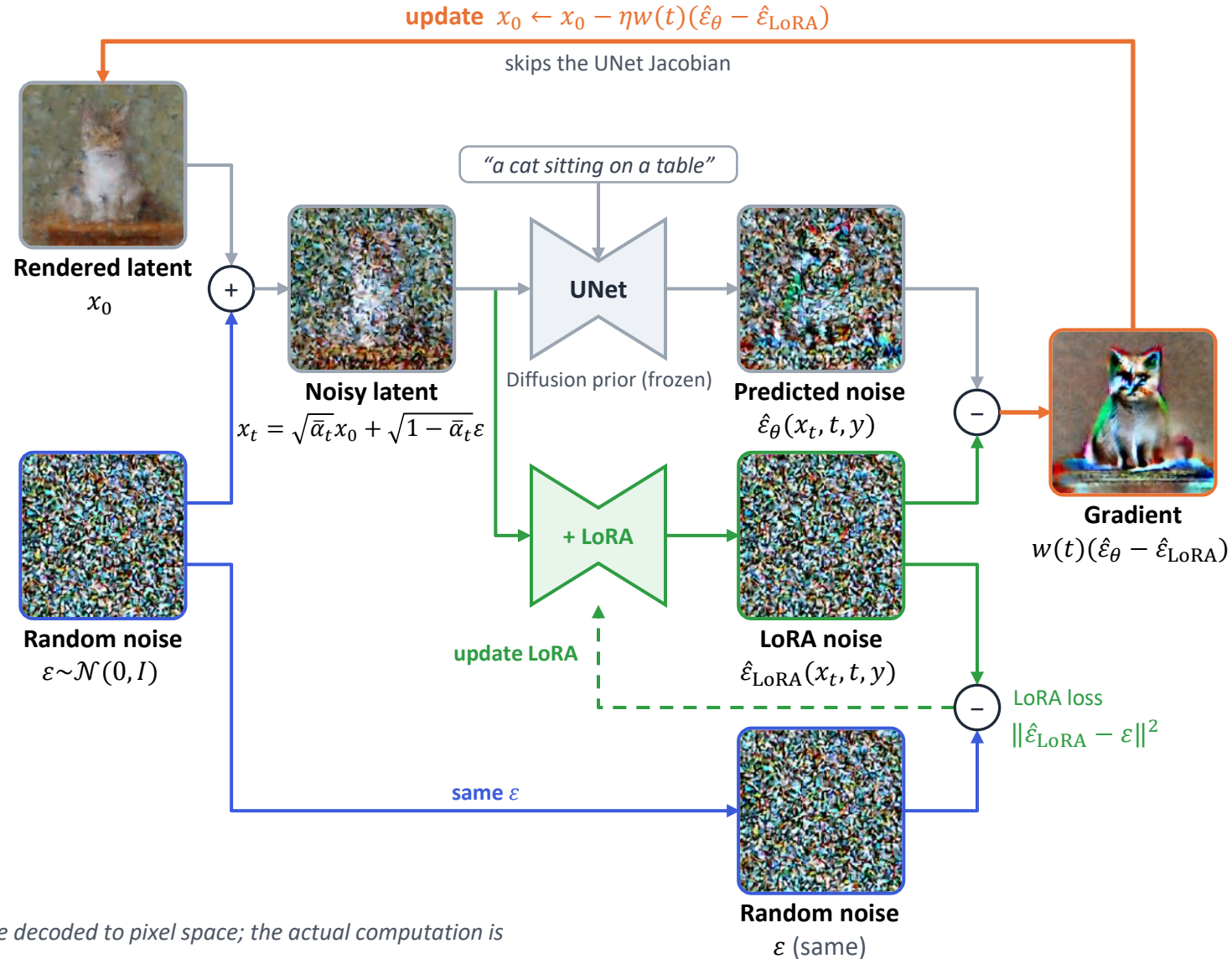
Task2 - Variational Score Distillation (VSD)

Use **LoRA** to mimic the noise prediction of a pretrained diffusion model.



ProlificDreamer, Wang *et al.*, NeurIPS 2023

Task2 - Variational Score Distillation (VSD)



Note: For clarity, all images shown here are decoded to pixel space; the actual computation is performed in latent space.

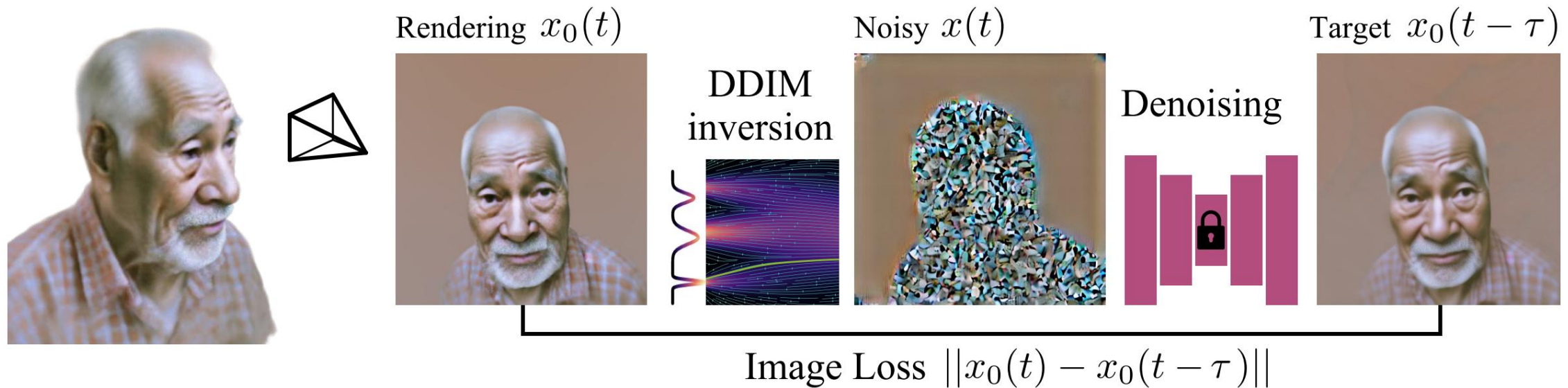
Task2 - Variational Score Distillation (VSD)

```
def get_vsd_loss(self, latents, text_embeddings, guidance_scale=7.5, lora_loss_weight=1.0):  
    """  
    Variational Score Distillation (VSD) loss.  
  
    Reference: ProlificDreamer (https://arxiv.org/abs/2305.16213)  
  
    Same as SDS, but the added noise is replaced by the prediction of a  
    LoRA-finetuned UNet that models the distribution of the current latents.  
    The LoRA is trained jointly with a standard denoising loss.  
  
    Args:  
        lora_loss_weight: Weight of the LoRA training loss  
  
    Returns:  
        loss: Scalar VSD loss + lora_loss_weight * LoRA loss  
    """  
    # TODO: Implement VSD loss  
    raise NotImplementedError("TODO: Implement VSD loss")
```

```
python main.py \  
    --prompt "${PROMPT}" \  
    --loss_type vsd \  
    --guidance_scale 7.5 \  
    --lora_lr 1e-4 \  
    --lora_loss_weight 1.0 \  
    --lora_rank 4
```

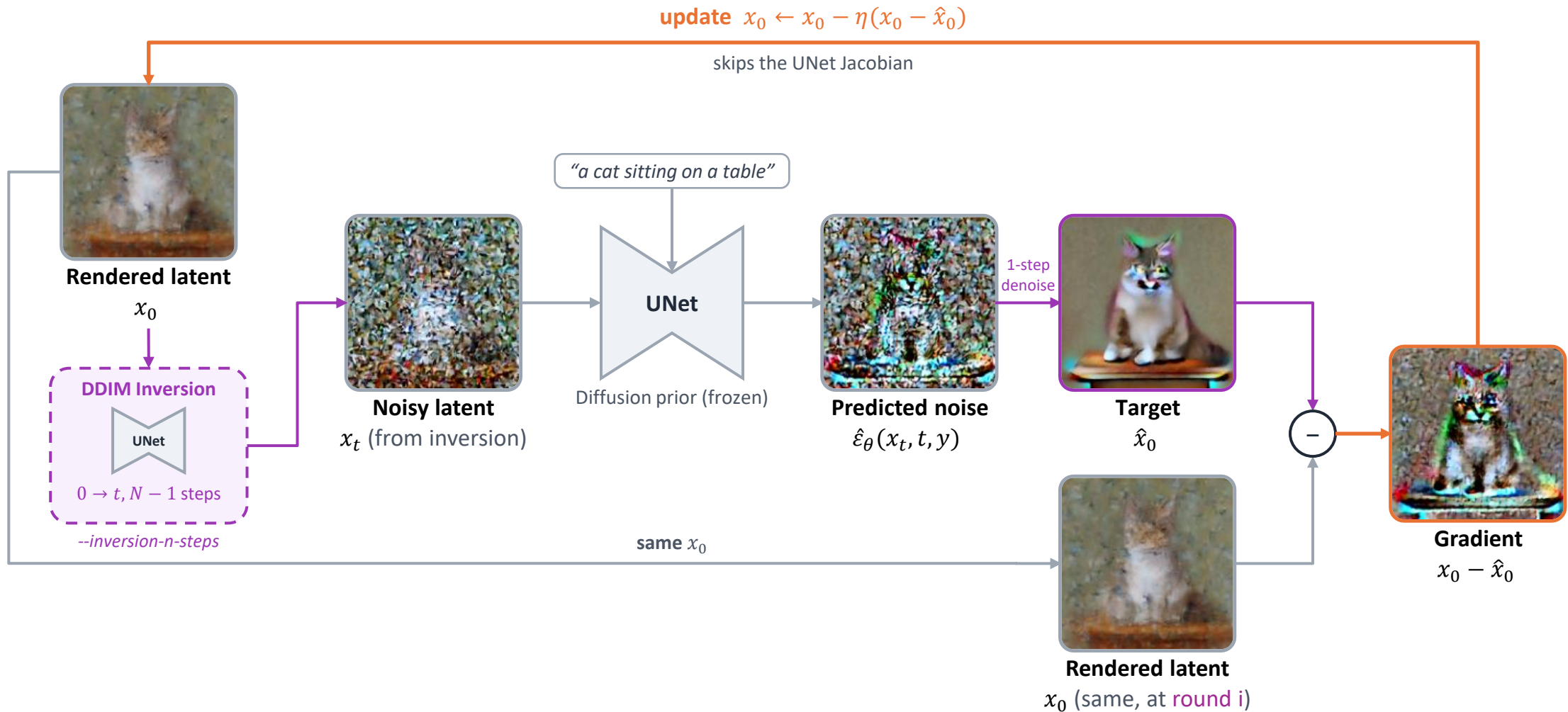
Task3 - Score Distillation via Inversion (SDI)

Perform **DDIM inversion** before computing score differences on x_0 .



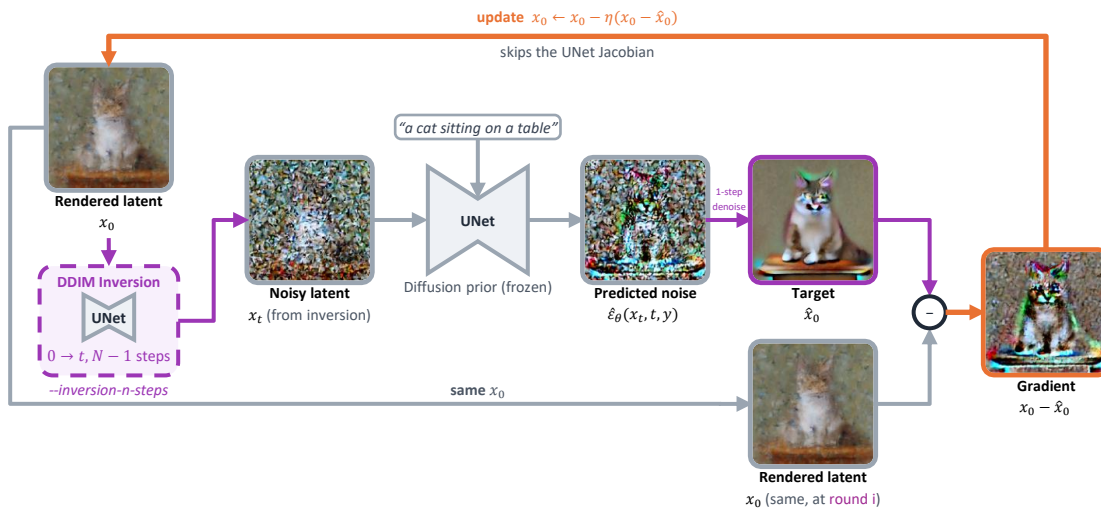
Score Distillation via Inversion, Lukoianov *et al.*, NeurIPS 2024

Task3 - Score Distillation via Inversion (SDI)



Note: For clarity, all images shown here are decoded to pixel space; the actual computation is performed in latent space.

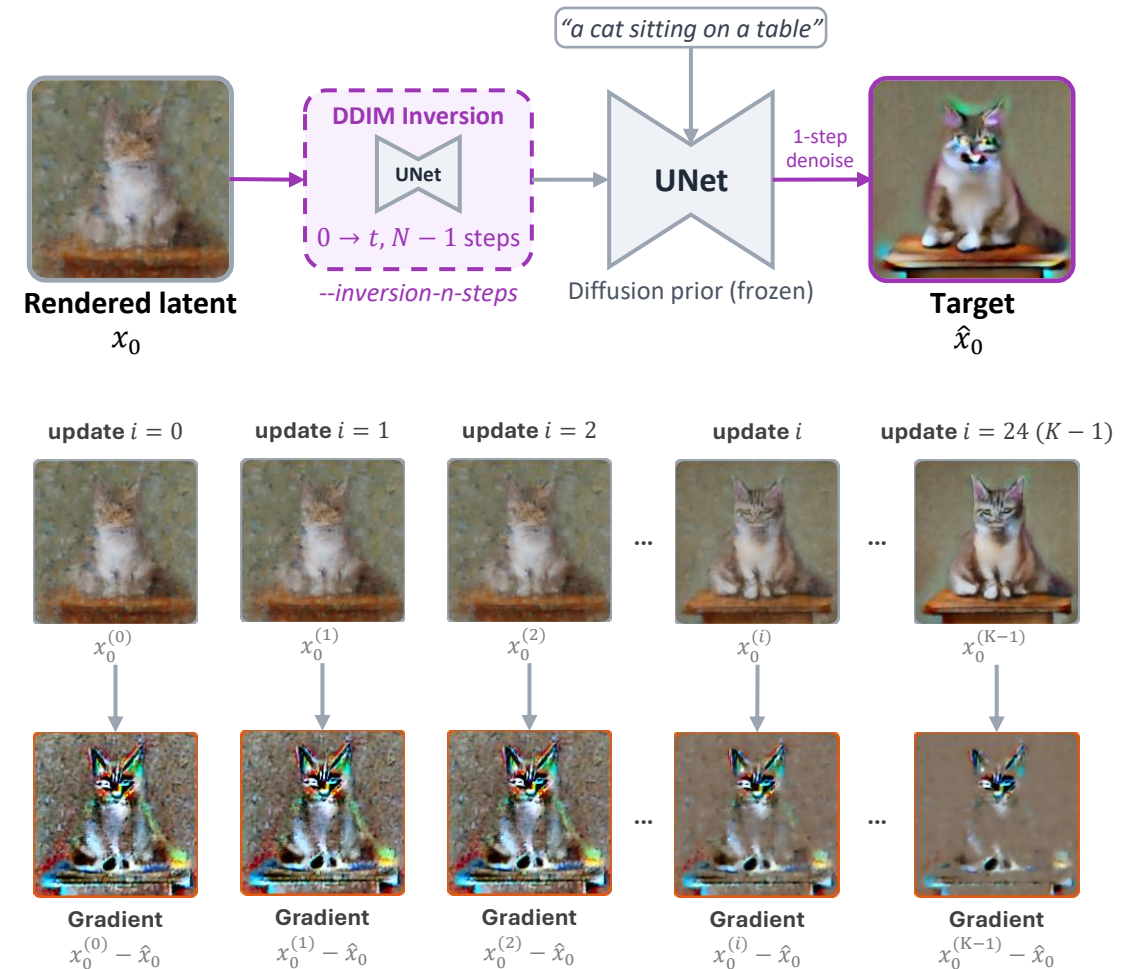
Task3 - Score Distillation via Inversion (SDI)



Method	Fwd / iter	Bwd / iter	UNet calls	Total fwd ¹
SDS	1	0	Frozen UNet	1,000
VSD	2	1	Frozen + LoRA UNet	2,000
SDI	N / K	0	$N-1$ inv. + 1 denoise $K = \text{update_interval}$	400

¹ 1,000 iterations; CFG ignored (CFG doubles every forward).

SDI: $N = \text{inversion_n_steps} = 10$. $K = \text{sdi_update_interval} = 25$



Task3 - Score Distillation via Inversion (SDI)

```
sd.py

@torch.no_grad()
def invert_noise(self, latents, target_t, text_embeddings, guidance_scale=-7.5, n_steps=10, eta=0.3):
    """
    DDIM inversion: x0 -> x_t.

    Reference: Score Distillation via Reparametrized DDIM (https://arxiv.org/abs/2405.15891)

    Runs the DDIM update in reverse, from t=0 up to `target_t`, to find noisy
    latents consistent with x0. Write the loop yourself: you must NOT call any
    "invert" / "ddim_invert" / "invert_step" utilities from diffusers or other libraries.

    Args:
        target_t: Target timestep to invert to
        guidance_scale: CFG scale (typically negative for inversion)
        n_steps: Number of inversion steps
        eta: Stochasticity level (0 = deterministic)

    Returns:
        latents_noisy: Inverted noisy latents x_t (B, 4, H, W)
    """
    # TODO: Implement DDIM inversion
    # Hint: You may assume eta=0 (deterministic inversion) and ignore `eta`;
    # no points will be deducted for this.
    raise NotImplementedError("TODO: Implement DDIM inversion")
```

```
sd.py

def get_sdi_loss(self, latents, text_embeddings, guidance_scale=7.5, current_iter=0, total_iters=500,
                 inversion_guidance_scale=-7.5, inversion_n_steps=10, inversion_eta=0.3, update_interval=25,
                 ):
    """
    Score Distillation via Inversion (SDI) loss.

    Reference: Score Distillation via Reparametrized DDIM (https://arxiv.org/abs/2405.15891)

    Instead of random noise as in SDS, SDI gets the noise by DDIM inversion of
    the current latents. The timestep is annealed from max_step to min_step, and
    the denoised target is cached and refreshed every `update_interval` iterations.

    Args:
        current_iter: Current training iteration
        total_iters: Total number of training iterations
        inversion_guidance_scale, inversion_n_steps, inversion_eta:
            Passed to `invert_noise` as guidance_scale, n_steps, eta
        update_interval: Refresh the cached target every N iterations

    Returns:
        loss: Scalar MSE loss between current latents and the cached target
    """
    # TODO: Implement SDI Loss
    raise NotImplementedError("TODO: Implement VSD loss")
```

```
python main.py --prompt "${PROMPT}" --loss_type sdi --lr 0.005 --steps 1000 --guidance_scale 7.5 \
--inversion_n_steps 10 --inversion_guidance_scale -7.5 \
--inversion_eta 0.3 --sdi_update_interval 25
```

Grading

Points are based on task completion and CLIP score.

Component	Points	Description
Task 1 – SDS	20	Correct implementation + results
Task 2 – VSD	20	Correct implementation + results
Task 3 – SDI	30	Correct implementation + results
Report	30	Clear analysis, comparisons, insights
Total	100	

CLIP Score	SDS/VSD (20 pts)	SDI (30 pts)
≥ 0.28	20 pts	30 pts
0.26–0.28	15 pts	25 pts
0.24–0.26	10 pts	20 pts
< 0.24	0	0

Evaluation

Run the evaluation script to compute CLIP scores automatically.

Use the unified evaluation script `eval.sh`:

```
bash eval.sh --sds
bash eval.sh --vsd --guidance 7.5 --lora-lr 1e-4 --lora-loss-weight 1.0 --lora-rank 4
bash eval.sh --sdi --guidance 7.5 --lr 0.005 --steps 1000 \
    --inversion-n-steps 10 --inversion-guidance-scale -7.5 \
    --inversion-eta 0.3 --sdi-update-interval 25
```

Each command will:

1. Generate results for all default prompts.
2. Run CLIP evaluation automatically and save final results to:

```
./outputs/{loss_type}/
├── <prompt>.png
└── eval.json
```

Evaluation

Sample images generated from the default evaluation prompts.



Report

Please include those sections and contents in your report:

1. Introduction

- [2 pts] Briefly describe and summarize the goal of this assignment.

2. Methodology

- [6 pts] Explain SDS, VSD, and SDI, covering both the concepts and your implementation.

3. Experiments & Results

- [6 pts] Provide both quantitative results (CLIP score) and qualitative results (output images).
- [6 pts] Conduct an ablation study on (1) SDS: guidance_scale; (2) SDI: inversion_n_steps; (3) SDI: sdi_update_interval
- [2 pts] Specify all hyperparameters you used for each task (task1-3).

4. Discussion

- [3 pts] Intuitively explain why the original loss function $\mathcal{L}_{\text{Diff}} = \|\hat{\epsilon}_{\theta}(x_t, y, t) - \epsilon_t\|^2$ does not produce realistic samples in practice (use at most one math equation & provide the CLIP score + visual results when applying the original $\mathcal{L}_{\text{Diff}}$).
- [3 pts] Intuitively analyze why SDI and VSD perform better than SDS in your words (without heavy math).

5. Conclusion

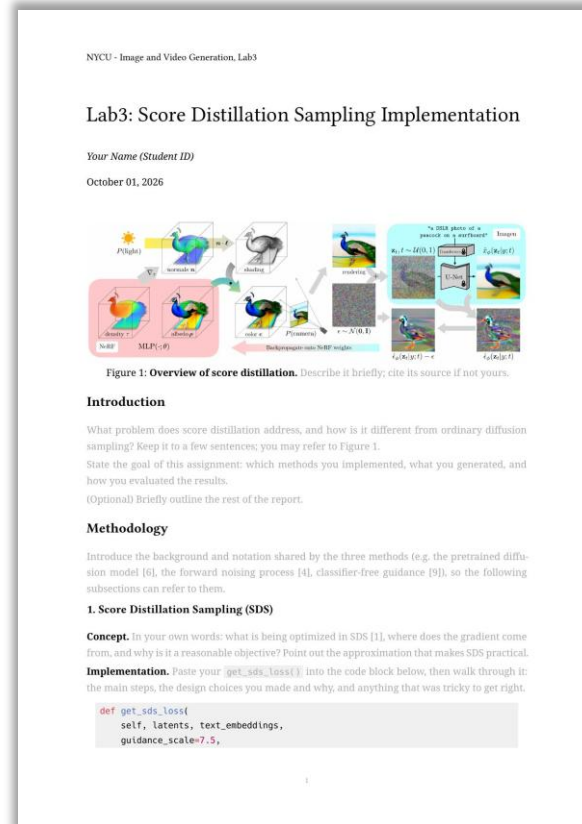
- [2 pts] Wrap up all the context in your report and give a short summary.

Report Templates

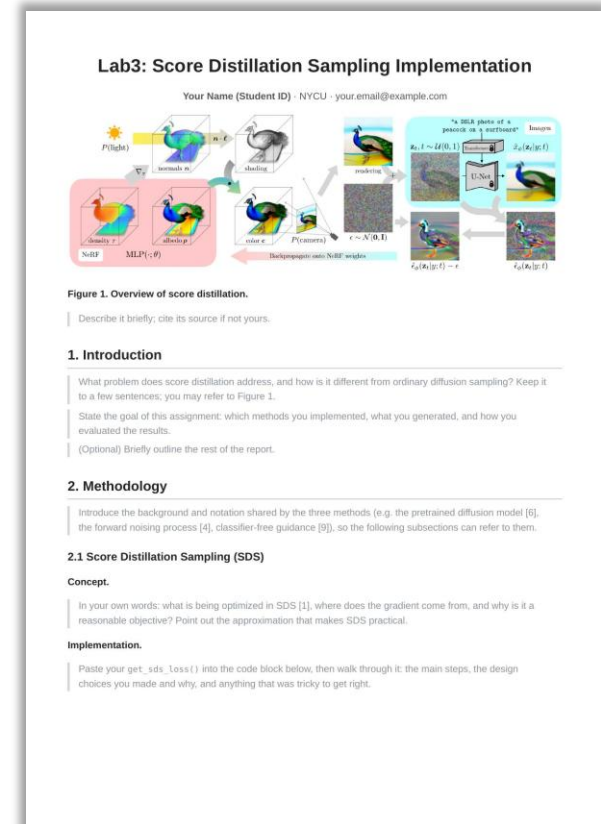
We recommend using one of these templates, but it is not required.



(a) LaTeX Template



(b) Typst Template



(c) Markdown Template

Submission

Create `{STUDENT_ID}_lab3.zip` with the following structure:

```
./{STUDENT_ID}_lab3.zip/  
├── guidance/sd.py  
├── outputs/{sds,vsd,sdi}/ + eval.json  
└── report.pdf
```

- Deadline: **2026/10/29 (Thu.) 23:59**
- Platform: New E3 System
- Late Policy: No Late Submissions Accepted

Reference

- [1] Denoising Diffusion Probabilistic Models (<https://arxiv.org/abs/2006.11239>)
- [2] Denoising Diffusion Implicit Models (<https://arxiv.org/abs/2010.02502>)
- [3] Null-text Inversion (<https://arxiv.org/abs/2211.09794>)
- [4] SDEdit: Guided Image Synthesis and Editing with SDE (<https://arxiv.org/abs/2108.01073>)
- [5] LoRA: Low-Rank Adaptation of Large Language Models (<https://arxiv.org/abs/2106.09685>)
- [6] DreamFusion: Text-to-3D using 2D Diffusion (<https://arxiv.org/abs/2209.14988>)
- [7] ProlificDreamer (<https://arxiv.org/abs/2305.16213>)
- [8] Score Distillation via Reparametrized DDIM (<https://arxiv.org/abs/2405.15891>)
- [9] KAIST CS492(D): Diffusion Models (<https://mhsung.github.io/kaist-cs492d-fall-2024/>)

Appendix

Lab3 - Distillation

Some Useful Functions

You don't need to re-implement the following functions yourself:

Mathematics: Obtain $\bar{\alpha}_t$ for each time step t

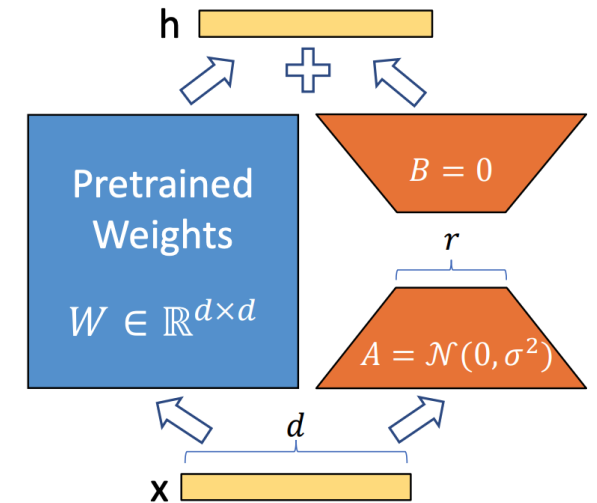
- `self.scheduler.alphas_cumprod()`
- `self.inverse_scheduler.alphas_cumprod()`

Noise Operation: Add noise ε or get noise prediction $\hat{\varepsilon}_\theta(x_t, t)$

- `self.scheduler.add_noise(x_0, noise, timestep)`
- `self.get_noise_preds(x_t, timestep, text_embed, cfg_scale)`

LoRA Fine-tuning: Determine whether to apply LoRA to the model

- `self.unet.enable_adapters()`
- `self.unet.disable_adapters()`



Surrogate Loss

A simple MSE loss sends any guidance gradient (grad) back to θ .

