Understand Stable Diffusion from Code

Prompt: Understand Stable Diffusion from code, cyberpunk theme, best quality, high resolution, concept art

2. Masamune Ishihara

Computer Engineering Undergrad at University of California, Santa Cruz
I'm interested in AI/ML and GIS.

Likes:

- Tea
- Tennis

masaishi

@masaishi2001

masamune-ishihara

masaishi

Purpose

Introduce image generation process with code

About

In writing this slide, I realized many things that I did not understand. I may have explained things incorrectly, so I would appreciate it if you could tell me if I am unclear or if I have made any mistakes by clicking on the link below.

understand-stable-diffusion-slidev

Issues: Please let me know if you find any mistakes.

Discussions: Please let me know if you have questions.

Pull Requests: Please let me know if you have any improvements.

About

The concept of this slide is to introduce the flow of image generation with code, so basically all the code in this slide can be actually run.

Repository list

understand-stable-diffusion-slidev: Repository of this slide.

understand-stable-diffusion-slidev-notebooks: Notebooks for generating sample images and gifs.

parediffusers: Simple library for generating images without using huggingface/diffusers.

Table of Contents

4. Flow of Image Generation

Prompt: Stable Diffusion, watercolor painting, best quality, high resolution

What is Stable Diffusion?

- An image generation model based on the Latent Diffusion Model (LDM) developed by Stability AI.
- It can be used for Text-to-Image, Image-to-Image.
- It can be easily moved by using Diffusers.
- https://arxiv.org/abs/2112.10752

What is Diffusers?

- Library for Diffusion Models developed by Hugging Face🤗.
- Easy to run many image generation models.
- https://github.com/huggingface/diffusers

Diffusers

Open in Colab

Install the Diffusers library:

```
!pip install transformers diffusers accelerate
```

Generate an image from text:

```
import torch
from diffusers import StableDiffusionPipeline

pipe = StableDiffusionPipeline.from_pretrained
  "stabilityai/stable-diffusion-2",
  dtype=torch.float16,
).to(device=torch.device("cuda"))
prompt = "painting depicting the sea, sunrise,

image = pipe(prompt, width=512, height=512).im
display(image)
```

Diffusers are highly flexible,
but understanding the code is difficult.

diffusers/.../pipeline_stable_diffusion.py

```python
# Copyright 2024 The HuggingFace Team. All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
import inspect
from typing import Any, Callable, Dict, List, Optional, Union

import torch
from packaging import version
from transformers import CLIPImageProcessor, CLIPTextModel, CLIPTokenizer, CLIPVisionModelWithProjection
```

parediffusers/.../pipeline.py

```python
import torch
from torchvision.transforms import ToPILImage
from transformers import CLIPTokenizer, CLIPTextModel
from .scheduler import PareDDIMScheduler
from .unet import PareUNet2DConditionModel
from .vae import PareAutoencoderKL


class PareDiffusionPipeline:
    def __init__(
        self,
        tokenizer,
        text_encoder,
        scheduler,
        unet,
        vae,
        device=torch.device("cuda"),
        dtype=torch.float16,
    ):
```

PareDiffusers

Open in Colab

Install the PareDiffusers library:

```
!pip install parediffusers
```

Generate an image from text:

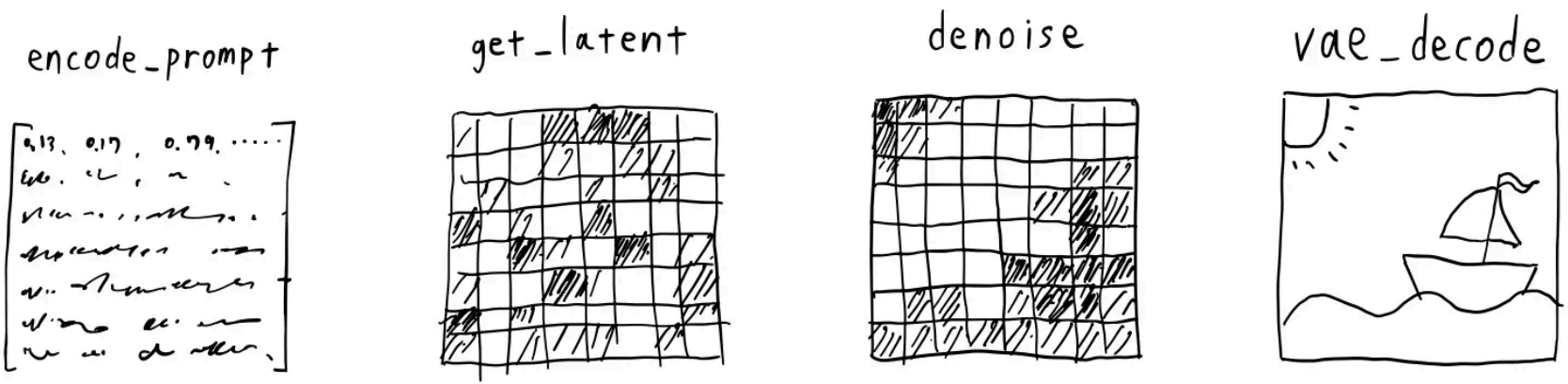
```
import torch
from parediffusers import PareDiffusionPipelin

pipe = PareDiffusionPipeline.from_pretrained(
  "stabilityai/stable-diffusion-2",
  device=torch.device("cuda"),
  dtype=torch.float16,
)
prompt = "painting depicting the sea, sunrise,

image = pipe(prompt, width=512, height=512)
display(image)
```

How is image generation performed?

pipeline.py#L117-L135

1. `encode_prompt` : Convert prompt to embedding.
2. `get_latent` : Create random Latent.
3. `denoise` : Denosing by using Scheduler and UNet.
4. `vae_decode` : Decode to pixel space by VAE.

1. `encode_prompt` : Convert prompt to embedding.
2. `get_latent` : Create random Latent.
3. `denoise` : Denosing by using Scheduler and UNet.
4. `vae_decode` : Decode to pixel space by VAE.



A Briefly Theory

What is Latent Diffusion Model (LDM)?

Models Which Run DDPM on Latent Space

What is Denoising Diffusion Probabilistic Model (DDPM)?

Adding noise to the image and restoring the original image from it.

It is used for audio and other data in general, but this slide will discuss images.

- Diffusion process is used to preprocess training data. Stochastic process（Markov chain)
- Reverse process is used to recover the original data from the noisy data.

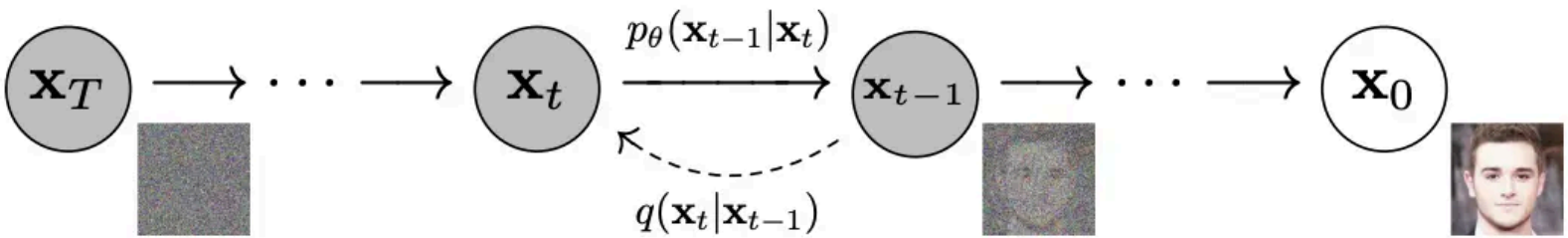


Figure 2: The directed graphical model considered in this work.

Jonathan Ho, Ajay Jain, Pieter Abbeel: “Denoising Diffusion Probabilistic Models”, 2020; arXiv:2006.11239.

It is interesting that it is called a Diffusion Model even if diffusion is not NN just processing.

What is Latent Diffusion Model (LDM)?

Models Which Run DDPM on Latent Space

Difference of Loss Functions

$$L_{DM} := \mathbb{E}_{x,\epsilon\sim\mathcal{N}(0,1),t}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|_2^2\right].$$

$$L_{LDM} := \mathbb{E}_{\mathcal{E}(x),\epsilon\sim\mathcal{N}(0,1),t}\left[\|\epsilon - \epsilon_\theta(z_t, t)\|_2^2\right].$$

Latent Diffusion Model (LDM)

$$L_{LDM} := \mathbb{E}_{\mathcal{E}(x),\epsilon\sim\mathcal{N}(0,1),t}\left[\|\epsilon - \epsilon_\theta(z_t, t)\|_2^2\right].$$

Latent Diffusion Model (LDM) with Conditioning

$$L_{LDM} := \mathbb{E}_{\mathcal{E}(x),y,\epsilon\sim\mathcal{N}(0,1),t}\left[\|\epsilon - \epsilon_\theta(z_t, t, \tau_\theta(y))\|_2^2\right],$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) \cdot V$$

$$Q = W_Q^{(i)} \cdot \varphi_i(z_t),\ K = W_K^{(i)} \cdot \tau_\theta(y),\ V = W_V^{(i)} \cdot \tau_\theta(y).$$

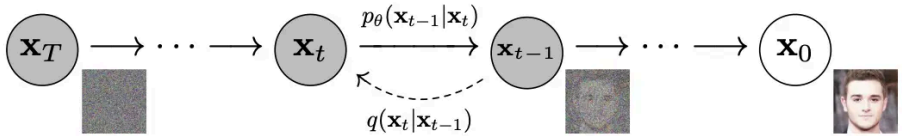
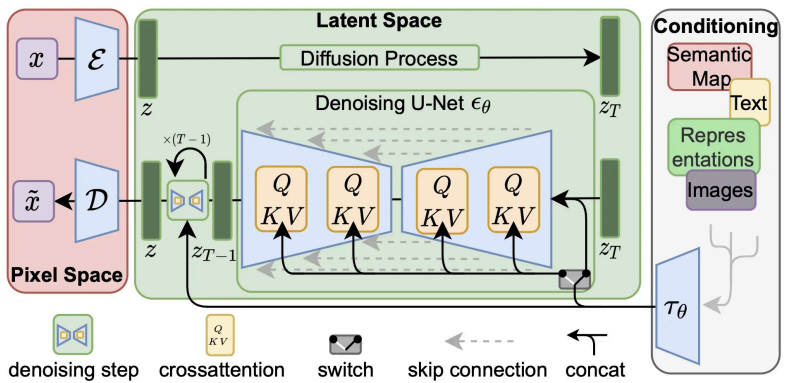


Figure 2: The directed graphical model considered in this work.

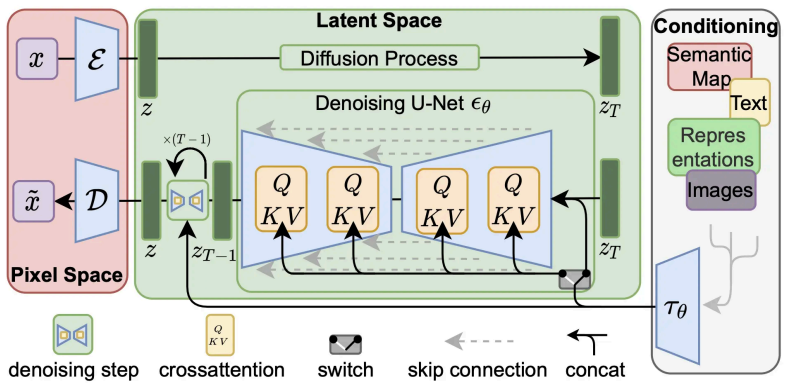
Jonathan Ho, Ajay Jain, Pieter Abbeel: “Denoising Diffusion Probabilistic Models”, 2020; arXiv:2006.11239.



Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, Björn Ommer: “High-Resolution Image Synthesis with Latent Diffusion Models”, 2021; arXiv:2112.10752.

```
131  prompt_embeds = self.encode_prompt(prompt)
132  latents = self.get_latent(width, height).unsqueeze(dim=0)
133  latents = self.denoise(latents, prompt_embeds, num_inference_steps, guidance_scale)
134  image = self.vae_decode(latents)
```

src/parediffusers/pipeline.py delivered with ❤️ by emgithub — view raw



Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, Björn Ommer: “High-Resolution Image Synthesis with Latent Diffusion Models”, 2021; arXiv:2112.10752.

What is Latent Space?

Features of the Input Image are Extracted

The flow of image generation in 4 steps

Step 1: Convert prompt to embedding.
Step 2: Create random Latent.
Step 3: Denosing by using Scheduler and UNet.
Step 4: Decode to pixel space by VAE.

The flow of image generation in 4 steps

Step 1: encode_prompt
Step 2: get_latent
Step 3: denoise
Step 4: vae_decode

Step 1: encode_prompt

Prompt: Pipeline, cyberpunk theme, best quality, high resolution, concept art

Step 1: encode_prompt

Convert prompt to embedding.

Step 1: encode_prompt

Convert prompts into a form that is easy for the model to handle.

Necessities

- CLIPTokenizer
- CLIPTextModel

From huggingface/transformers

Step 1: encode_prompt

Calling another function twice within encode_prompt

pipeline.py#L41-L48

```
41  def encode_prompt(self, prompt: str):
42      """
43      Encode the text prompt into embeddings using the t
44      """
45      prompt_embeds = self.get_embes(prompt, self.tokeni
46      negative_prompt_embeds = self.get_embes([''], prom
47      prompt_embeds = torch.cat([negative_prompt_embeds,
48      return prompt_embeds
```

Step 1: encode_prompt

Where are Necessities used?

pipeline.py#L41-L57

```
41  def encode_prompt(self, prompt: str):
42      """
43      Encode the text prompt into embeddings using the t
44      """
45      prompt_embeds = self.get_embes(prompt, self.tokeni
46      negative_prompt_embeds = self.get_embes([''], prom
47      prompt_embeds = torch.cat([negative_prompt_embeds,
48      return prompt_embeds
49
50  def get_embes(self, prompt, max_length):
51      """
52      Encode the text prompt into embeddings using the t
53      """
54      text_inputs = self.tokenizer(prompt, padding="max_
55      text_input_ids = text_inputs.input_ids.to(self.dev
56      prompt_embeds = self.text_encoder(text_input_ids)[
57      return prompt_embeds
```

Step 1: encode_prompt

Where are Necessities used?

- L54: `CLIPTokenizer` : Token into text (Prompt). By making it a vector, it makes it easier to handle AI.
- L56: `CLIPTextModel` : Multi -modal model of language and image. In the image generation, the expression (embedding) of the image you want to make at the prompt is extracted.

pipeline.py#L21-L39

```
@classmethod
def from_pretrained(cls, model_name, device=torch.device("
  # Ommit comments
  tokenizer = CLIPTokenizer.from_pretrained(model_name, su
  text_encoder = CLIPTextModel.from_pretrained(model_name,
  scheduler = PareDDIMScheduler.from_config(model_name, su
  unet = PareUNet2DConditionModel.from_pretrained(model_na
  vae = PareAutoencoderKL.from_pretrained(model_name, subf
  return cls(tokenizer, text_encoder, scheduler, unet, vae
```

pipeline.py#L50-L57

```
50    def get_embes(self, prompt, max_length):
51        """
52        Encode the text prompt into embeddings using the t
53        """
54        text_inputs = self.tokenizer(prompt, padding="max_
55        text_input_ids = text_inputs.input_ids.to(self.dev
56        prompt_embeds = self.text_encoder(text_input_ids)[
57        return prompt_embeds
```

Step 1: encode_prompt

Understand the whole flow

- L54: `CLIPTokenizer` : Token into text (Prompt). By making it a vector, it makes it easier to handle AI.
- L56: `CLIPTextModel` : Multi -modal model of language and image. In the image generation, the expression (embedding) of the image you want to make at the prompt is extracted.
- L46: Negative_prompt is an empty character string to make it simple.

pipeline.py#L34-L35

```
34      tokenizer = CLIPTokenizer.from_pretrained(model_na
35      text_encoder = CLIPTextModel.from_pretrained(model
```

pipeline.py#L41-L57

```
41  def encode_prompt(self, prompt: str):
42      """
43      Encode the text prompt into embeddings using the t
44      """
45      prompt_embeds = self.get_embes(prompt, self.tokeni
46      negative_prompt_embeds = self.get_embes([''], prom
47      prompt_embeds = torch.cat([negative_prompt_embeds,
48      return prompt_embeds
49
50  def get_embes(self, prompt, max_length):
51      """
52      Encode the text prompt into embeddings using the t
53      """
54      text_inputs = self.tokenizer(prompt, padding="max_
55      text_input_ids = text_inputs.input_ids.to(self.dev
56      prompt_embeds = self.text_encoder(text_input_ids)[
57      return prompt_embeds
```

In [15]:

```
prompt = "painting depicting the sea, sunrise, ship, artstation, 4k, concept art"

text_inputs = pipe.tokenizer(prompt, padding="max_length", max_length=max_length, truncation=True,

print(text_inputs.input_ids.shape, '\n')
print(text_inputs.input_ids)
```

Out [15]:

```
torch.Size([1, 77])

tensor([[49406,  3086, 24970,   518,  2102,   267,  5610,   267,  1158,   267,
          1486,  2631,   267,   275,   330,   267,  6353,   794, 49407,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0]])
```

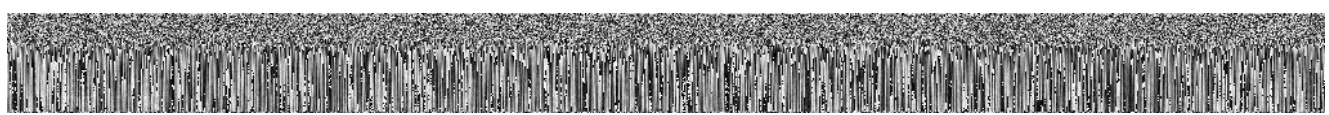
In [18]:

```
text_input_ids = text_inputs.input_ids.to(pipe.device)
prompt_embeds = pipe.text_encoder(text_input_ids)[0].to(dtype=pipe.dtype, device=pipe.device)

print(prompt_embeds.shape, '\n')
print(prompt_embeds, '\n')
display(pipe.tensor_to_image(prompt_embeds))
```

Out [18]:

```
torch.Size([1, 77, 1024])

tensor([[[-0.3135, -0.4475, -0.0083,  ...,  0.2544, -0.0327, -0.2959],
         [ 0.6030, -0.9849, -0.6772,  ...,  1.6953, -0.1359, -0.3333],
         [-0.1290, -0.3606, -0.1376,  ...,  0.5244, -0.8164,  0.8135],
         ...,
         [ 0.4788, -1.6396,  0.7124,  ..., -0.0309, -0.2939,  0.4023],
         [ 0.4790, -1.6875,  0.7222,  ..., -0.2104, -0.3623,  0.5020],
         [ 0.5854, -2.2852,  0.3623,  ..., -0.2250,  0.1294,  0.7236]]],
       device='cuda:0', dtype=torch.float16, grad_fn=<NativeLayerNormBackward0>)
```

Out [18]:



Ch 0.1.2 Play prompt_embeds

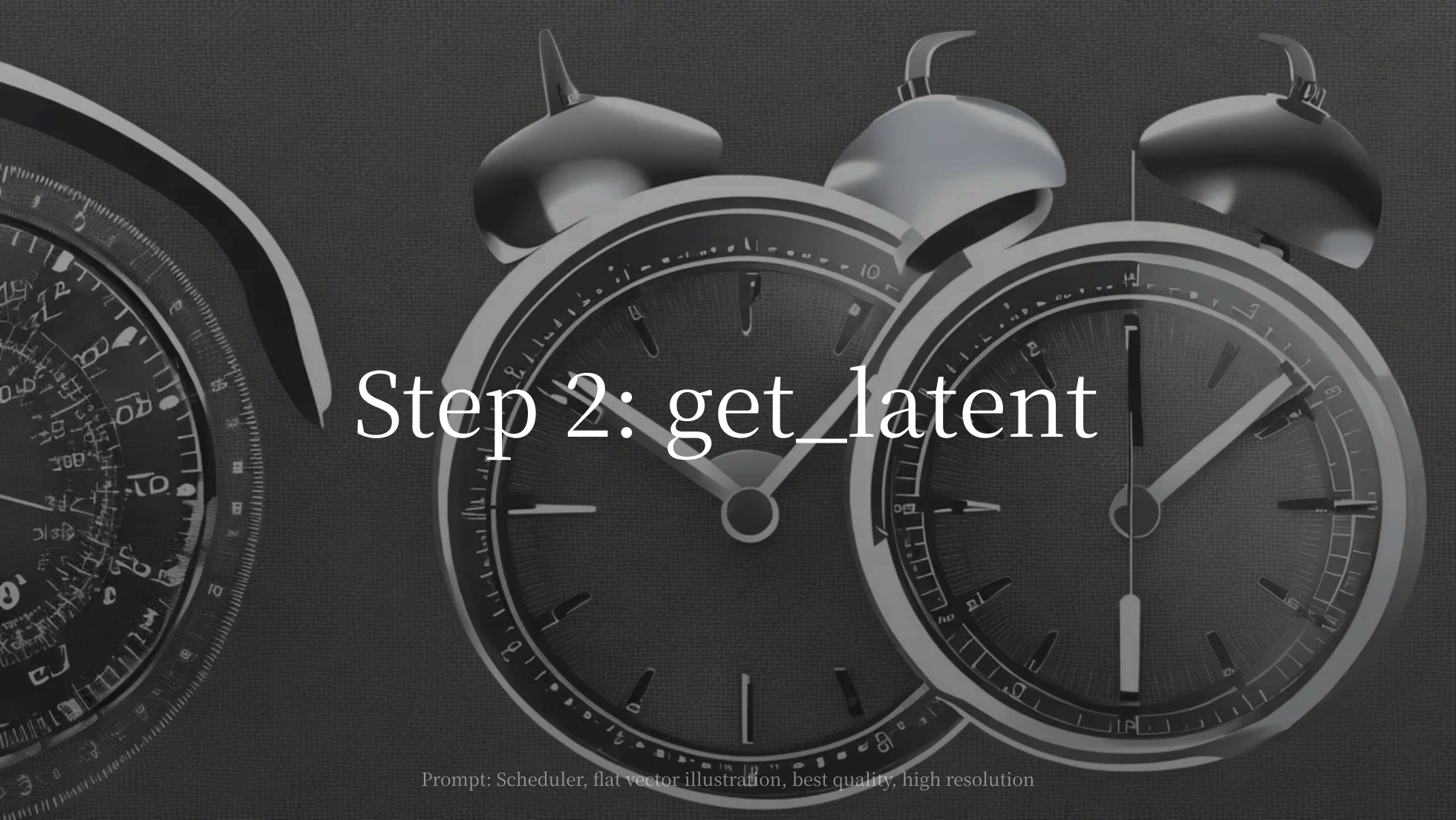
Part 0: Setup

In [1]:

```
# Install diffusers and parediffusers
!pip install transformers diffusers accelerate -U
!pip install parediffusers==0.0.0
```

Out [1]:

```
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.36.2)
Requirement already satisfied: diffusers in /usr/local/lib/python3.10/dist-packages (0.25.0)
Requirement already satisfied: accelerate in /usr/local/lib/python3.10/dist-packages (0.25.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transfo
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transfo
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from tra
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transfor
Requirement already satisfied: tokenizers<0.19,>=0.14 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transforme
Requirement already satisfied: huggingface-hub<1.0,>=0.19.3 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transforme
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from t
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packages (from diffusers) (
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: torch>=1.10.0 in /usr/local/lib/python3.10/dist-packages (from accel
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from accelerate)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from hu
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-packages (from importlib
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: idna<4,>=2.5 in /usr/lib/python3/dist-packages (from requests->trans
Requirement already satisfied: certifi>=2017.4.17 in /usr/lib/python3/dist-packages (from requests-
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (fr
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour
Obtaining file:///
ERROR: file:/// does not appear to be a Python project: neither 'setup.py' nor 'pyproject.toml' fou
```



Step 2: get_latent

Prompt: Scheduler, flat vector illustration, best quality, high resolution

Step 2: get_latent

Generate random tensor of 1/8 size

Necessities

torch.randn

Step 2: get_latent

Understand the whole flow

- L63: Generate random tensor of 1/8 size

pipeline.py#L59-L65

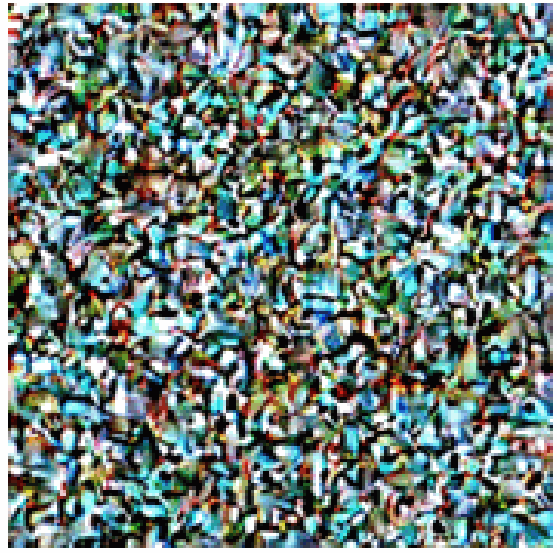
```
59  def get_latent(self, width: int, height: int):
60    """
61    Generate a random initial latent tensor to start
62    """
63    return torch.randn((4, width // 8, height // 8)
64      device=self.device, dtype=self.dtype
65    )
```


Step 3: denoise
Prompt: UNet, watercolor painting, detailed, brush strokes, best quality, high resolution

Step 3: denoise

Denosing by using Scheduler and UNet.

understand-stable-diffusion-slidev-notebooks/denoise.ipynb



Index: 0

understand-stable-diffusion-slidev-notebooks/denoise.ipynb

Necessities

scheduler.py

unet.py

Step 3: denoise

Aside from Necessities, the whole flow

Step 3: denoise

Where are Necessities used?

- L86: UNet
- L91: Scheduler

pipeline.py#L75-L93

```
75  @torch.no_grad()
76  def denoise(self, latents, prompt_embeds, num_inferenc
77    """
78    Iteratively denoise the latent space using the diff
79    """
80    timesteps, num_inference_steps = self.retrieve_time
81
82    for t in timesteps:
83      latent_model_input = torch.cat([latents] * 2)
84
85      # Predict the noise residual for the current time
86      noise_residual = self.unet(latent_model_input, t,
87      uncond_residual, text_cond_residual = noise_resid
88      guided_noise_residual = uncond_residual + guidanc
89
90      # Update latents by reversing the diffusion proce
91      latents = self.scheduler.step(guided_noise_residu
92
93    return latents
```

Step 3: denoise

Where are Necessities used?

- L86: UNet2DConditionModel
- L91: DDIMScheduler

pipeline.py#L21-L39

```
@classmethod
def from_pretrained(cls, model_name, device=torch.device("c
  # Ommit comments
  tokenizer = CLIPTokenizer.from_pretrained(model_name, sub
  text_encoder = CLIPTextModel.from_pretrained(model_name,
  scheduler = PareDDIMScheduler.from_config(model_name, sub
  unet = PareUNet2DConditionModel.from_pretrained(model_nam
  vae = PareAutoencoderKL.from_pretrained(model_name, subfo
  return cls(tokenizer, text_encoder, scheduler, unet, vae,
```

pipeline.py#L82-L93

```
82      for t in timesteps:
83        latent_model_input = torch.cat([latents] * 2)
84
85        # Predict the noise residual for the current time
86        noise_residual = self.unet(latent_model_input, t,
87        uncond_residual, text_cond_residual = noise_resid
88        guided_noise_residual = uncond_residual + guidanc
89
90        # Update latents by reversing the diffusion proce
91        latents = self.scheduler.step(guided_noise_residu
```

Step 3: denoise

Understand the whole flow

- L80: Acquisition of timesteps using Scheduler
  (Scheduler will be described later)
- L82: timesteps length loop
  (timesteps length = num_inference_steps)
- L86: Denose by UNet
  (UNet will be described later)
- L88: Calculate how much considering the prompt
  (Reference: Jonathan Ho, Tim Salimans: “Classifier-Free Diffusion Guidance”, 2022; arXiv:2207.12598.)
- L91: The strength of the deny is determined by Scheduler.

pipeline.py#L82-L93

```
75    @torch.no_grad()
76    def denoise(self, latents, prompt_embeds, num_inferenc
77      """
78      Iteratively denoise the latent space using the diff
79      """
80      timesteps, num_inference_steps = self.retrieve_time
81
82      for t in timesteps:
83        latent_model_input = torch.cat([latents] * 2)
84
85        # Predict the noise residual for the current time
86        noise_residual = self.unet(latent_model_input, t,
87        uncond_residual, text_cond_residual = noise_resid
88        guided_noise_residual = uncond_residual + guidanc
89
90        # Update latents by reversing the diffusion proce
91        latents = self.scheduler.step(guided_noise_residu
92
93      return latents
```

scheduler.py

Determine the strength of denoising

```python
import json
import numpy as np
import torch
from huggingface_hub import hf_hub_download
from .utils import DotDict


class PareDDIMScheduler:
    def __init__(self, config_dict: dict):
        """Initialize beta and alpha values for the scheduler."""
        self.config = DotDict(**config_dict)
        self.betas = (
            torch.linspace(
                self.config.beta_start**0.5,
                self.config.beta_end**0.5,
                self.config.num_train_timesteps,
                dtype=torch.float32,
            )
            ** 2
        )
        self.alphas = 1.0 - self.betas
        self.alphas_cumprod = torch.cumprod(self.alphas, dim=0)
        self.final_alpha_cumprod = torch.tensor(1.0)

    @classmethod
    def from_config(
        cls,
        model_name: str,
        subfolder: str = "scheduler",
        filename: str = "scheduler_config.json",
    ) -> "PareDDIMScheduler":
        """Create scheduler instance from configuration file."""
        config_file = hf_hub_download(
            model_name, filename=filename, subfolder=subfolder
        )
        with open(config_file, "r", encoding="utf-8") as reader:
            text = reader.read()
        config_dict = json.loads(text)
        return cls(config_dict)

    def set_timesteps(
        self, num_inference_steps: int, device: torch.device = None
    ) -> None:
        """Set the timesteps for the scheduler based on the number of inference steps."""
        self.num_inference_steps = num_inference_steps
        step_ratio = self.config.num_train_timesteps // self.num_inference_steps
        timesteps = (
            (np.arange(0, num_inference_steps) * step_ratio)
            .round()[::-1]
            .copy()
            .astype(np.int64)
        )
        timesteps += self.config.steps_offset
        self.timesteps = torch.from_numpy(timesteps).to(device)

    def step(
        self,
        model_output: torch.FloatTensor,
        timestep: int,
        sample: torch.FloatTensor,
    ) -> list:
        """Perform a single step of denoising in the diffusion process."""
        prev_timestep = (
```

Scheduler

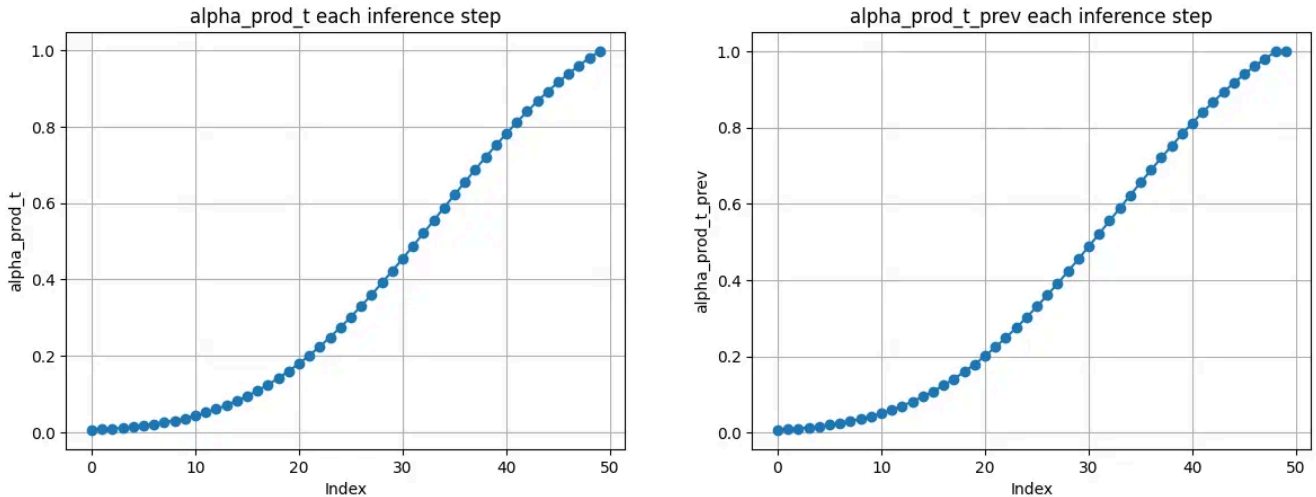
- L49: Get alpha_prod_t(0~1.0)
  (Indicates how much the original data is retained.)
- L50: Get alpha_prod_t_prev(0~1.0)
- L52: alpha_prod_t + beta_prod_t = 1
- L53: Estimate the original sample from the current sample and model output.
- L54: Calculate the estimated value of the added noise.
- L56: Calculate the direction to restore it to the original image.
- L57: Calculate the sample that goes one step further in the denoising by using 3 values.

scheduler.py#L40-L59

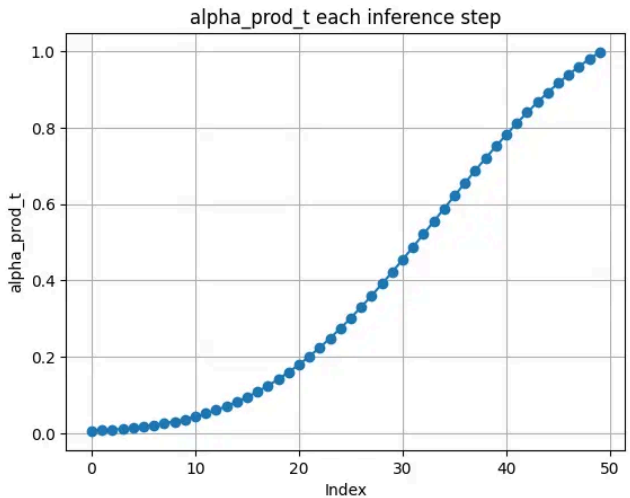
```
40   def step(
41     self,
42     model_output: torch.FloatTensor,
43     timestep: int,
44     sample: torch.FloatTensor,
45   ) -> list:
46     """Perform a single step of denoising in the diffusion p
47     prev_timestep = timestep - self.config.num_train_timeste
48
49     alpha_prod_t = self.alphas_cumprod[timestep]
50     alpha_prod_t_prev = self.alphas_cumprod[prev_timestep] :
51
52     beta_prod_t = 1 - alpha_prod_t
53     pred_original_sample = (alpha_prod_t**0.5) * sample - (b
54     pred_epsilon = (alpha_prod_t**0.5) * model_output + (bet
55
56     pred_sample_direction = (1 - alpha_prod_t_prev) ** (0.5)
57     prev_sample = alpha_prod_t_prev ** (0.5) * pred_original
58
59     return prev_sample, pred_original_sample
```

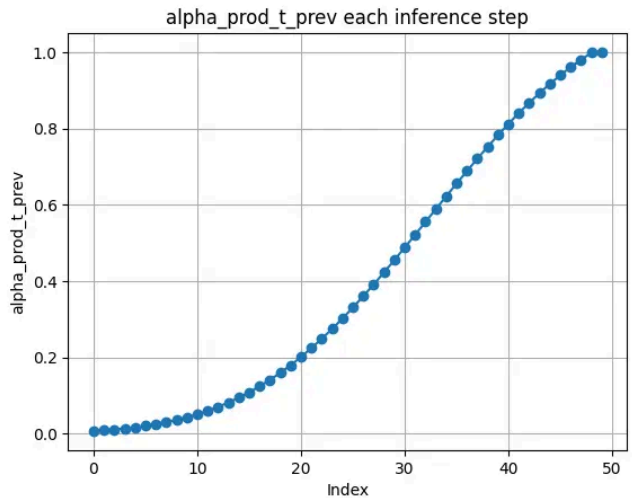
scheduler.py#L40-L59

```python
  def step(
    self,
    model_output: torch.FloatTensor,
    timestep: int,
    sample: torch.FloatTensor,
  ) -> list:
    """Perform a single step of denoising in the diffusion process."""
    prev_timestep = timestep - self.config.num_train_timesteps // self.num_inference_steps

    alpha_prod_t = self.alphas_cumprod[timestep]
    alpha_prod_t_prev = self.alphas_cumprod[prev_timestep] if prev_timestep >= 0 else self.final_alpha_cumprod

    beta_prod_t = 1 - alpha_prod_t
    pred_original_sample = (alpha_prod_t**0.5) * sample - (beta_prod_t**0.5) * model_output
    pred_epsilon = (alpha_prod_t**0.5) * model_output + (beta_prod_t**0.5) * sample

    pred_sample_direction = (1 - alpha_prod_t_prev) ** (0.5) * pred_epsilon
    prev_sample = alpha_prod_t_prev ** (0.5) * pred_original_sample + pred_sample_direction

    return prev_sample, pred_original_sample
```

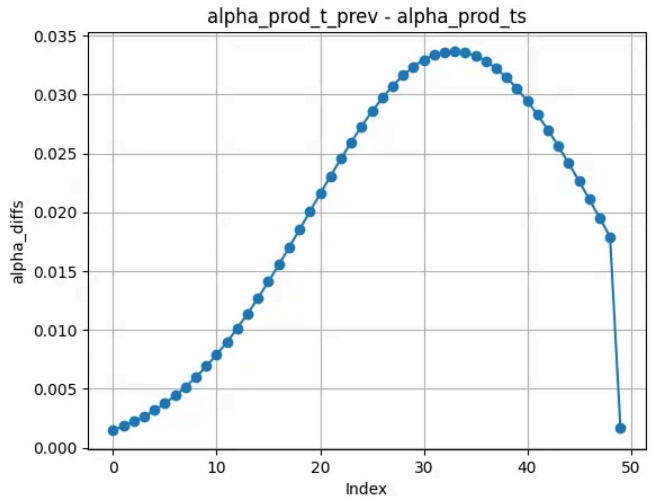


understand-stable-diffusion-slidev-notebooks/scheduler.ipynb





understand-stable-diffusion-slidev-notebooks/scheduler.ipynb



understand-stable-diffusion-slidev-notebooks/scheduler.ipynb

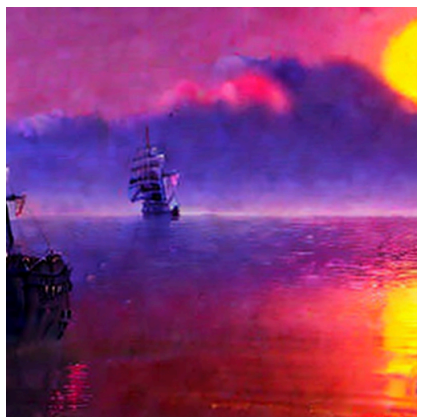
In [5]:

```
latents = init_latents.detach().clone()

# Denoise loop without scheduler
@torch.no_grad()
def custom_denoise(
        pipe, latents, prompt_embeds, num_inference_steps, guidance_scale, ratio
):
        for i in range(num_inference_steps):
                latent_model_input = torch.cat([latents] * 2)

                # Predict the noise residual for the current timestep
                t = torch.tensor((1000/num_inference_steps)*(num_inference_steps-i-1)+1, device=pip
                noise_residual = pipe.unet(latent_model_input, t, encoder_hidden_states=prompt_embe
                uncond_residual, text_cond_residual = noise_residual.chunk(2)
                guided_noise_residual = uncond_residual + guidance_scale * (text_cond_residual - un

                # Update the latents with the predicted noise residual
                latents = latents - (ratio * guided_noise_residual / num_inference_steps)

                #display(pipe.vae_decode(latents))
        return latents

ratio = 1.475
latents = custom_denoise(pipe, latents, prompt_embeds, num_inference_steps, guidance_scale, ratio)
image = pipe.vae_decode(latents)
image
```

Out [5]:

```
/tmp/ipykernel_32/1826561155.py:12: DeprecationWarning: an integer is required (got type float).  I
  t = torch.tensor((1000/num_inference_steps)*(num_inference_steps-i-1)+1, device=pipe.device, dtyp
```

Out [5]:

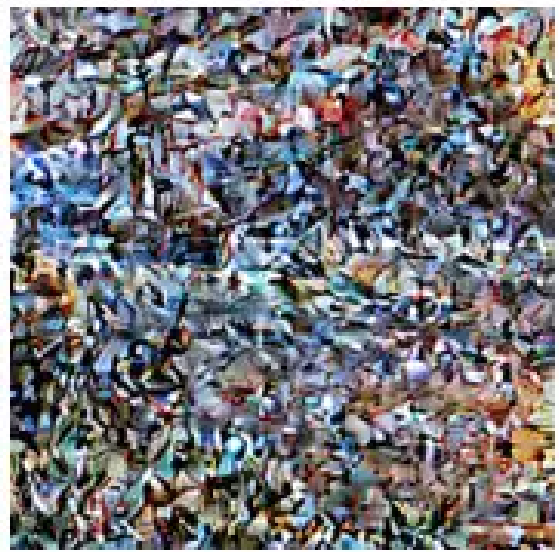


In [4]:

```
latents = init_latents.detach().clone()

@torch.no_grad()
def denoise(
        pipe, latents, prompt_embeds, num_inference_steps=50, guidance_scale=7.5
):
        timesteps, num_inference_steps = pipe.retrieve_timesteps(num_inference_steps)

        for t in timesteps:
                latent_model_input = torch.cat([latents] * 2)

                # Predict the noise residual for the current timestep
                noise_residual = pipe.unet(
                        latent_model_input, t, encoder_hidden_states=prompt_embeds
                )
                uncond_residual, text_cond_residual = noise_residual.chunk(2)
                guided_noise_residual = uncond_residual + guidance_scale * (
                        text_cond_residual - uncond_residual
                )

                # Update latents by reversing the diffusion process for the current timestep
                latents = pipe.scheduler.step(guided_noise_residual, t, latents)[0]

        return latents

latents = denoise(pipe, latents, prompt_embeds, num_inference_steps, guidance_scale)
image = pipe.vae_decode(latents)
image
```

Out [4]:

I don't have any idea why `ratio = 1.5` looks good.



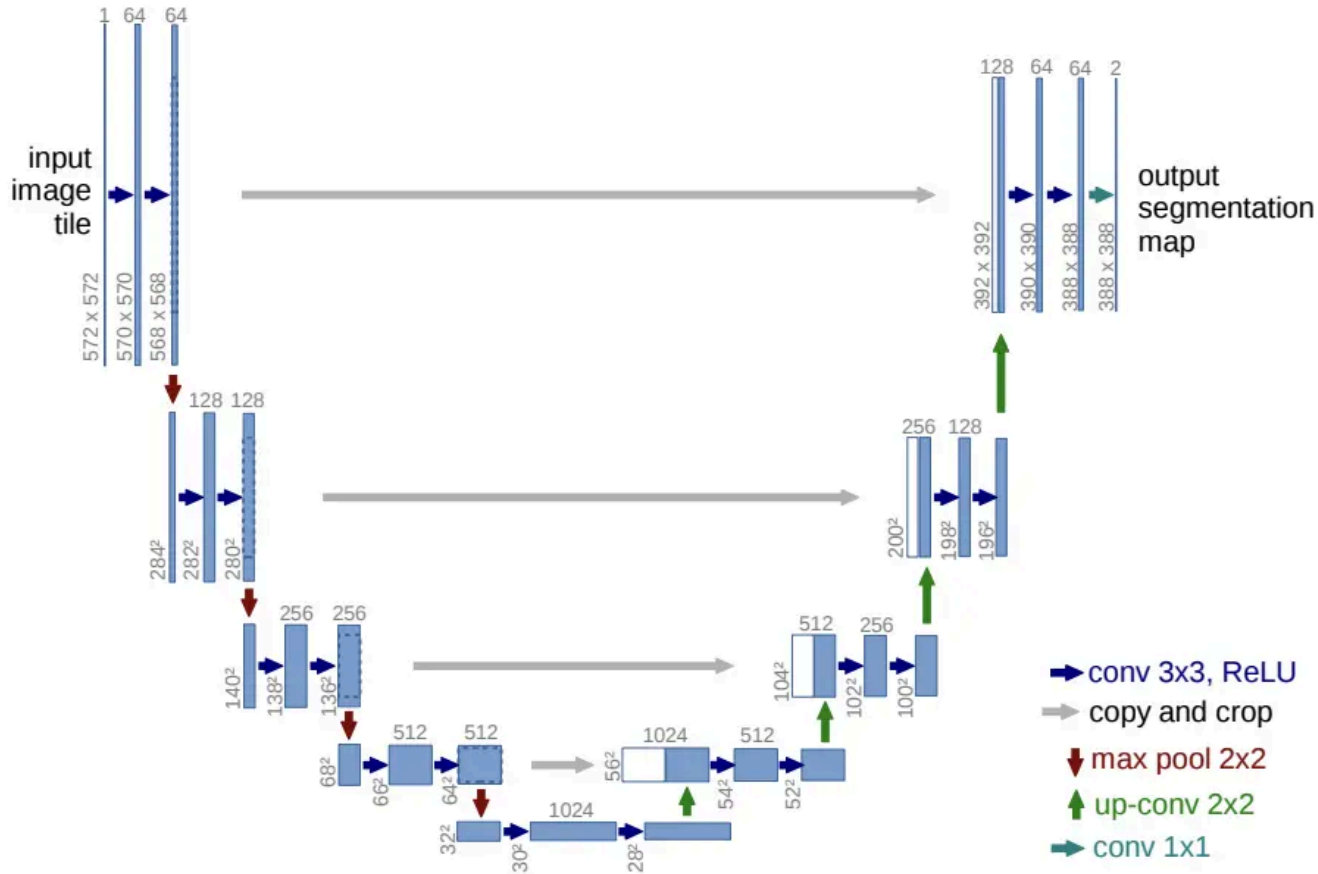
0.5000

understand-stable-diffusion-slidev-notebooks/scheduler_necessity.ipynb

unet.py

Using for Denoising

```python
import torch
from torch import nn
from typing import List, Union
import json
from huggingface_hub import hf_hub_download
from .utils import DotDict, get_activation
from .defaults import DEFAULT_UNET_CONFIG
from .models.embeddings import PareTimestepEmbedding, PareTimesteps
from .models.unet_2d_get_blocks import pare_get_down_block, pare_get_up_block
from .models.unet_2d_mid_blocks import PareUNetMidBlock2DCrossAttn


class PareUNet2DConditionModel(nn.Module):
    def __init__(self, **kwargs):
        super().__init__()
        self.config = DotDict(DEFAULT_UNET_CONFIG)
        self.config.update(kwargs)
        self.config.only_cross_attention = [self.config.only_cross_attention] * len(
            self.config.down_block_types
        )
        self.config.num_attention_heads = (
            self.config.num_attention_heads or self.config.attention_head_dim
        )
        self._setup_model_parameters()

        self._build_input_layers()
        self._build_time_embedding()
        self._build_down_blocks()
        self._build_mid_block()
        self._build_up_blocks()
        self._build_output_layers()

    def _setup_model_parameters(self) -> None:
        if isinstance(self.config.num_attention_heads, int):
            self.config.num_attention_heads = (self.config.num_attention_heads,) * len(
                self.config.down_block_types
            )
        if isinstance(self.config.attention_head_dim, int):
            self.config.attention_head_dim = (self.config.attention_head_dim,) * len(
                self.config.down_block_types
            )
        if isinstance(self.config.cross_attention_dim, int):
            self.config.cross_attention_dim = (self.config.cross_attention_dim,) * len(
                self.config.down_block_types
            )
        if isinstance(self.config.layers_per_block, int):
            self.config.layers_per_block = [self.config.layers_per_block] * len(
                self.config.down_block_types
            )
        if isinstance(self.config.transformer_layers_per_block, int):
            self.config.transformer_layers_per_block = [
                self.config.transformer_layers_per_block
            ] * len(self.config.down_block_types)

    def _build_input_layers(self) -> None:
        conv_in_padding = (self.config.conv_in_kernel - 1) // 2
        self.conv_in = nn.Conv2d(
            self.config.in_channels,
            self.config.block_out_channels[0],
            kernel_size=self.config.conv_in_kernel,
            padding=conv_in_padding,
        )

```



Olaf Ronneberger, Philipp Fischer, Thomas Brox: “U-Net: Convolutional Networks for Biomedical Image Segmentation”, 2015; arXiv:1505.04597.

Create UNet using Resnet and Transformer

```python
for i in range(num_layers):
    in_channels = in_channels if i == 0 else out_channels
    resnets.append(
        PareResnetBlock2D(
            in_channels=in_channels,
            out_channels=out_channels,
            temb_channels=temb_channels,
            eps=resnet_eps,
            groups=resnet_groups,
            dropout=dropout,
            non_linearity=resnet_act_fn,
            output_scale_factor=output_scale_factor,
        )
    )
    if not dual_cross_attention:
        attentions.append(
            PareTransformer2DModel(
                num_attention_heads,
                out_channels // num_attention_heads,
                in_channels=out_channels,
                num_layers=transformer_layers_per_block[i],
                cross_attention_dim=cross_attention_dim,
                norm_num_groups=resnet_groups,
                use_linear_projection=use_linear_projection,
                only_cross_attention=only_cross_attention,
                upcast_attention=upcast_attention,
            )
        )
```

src/parediffusers/models/unet_2d_blocks.py delivered with ❤️ by emgithub

view raw


Step 4: vae_decode
Prompt: VAE, abstract style, highly detailed, colors and shapes

Step 4: vae_decode

Decode into the image with VAE

Step 4: vae_decode

Understand the whole flow

- L112: Decode into the image space.

- L113: Normalization is performed during training, so denormalize it.

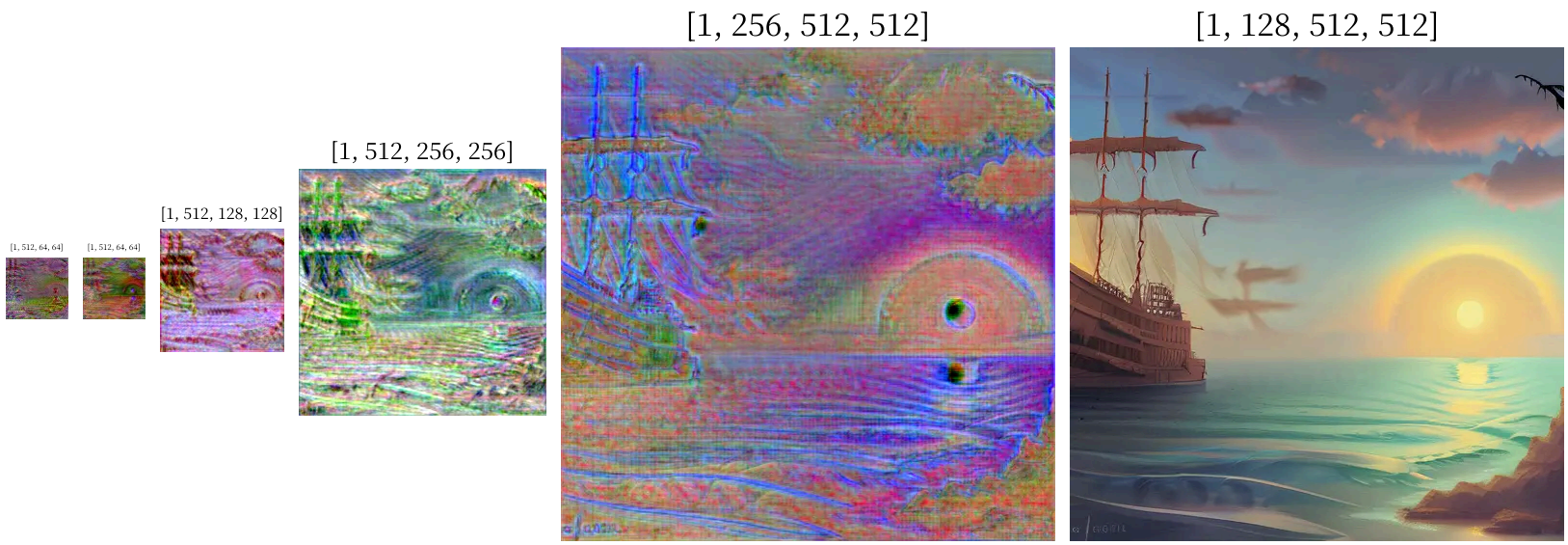
- L114: Convert from tensor to Pil.image

pipeline.py#L107-L105

```
107   @torch.no_grad()
108   def vae_decode(self, latents):
109     """
110     Decode the latent tensors using the VAE to produce an image.
111     """
112     image = self.vae.decode(latents / self.vae.config.scaling_facto
113     image = self.denormalize(image)
114     image = self.tensor_to_image(image)
115     return image
```

Variational Autoencoder (VAE)


```python
import json
from typing import List
import torch
import torch.nn as nn
from huggingface_hub import hf_hub_download
from .utils import DotDict
from .defaults import DEFAULT_VAE_CONFIG
from .models.vae_blocks import (
    PareEncoder,
    PareDecoder,
    PareDiagonalGaussianDistribution,
)


class PareAutoencoderKL(nn.Module):
    def __init__(self, **kwargs):
        super().__init__()
        self.config = DotDict(DEFAULT_VAE_CONFIG)
        self.config.update(kwargs)

        # pass init params to Encoder
        self.encoder = PareEncoder(
            in_channels=self.config.in_channels,
            out_channels=self.config.latent_channels,
            down_block_types=self.config.down_block_types,
            block_out_channels=self.config.block_out_channels,
            layers_per_block=self.config.layers_per_block,
        )

        # pass init params to Decoder
        self.decoder = PareDecoder(
            in_channels=self.config.latent_channels,
            out_channels=self.config.out_channels,
            up_block_types=self.config.up_block_types,
            block_out_channels=self.config.block_out_channels,
            layers_per_block=self.config.layers_per_block,
        )

        self.quant_conv = nn.Conv2d(
            2 * self.config.latent_channels, 2 * self.config.latent_channels, 1
        )
        self.post_quant_conv = nn.Conv2d(
            self.config.latent_channels, self.config.latent_channels, 1
        )

        self.use_slicing = False
        self.use_tiling = False

        # only relevant if vae tiling is enabled
        self.tile_sample_min_size = self.config.sample_size
        sample_size = (
            self.config.sample_size[0]
            if isinstance(self.config.sample_size, (list, tuple))
            else self.config.sample_size
        )
        self.tile_latent_min_size = int(
            sample_size / (2 ** (len(self.config.block_out_channels) - 1))
        )
        self.tile_overlap_factor = 0.25

    @classmethod
    def _get_config(
        cls, model_name: str, filename: str = "config.json", subfolder: str = "unet"
```



understand-stable-diffusion-slidev-notebooks/vae.ipynb

9. Conclusion

Prompt: Summary, long-exposure photography, masterpieces

It’s fun to read the library code!

A dissertation quoted everywhere in the library

diffusers/.../pipeline.py

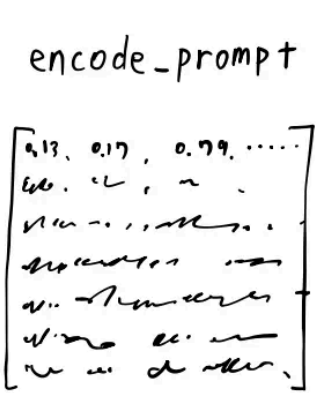
Conclusion

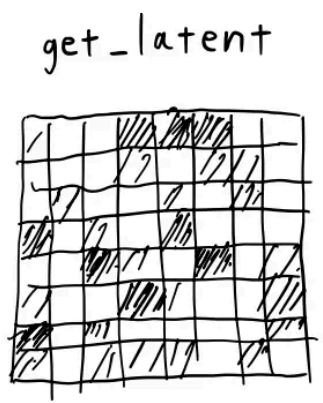
Step 1: Convert prompt to embedding.
Step 2: Create random Latent.
Step 3: Denosing by using Scheduler and UNet.
Step 4: Decode to pixel space by VAE.

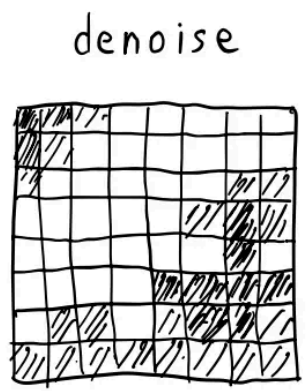
Conclusion

pipeline.py#L117-L135

```python
def __call__(self, prompt: str, height: int = 512, width: int = 512, ...):
  prompt_embeds = self.encode_prompt(prompt)
  latents = self.get_latent(width, height).unsqueeze(dim=0)
  latents = self.denoise(latents, prompt_embeds, ...)
  image = self.vae_decode(latents)
  return image
```

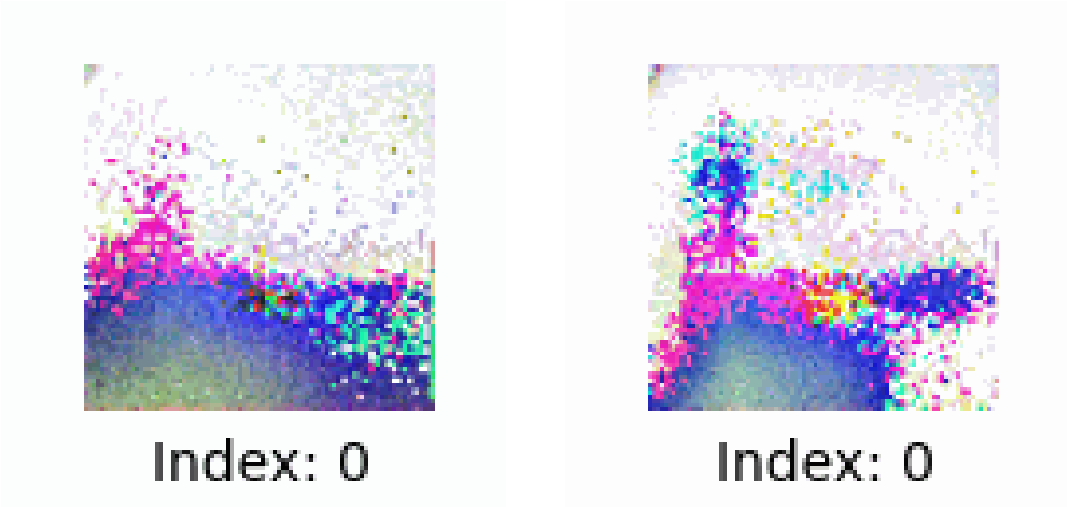






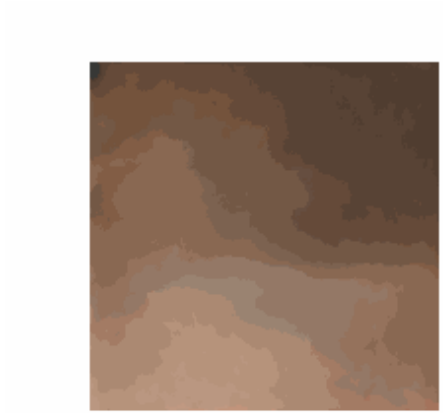
Appendix

Other denoising samples

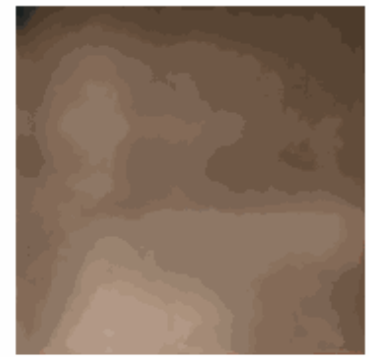


understand-stable-diffusion-slidev-notebooks/denoise.ipynb

Other decorded denoising samples



Index: 0



Index: 0

understand-stable-diffusion-slidev-notebooks/denoise.ipynb

UNet is really U form?

```
init            torch.Size([2, 4, 64, 64])
conv_in         torch.Size([2, 320, 64, 64])

down_blocks_0   torch.Size([2, 320, 32, 32])
down_blocks_1   torch.Size([2, 640, 16, 16])
down_blocks_2   torch.Size([2, 1280, 8, 8])
down_blocks_3   torch.Size([2, 1280, 8, 8])

mid_block       torch.Size([2, 1280, 8, 8])

up_blocks0      torch.Size([2, 1280, 16, 16])
up_blocks1      torch.Size([2, 1280, 32, 32])
up_blocks2      torch.Size([2, 640, 64, 64])
up_blocks3      torch.Size([2, 320, 64, 64])

conv_out        torch.Size([2, 4, 64, 64])
```

understand-stable-diffusion-slidev-notebooks/unet.ipynb