Grooming Diffusion

Hands-on diffusion

NAVER AI Lab
김준호
https://github.com/taki0112

NAVER AI LAB

Intro | Contents

NAVER AI LAB

오전

Basic Diffusion

기본
: Scientist

Advanced Diffusion

심화
: Scientist, Engineer

Hands-on Diffusion

구현
: Scientist, Engineer

오후

오전

Basic Diffusion

기본
: Scientist

Advanced Diffusion

심화
: Scientist, Engineer

Hands-on Diffusion

구현
: Scientist, Engineer

오후

DDPM, DDIM

Diffusion

Clip score

Evaluation

2022

Test-to-Image

Stable diffusion

???

Recap | DDPM & DDIM

NAVER AI LAB

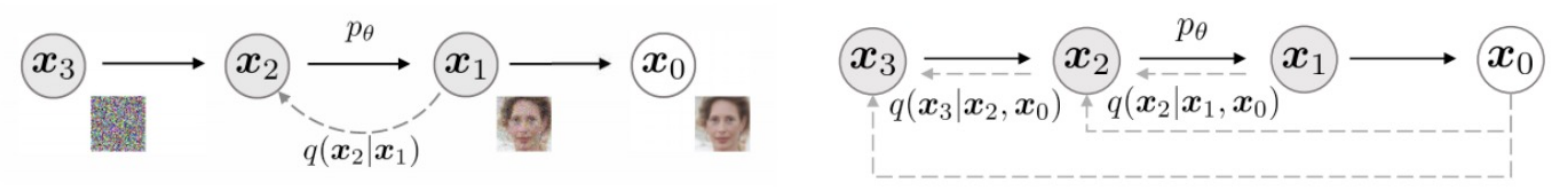


Figure 1: Graphical models for diffusion (left) and non-Markovian (right) inference models.

Denoising Diffusion Implicit Models

최근 Trend

$$x_{t-1} = \hat{\mu}_t(x_t) + \sigma_t \epsilon_t$$

$$x_{t-1} = \sqrt{\bar{\alpha}_{t-1}} P(f_t(x_t)) + D(f_t(x_t)) + \sigma_t \epsilon_t$$

$$* \; P(f_t(x_t)) = \frac{x_t - \sqrt{1 - \bar{\alpha}_t} f_t(x_t)}{\sqrt{\bar{\alpha}_t}}$$

$$* \; D(f_t(x_t)) = \sqrt{1 - \bar{\alpha}_{t-1} - \sigma_t^2} f_t(x_t)$$

DDPM

- $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, \; \epsilon \sim \mathcal{N}(0, \mathbf{I})$ `(Forward)`
  - $\beta_1 = 10^{-4}, \beta_T = 0.02$
  - $\alpha_t := 1 - \beta_t, \; \bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$
- $\epsilon - \epsilon_\theta(\mathbf{x}_t) = \epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon)$ `(Loss)`
  - ϵ_θ = prediction network
- $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t, t)\right) + \sqrt{\tilde{\beta}_t}\epsilon, \; \epsilon \sim \mathcal{N}(0, \mathbf{I})$ `(Reverse)`
  - $\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$
    - $\tilde{\beta}_t = \beta_t$로 해도 성능차이 없음

DDIM

- `In paper,` DDIM α = DDPM $\bar{\alpha}$
- $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ `(Forward)`
- $\epsilon - \epsilon_\theta(\mathbf{x}_t) = \epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon)$ `(Loss)`
  - ϵ_θ = prediction network
- $\mathbf{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}} \underbrace{\left(\frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\epsilon_\theta(\mathbf{x}_t)}{\sqrt{\bar{\alpha}_t}}\right)}_{\text{predicted } \mathbf{x}_0 = f_\theta(\mathbf{x}_t)} + \underbrace{\sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2} \cdot \epsilon_\theta(\mathbf{x}_t)}_{\text{direction pointing to } \mathbf{x}_t} + \underbrace{\sigma_t \epsilon}_{\text{noise}}$

  `(Reverse)`
  - deterministic when $\sigma_t = 0$ → consistency (DDIM)
  - stochastic when $\sigma_t = 1$ → inconsistency (DDPM)

Implement | DDPM & DDIM

NAVER AI LAB

DDPM

- $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon,\ \epsilon \sim \mathcal{N}(0, \mathbf{I})$ `(Forward)`
  - $\beta_1 = 10^{-4}, \beta_T = 0.02$
  - $\alpha_t := 1 - \beta_t,\ \bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$
- $\epsilon - \epsilon_\theta(\mathbf{x}_t) = \epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon)$ `(Loss)`
  - ϵ_θ = prediction network
- $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t, t)\right) + \sqrt{\tilde{\beta}_t}\epsilon,\ \epsilon \sim \mathcal{N}(0, \mathbf{I})$ `(Reverse)`
  - $\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$
    - $\tilde{\beta}_t = \beta_t$로 해도 성능차이 없음

DDIM

- `In paper,` DDIM α = DDPM $\bar{\alpha}$
- $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ `(Forward)`
- $\epsilon - \epsilon_\theta(\mathbf{x}_t) = \epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon)$ `(Loss)`
  - ϵ_θ = prediction network
- $\mathbf{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\underbrace{\left(\frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\epsilon_\theta(\mathbf{x}_t)}{\sqrt{\bar{\alpha}_t}}\right)}_{\text{predicted } \mathbf{x}_0 = f_\theta(\mathbf{x}_t)} + \underbrace{\sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\cdot\epsilon_\theta(\mathbf{x}_t)}_{\text{direction pointing to } \mathbf{x}_t} + \underbrace{\sigma_t\epsilon}_{\text{noise}}$

  `(Reverse)`
  - deterministic when $\sigma_t = 0$ → consistency (DDIM)
  - stochastic when $\sigma_t = 1$ → inconsistency (DDPM)

Step은 **총 5개**입니다. 나눠서 한번 구현해봅시다.

easy 2개, medium 2개, hard 1개

https://github.com/taki0112/diffusion-pytorch

NAVER AI LAB

Step 1. alpha값 (easy)

- $\beta_1 = 10^{-4}, \beta_T = 0.02$
- $\alpha_t := 1 - \beta_t, \bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$

```
"""
Step 1.

self.alpha = ?
self.alpha_hat = ?

"""
```

```
self.beta = self.prepare_noise_schedule(schedule, beta_start, beta_end).to(device)
```

Step 2. Sampling timestep (easy)

```python
def sample_timesteps(self, n):
    """
    Step 2.
    n개의 랜덤한 timestep을 샘플링 하세요. range = [1, self.noise_steps]

    :param n: int
    :return: [n, ] shape을 갖고있을것입니다.

    주의사항: timestep이니까, 값은 int형이어야 합니다.

    """
    return
```

Step 3. Forward process (medium)

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon,\ \epsilon \sim \mathcal{N}(0, \mathbf{I})$$ (Forward)

```python
def noise_images(self, x, t):
    """
    Step 3.
    forward process를 작성하세요.
    -> 이미지에 noise를 입히는 과정입니다.

    return은 노이즈를 입힌 이미지와, 입혔던 노이즈를 리턴하세요 !! 총 2개입니다.

    :param x: [n, 3, img_size, img_size]
    :param t: [n, ]
    :return: [n, 3, img_size, img_size], [n, 3, img_size, img_size]

    """
    return
```

Step 4. Training (medium)

$$\epsilon - \epsilon_\theta(\mathbf{x}_t) = \epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon)$$ (Loss)

- ϵ_θ = prediction network

```python
def train(args):
    setup_logging(args.run_name)
    device = args.device
    dataloader = get_data(args)
    model = UNet(device=device).to(device)
    optimizer = optim.AdamW(model.parameters(), lr=args.lr)
    diffusion = Diffusion(img_size=args.image_size, device=device)
    logger = SummaryWriter(os.path.join("logs", args.run_name))
    l = len(dataloader)

    for epoch in range(args.epochs):
        logging.info(f"Starting epoch {epoch}:")
        pbar = tqdm(dataloader)
        for i, images in enumerate(pbar):
            images = images.to(device)
            """
            Step 4.
            학습코드를 작성해보세요.
            다음 hint를 참고하여 작성하면됩니다.

            hint:
            (1) timestep을 샘플링 하세요.
            (2) 해당 timestep t에 대응되는 노이즈 입힌 이미지를 만드세요.
            (3) 모델에 넣어서, 노이즈를 predict 하세요.
            (4) 적절한 loss를 선택하세요. (L1 or L2)
            """

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            pbar.set_postfix(Loss=loss.item())
            logger.add_scalar("diffusion loss", loss.item(), global_step=epoch * l + i)
```

Implement | DDPM & DDIM

NAVER AI LAB

Step 5. Reverse process (hard)

```python
def sample(self, model, n):
    """
    Step 5. 마지막!
    reverse process를 완성하세요.

    :param model: Unet
    :param n: batch_size
    :return: x: [n, 3, img_size, img_size]
    """
    logging.info(f"Sampling {n} new images....")
    model.eval()
    with torch.no_grad():
        """
        (1) T스텝에서 부터 denoise하는것이기때문에, 가우시안 noise를 하나 만드세요.
        (2) T (self.noise_steps)부터 denoise하는 구문을 만드세요.
            hint: T, T-1, T-2, ... , 3, 2, 1 이런식으로 t가 나와야겠죠 ?
        (3) t에 해당하는 alpha_t, beta_t, alpha_hat_t, alpha_hat_(t-1), beta_tilde를 만드세요.

        (4) (1)의 noise와 (2)의 t를 모델에 넣어서, noise를 predict하세요.
        (5) predict한 noise를 가지고, ddpm과 ddim sampling를 작성하세요.

        """

    model.train()
    return torch.clamp(x, -1.0, 1.0)
```

DDPM

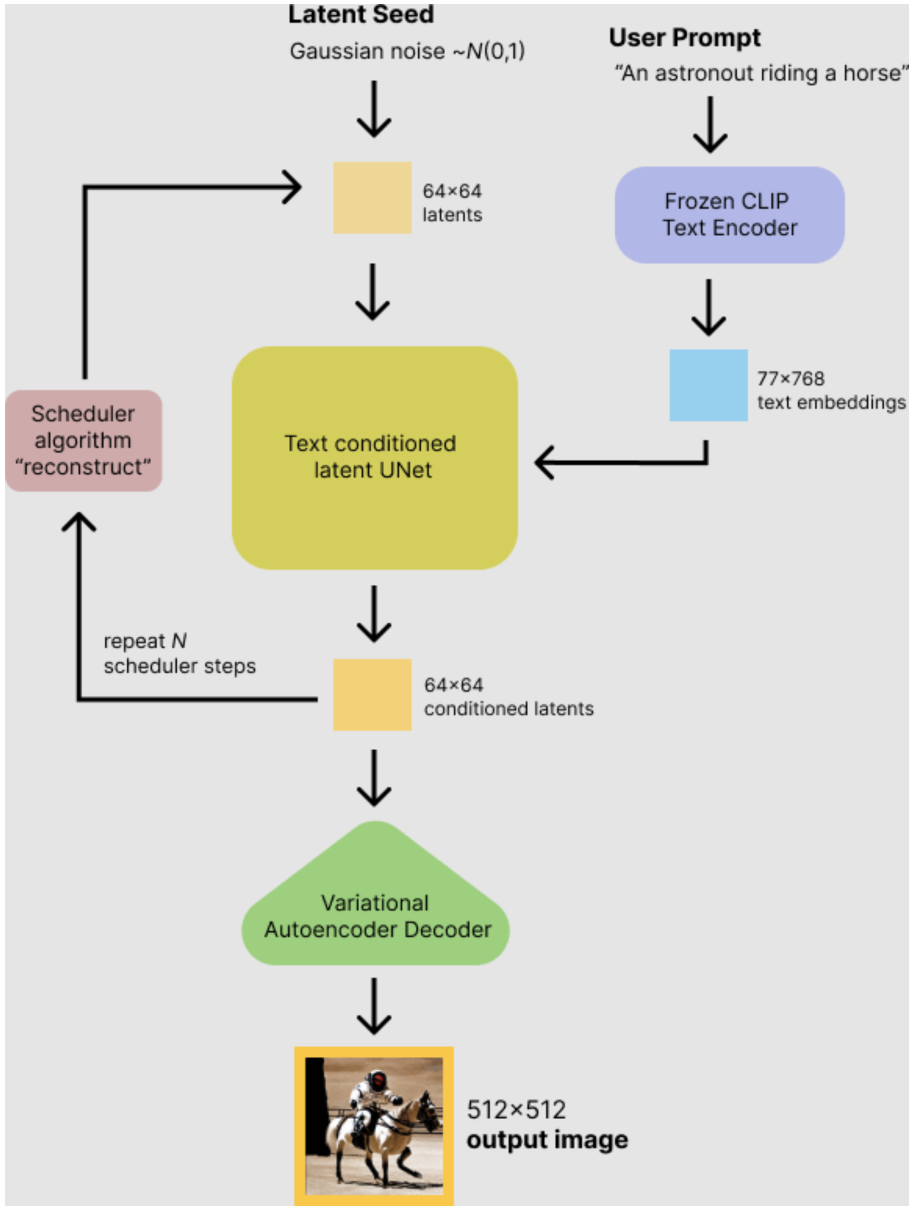
- $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t, t)\right) + \sqrt{\tilde{\beta}_t}\epsilon,\ \epsilon \sim \mathcal{N}(0, \mathbf{I})$ (Reverse)
  - $\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$
    - $\tilde{\beta}_t = \beta_t$로 해도 성능차이 없음

DDIM

- $\mathbf{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\underbrace{\left(\frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\epsilon_\theta(\mathbf{x}_t)}{\sqrt{\bar{\alpha}_t}}\right)}_{\text{predicted } \mathbf{x}_0 = f_\theta(\mathbf{x}_t)} + \underbrace{\sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\cdot\epsilon_\theta(\mathbf{x}_t)}_{\text{direction pointing to } \mathbf{x}_t} + \underbrace{\sigma_t\epsilon}_{\text{noise}}$

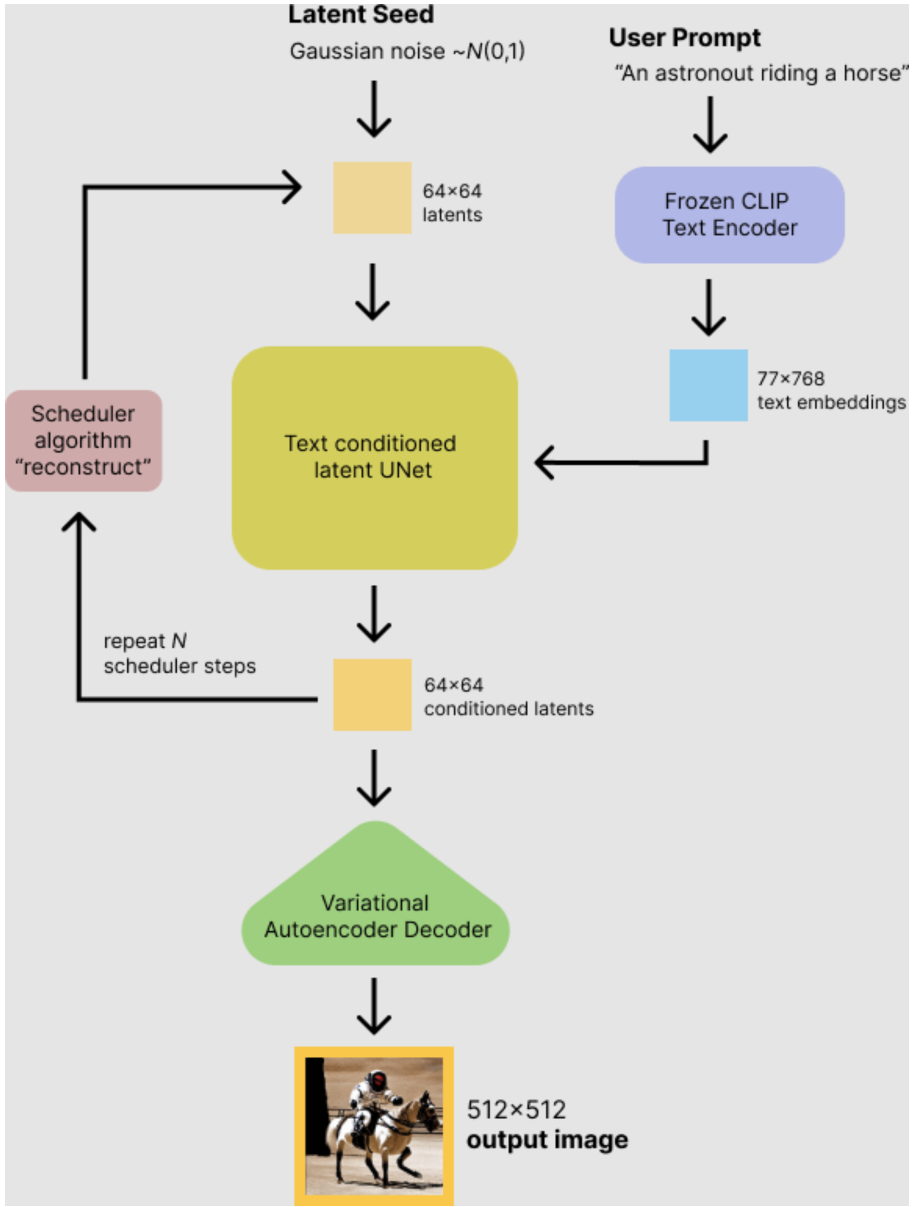
  (Reverse)
  - deterministic when $\sigma_t = 0$ → consistency (DDIM)
  - stochastic when $\sigma_t = 1$ → inconsistency (DDPM)

Practical use | Stable diffusion



https://github.com/taki0112/diffusion-pytorch

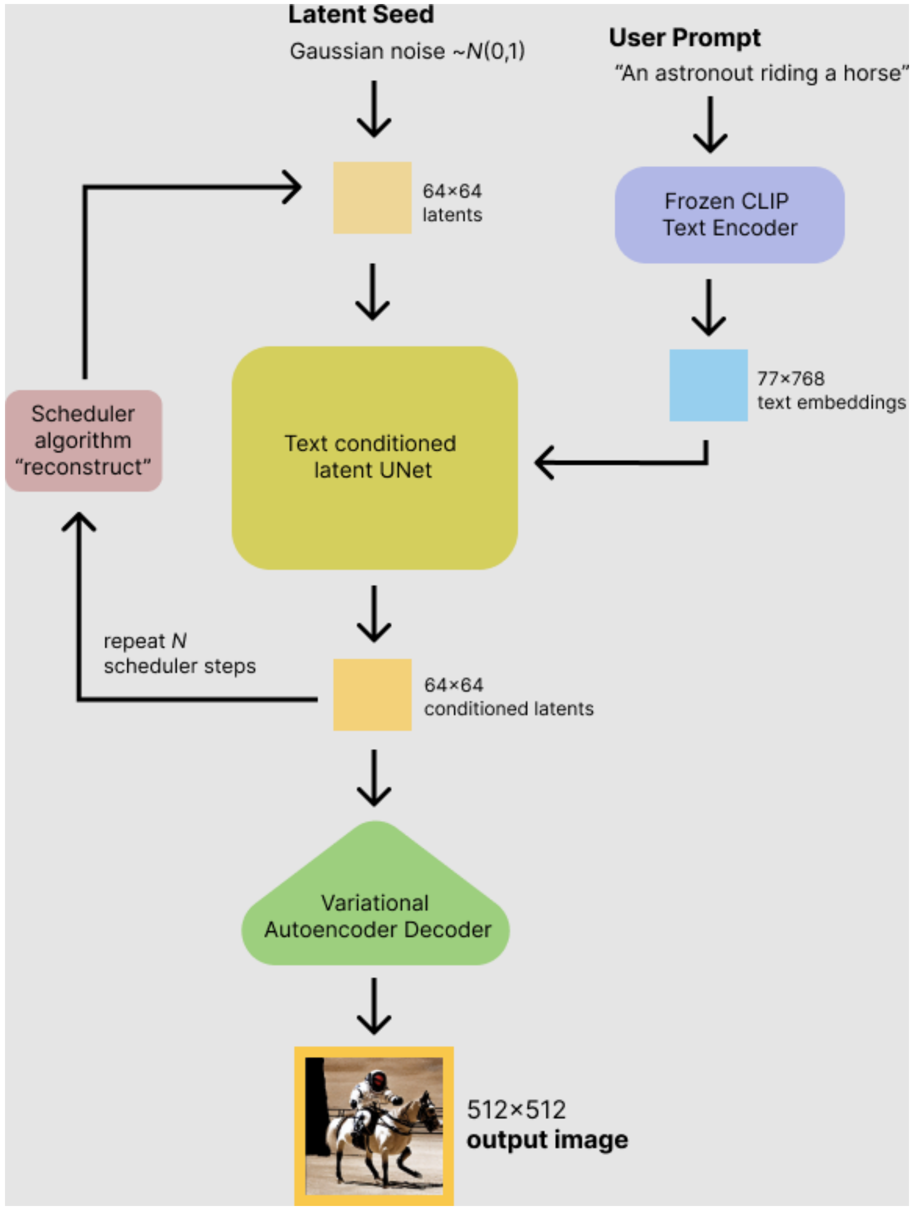
Practical use | Stable diffusion



https://github.com/taki0112/diffusion-pytorch

```python
import torch
from diffusers import StableDiffusionPipeline

# pip install diffusers

model_ckpt = "CompVis/stable-diffusion-v1-4"
device = "mps" # cuda, cpu, mps
weight_dtype = torch.float16

pipe = StableDiffusionPipeline.from_pretrained(model_ckpt, torch_dtype=weight_dtype)
pipe = pipe.to(device)

prompt = "a photograph of an astronaut riding a horse"
image = pipe(prompt).images[0]
image.save("simple_results.png")
```

Practical use | Stable diffusion (step-by-step)

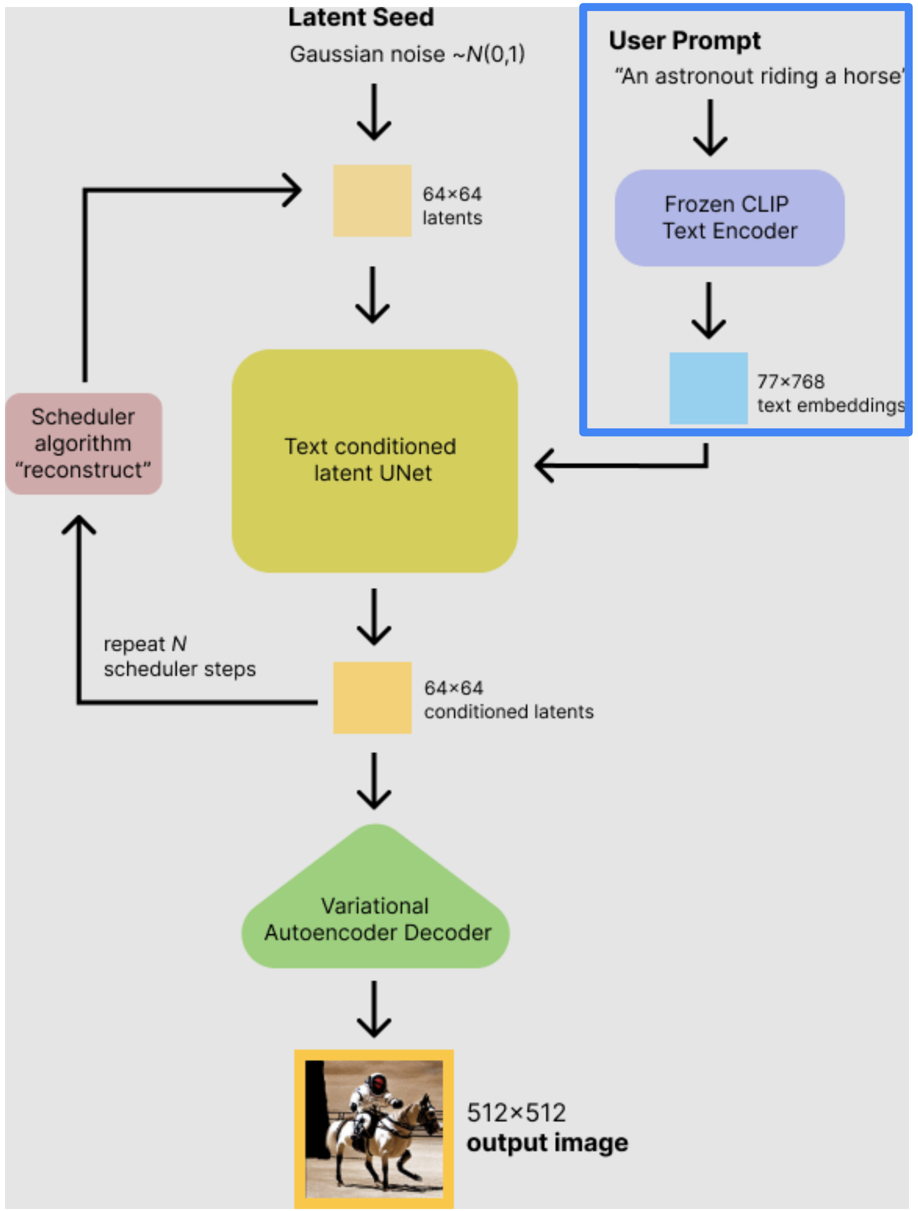


https://github.com/taki0112/diffusion-pytorch

```python
from PIL import Image
import torch
from transformers import CLIPTextModel, CLIPTokenizer
from diffusers import AutoencoderKL, UNet2DConditionModel, PNDMScheduler
from tqdm.auto import tqdm

# pip install diffusers

model_ckpt = "CompVis/stable-diffusion-v1-4"
torch_device = "cpu"


# init
vae = AutoencoderKL.from_pretrained(model_ckpt, subfolder="vae")
tokenizer = CLIPTokenizer.from_pretrained(model_ckpt, subfolder="tokenizer")
text_encoder = CLIPTextModel.from_pretrained(model_ckpt, subfolder="text_encoder")
unet = UNet2DConditionModel.from_pretrained(model_ckpt, subfolder="unet")
scheduler = PNDMScheduler.from_pretrained(model_ckpt, subfolder="scheduler")


# parameter
prompt = ["a photograph of an astronaut riding a horse"]
height = 512  # default height of Stable Diffusion
width = 512  # default width of Stable Diffusion
num_inference_steps = 25  # Number of denoising steps
guidance_scale = 7.5  # Scale for classifier-free guidance
generator = torch.manual_seed(0)  # Seed generator to create the inital latent noise
batch_size = len(prompt)
scheduler.set_timesteps(num_inference_steps)
print(scheduler.timesteps)
```

Practical use | Stable diffusion (step-by-step)

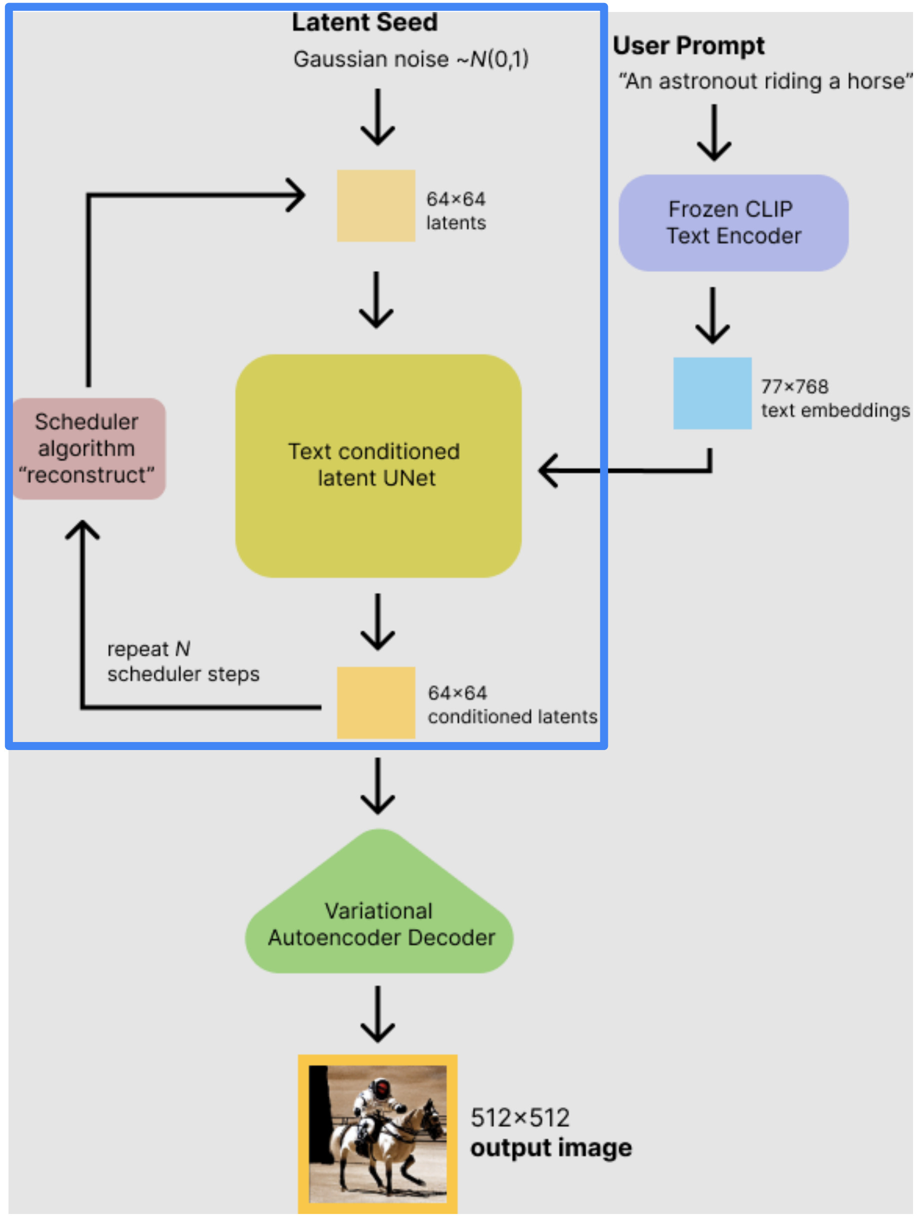
NAVER AI LAB




```python
"""
Step 1.
Make text embeddings
"""
text_input = tokenizer(
    prompt, padding="max_length", max_length=tokenizer.model_max_length, truncation=True, return_tensors="pt"
)

with torch.no_grad():
    text_embeddings = text_encoder(text_input.input_ids.to(torch_device))[0]

uncond_input = tokenizer([""] * batch_size, padding="max_length", max_length=tokenizer.model_max_length, return_tensors="pt")
uncond_embeddings = text_encoder(uncond_input.input_ids.to(torch_device))[0]

text_embeddings = torch.cat([uncond_embeddings, text_embeddings])
```

https://github.com/taki0112/diffusion-pytorch

Practical use | Stable diffusion (step-by-step)

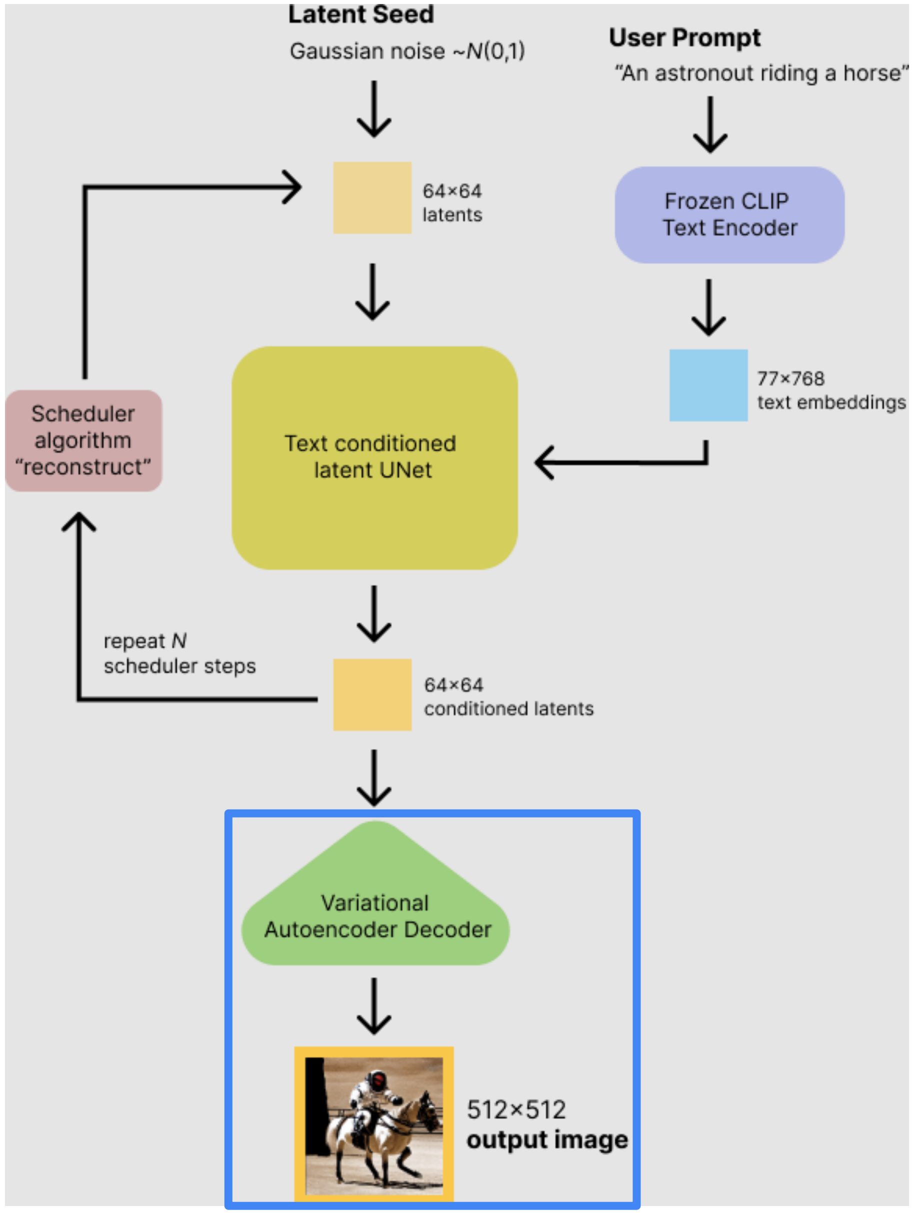


https://github.com/taki0112/diffusion-pytorch

```python
"""
Step 2.
Reverse process
"""
# Create random noise
latents = torch.randn(
    (batch_size, unet.config.in_channels, height // 8, width // 8),
    generator=generator,
)
latents = latents.to(torch_device)
latents = latents * scheduler.init_noise_sigma # PNDMS = 1

for t in tqdm(scheduler.timesteps):
    # expand the latents if we are doing classifier-free guidance to avoid doing two forward passes.
    latent_model_input = torch.cat([latents] * 2)

    latent_model_input = scheduler.scale_model_input(latent_model_input, timestep=t)

    # predict the noise residual
    with torch.no_grad():
        noise_pred = unet(latent_model_input, t, encoder_hidden_states=text_embeddings).sample

    # perform guidance
    noise_pred_uncond, noise_pred_text = noise_pred.chunk(2)
    noise_pred = noise_pred_uncond + guidance_scale * (noise_pred_text - noise_pred_uncond)

    # compute the previous noisy sample x_t -> x_t-1
    latents = scheduler.step(noise_pred, t, latents).prev_sample
```

Practical use | Stable diffusion (step-by-step)

NAVER AI LAB




```python
"""
Step 3.
Image decoding
"""
latents = 1 / 0.18215 * latents
with torch.no_grad():
    image = vae.decode(latents).sample

image = (image / 2 + 0.5).clamp(0, 1)
image = image.detach().cpu().permute(0, 2, 3, 1).numpy()
images = (image * 255).round().astype("uint8")
pil_images = [Image.fromarray(image) for image in images]
pil_images[0].save("main_results.png")
```

https://github.com/taki0112/diffusion-pytorch

Practical use | Evaluation (Clip Score)

NAVER AI LAB

```python
from transformers import CLIPTokenizer, CLIPTextModel, CLIPVisionModel, CLIPModel
import torch
from torchvision import transforms
import torch.nn.functional as F

# input
image = torch.randint(255, (2, 3, 224, 224))
text = ["a photo of a cat", "a photo of a cat"]
version = 'openai/clip-vit-large-patch14'


"""
Step 1. Model Init    # CLIPModel만 불러도 됩니다.
"""
tokenizer = CLIPTokenizer.from_pretrained(version)
clip_text_encoder = CLIPTextModel.from_pretrained(version)
clip_image_encoder = CLIPVisionModel.from_pretrained(version)
clip_model = CLIPModel.from_pretrained(version)
```

https://github.com/taki0112/diffusion-pytorch

Practical use | Evaluation (Clip Score)


```
"""
Step 2. Text
"""
batch_encoding = tokenizer(text, truncation=True, max_length=77, padding="max_length", return_tensors="pt")
# [input_ids, attention_mask] -> 둘다 [bs,77]의 shape을 갖고있습니다.
# input_ids는 주어진 텍스트를 토크나이즈한것이고, mask는 어디까지만이 유효한 token인지 알려줍니다. 1=유효, 0=의미없음
```

* input_ids: [49406,   320, 1125,   539,   320, 2368, 49407, 49407, ... , 49407]
* attention_mask: [1, 1, 1, 1, 1, 1, 1, 0, 0, ... , 0]

Practical use | Evaluation (Clip Score)

NAVER AI LAB

```python
"""
Step 2. Text
"""
batch_encoding = tokenizer(text, truncation=True, max_length=77, padding="max_length", return_tensors="pt")
# [input_ids, attention_mask] -> 둘다 [bs,77]의 shape을 갖고있습니다.
# input_ids는 주어진 텍스트를 토크나이즈한것이고, mask는 어디까지만이 유효한 token인지 알려줍니다. 1=유효, 0=의미없음

text_token = batch_encoding["input_ids"]
t_embed = clip_text_encoder(text_token) # 이것은 clip_model.text_model(text_token)과 같다.
# [last_hidden_state, pooler_output] -> [bs, 77, 768], [bs, 768]
# last_hidden_state = word embedding
# pooler_output = sentence embedding

text_feature = clip_model.get_text_features(text_token)
# pooler_output(sentence embedding) 에 Linear를 태운것
# [bs, 768]
```

t_embed, text_feature

필요에 따라 둘중에 하나 선택한다.

e.g.) clip score는 text_feature

* input_ids: [49406, 320, 1125, 539, 320, 2368, 49407, 49407, ... , 49407]
* attention_mask: [1, 1, 1, 1, 1, 1, 1, 0, 0, ... , 0]

Practical use | Evaluation (Clip Score)

NAVER AI LAB

```
"""
Step 3. Image
"""
image = clip_image_process(image)

i_embed = clip_image_encoder(image) # 이것은 clip_model.vision_model(image)과 같다.
# [last_hidden_state, pooler_output] -> [bs, 256, 1024], [bs, 1024]

image_feature = clip_model.get_image_features(image)
# pooler_output에 Linear을 태운것
```

i_embed, image_feature

필요에 따라 둘중에 하나 선택한다.

e.g.) clip score는 image_feature

https://github.com/taki0112/diffusion-pytorch

Practical use | Evaluation (Clip Score)

NAVER AI LAB

```python
def clip_image_process(x):
    def denormalize(x):
        # [-1, 1] ~ [0, 255]
        x = ((x + 1) / 2 * 255).clamp(0, 255).to(torch.uint8)

        return x

    def resize(x):
        x = transforms.Resize(size=[224, 224], interpolation=transforms.InterpolationMode.BICUBIC, antialias=True)(x)
        return x

    def zero_to_one(x):
        x = x.float() / 255.0
        return x

    def norm_mean_std(x):
        mean = [0.48145466, 0.4578275, 0.40821073]
        std = [0.26862954, 0.26130258, 0.27577711]
        x = transforms.Normalize(mean=mean, std=std, inplace=True)(x)
        return x

    # 만약 x가 [-1, 1] 이면, denorm을 해줍니다.
    # x = denormalize(x)
    x = resize(x)
    x = zero_to_one(x)
    x = norm_mean_std(x)

    return x
```

https://github.com/taki0112/diffusion-pytorch

Practical use | Evaluation (Clip Score)

NAVER AI LAB

```python
def clip_score(img_features, txt_features):
    img_features = img_features / img_features.norm(p=2, dim=-1, keepdim=True)
    txt_features = txt_features / txt_features.norm(p=2, dim=-1, keepdim=True)

    # score = 100 * (img_features * txt_features).sum(axis=-1)
    # score = torch.mean(score)

    # 위와 같다.
    score = F.cosine_similarity(img_features, txt_features).mean()
    return score
```

```python
def contrastive_loss(logits, dim) :
    neg_ce = torch.diag(nn.functional.log_softmax(logits, dim=dim))
    return -neg_ce.mean()

def clip_contra_loss(img_features, txt_features, logit_scale):
    img_features = img_features / img_features.norm(p=2, dim=-1, keepdim=True)
    txt_features = txt_features / txt_features.norm(p=2, dim=-1, keepdim=True)

    # cosine similarity as logits
    logit_scale = logit_scale.exp()
    similarity = torch.matmul(txt_features, img_features.t()) * logit_scale

    caption_loss = contrastive_loss(similarity, dim=0)
    image_loss = contrastive_loss(similarity, dim=1)

    return (caption_loss + image_loss) / 2.0 # minimize
```

전체적으로 잘 맞니 ?

Pair끼리 무조건 맞아라

https://github.com/taki0112/diffusion-pytorch

Project

Stable diffusion (Dreambooth)

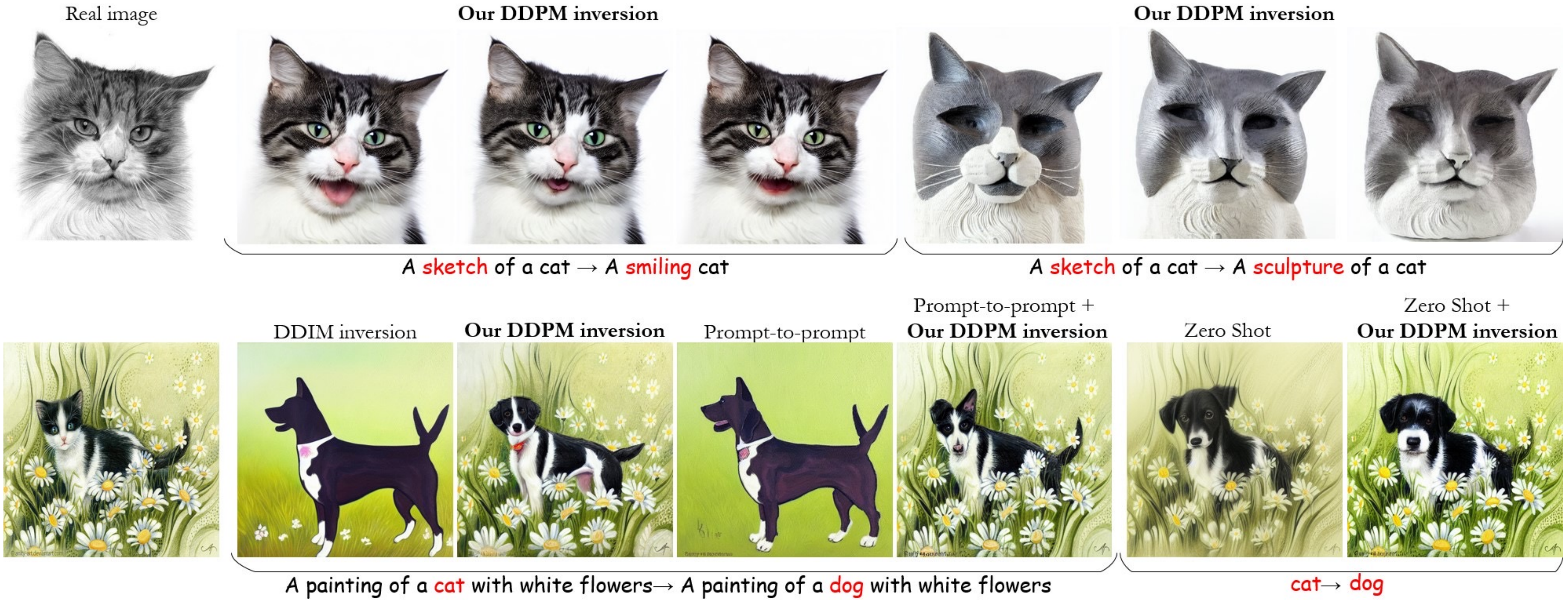
NAVER AI LAB

https://github.com/huggingface/diffusers/tree/main/examples/dreambooth

Project

DDPM Inversion, Prompt-to-Prompt

NAVER AI LAB



https://github.com/inbarhub/DDPM_inversion

Thank you !

jhkim.ai@navercorp.com

Junho Kim

https://github.com/taki0112