

A dark, atmospheric cyberpunk cityscape at night. The scene is filled with tall, multi-story buildings covered in intricate circuitry and glowing neon signs in shades of red, blue, and green. Rain falls heavily, creating a reflective surface on the wet streets. The overall mood is mysterious and futuristic.

# 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-slides](#)

 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

- Qunderstand-stable-diffusion-slidev: Repository of this slide.
- Qunderstand-stable-diffusion-slidev-notebooks: Notebooks for generating sample images and gifs.
- Qparediffusers: Simple library for generating images without using huggingface/diffusers.

# Table of Contents

1. [Understand Stable Diffusion from Code](#)
2. [Introduction](#)
3. [Table of Contents](#)
4. [Flow of Image Generation](#)
5. [Step 1: encode\\_prompt](#)
6. [Step 2: get\\_latent](#)
7. [Step 3: denoise](#)
8. [Step 4: vae\\_decode](#)
9. [Conclusion](#)
10. [Appendix](#)

A watercolor painting of a forest scene. In the foreground, several tall, thin trees stand in a row, their trunks dark and silhouetted against a bright sky. To the left, a small, simple building with a dark roof and a visible doorway is nestled among the trees. A path or road leads from the viewer's perspective through the trees towards the building. The background is filled with more trees and foliage, with soft, blended colors of blues, purples, and yellows creating a dreamlike atmosphere. The overall style is artistic and painterly.

## 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>



Open in Colab

Install the Diffusers library:

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

Generate an image from text:

```
1 import torch
2 from diffusers import StableDiffusionPipeline
3
4 pipe = StableDiffusionPipeline.from_pretrained(
5     "stabilityai/stable-diffusion-2",
6     dtype=torch.float16,
7 ).to(device=torch.device("cuda"))
8 prompt = "painting depicting the sea, sunrise,
9
10 image = pipe(prompt, width=512, height=512).im
11 display(image)
```



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

# diffusers/.../pipeline\_stable\_diffusion.py

```
1 # Copyright 2024 The HuggingFace Team. All rights reserved.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14 import inspect
15 from typing import Any, Callable, Dict, List, Optional, Union
16
17 import torch
18 from packaging import version
19 from transformers import CLIPImageProcessor, CLIPTextModel, CLIPTokenizer, CLIPVisionModelWithProjection
```

# parediffusers.../pipeline.py

```
1 import torch
2 from torchvision.transforms import ToPILImage
3 from transformers import CLIPTokenizer, CLIPTextModel
4 from .scheduler import PareDDIMScheduler
5 from .unet import PareUNet2DConditionModel
6 from .vae import PareAutoencoderKL
7
8
9 class PareDiffusionPipeline:
10     def __init__(
11         self,
12         tokenizer,
13         text_encoder,
14         scheduler,
15         unet,
16         vae,
17         device=torch.device("cuda"),
18         dtype=torch.float16,
19     ):
```



Open in Colab

Install the PareDiffusers library:

```
1 !pip install parediffusers
```

Generate an image from text:

```
1 import torch
2 from parediffusers import PareDiffusionPipeline
3
4 pipe = PareDiffusionPipeline.from_pretrained(
5     "stabilityai/stable-diffusion-2",
6     device=torch.device("cuda"),
7     dtype=torch.float16,
8 )
9 prompt = "painting depicting the sea, sunrise,
10
11 image = pipe(prompt, width=512, height=512)
12 display(image)
```



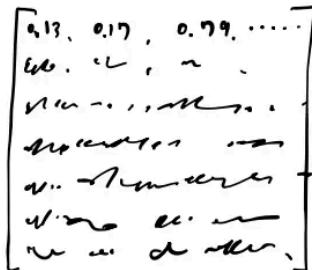
How is image generation performed?

## pipeline.py#L117-L135

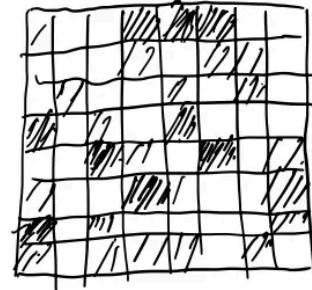
1. `encode\_prompt` : Convert prompt to embedding.
2. `get\_latent` : Create random Latent.
3. `denoise` : Denosing by using Scheduler and UNet.
4. `vae\_decode` : Decode to pixel space by VAE.

1. `encode\_prompt` : Convert prompt to embedding.
2. `get\_latent` : Create random Latent.
3. `denoise` : Denoising by using Scheduler and UNet.
4. `vae\_decode` : Decode to pixel space by VAE.

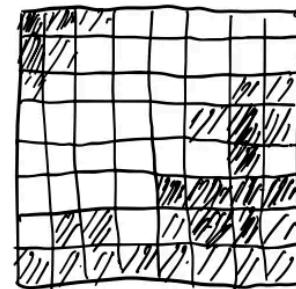
encode\_prompt



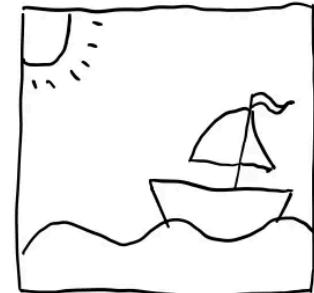
get\_latent



denoise



vae\_decode



# A Briefly Theory

What is Latent Diffusion Model (LDM)?

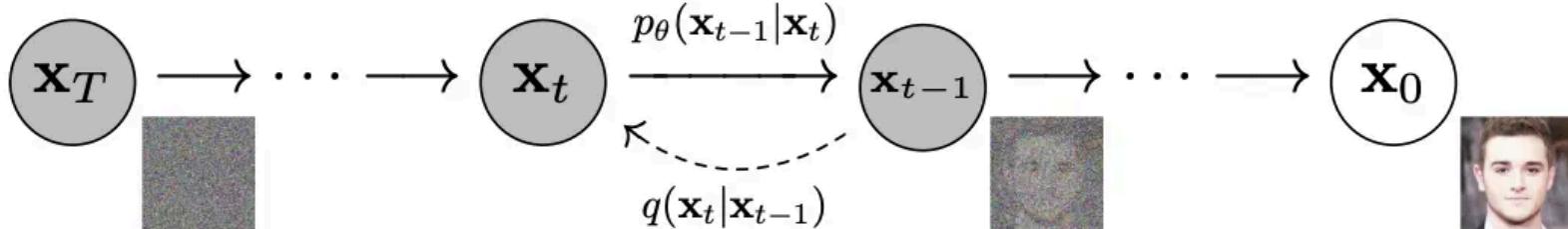
Models Which Run DDPM on Latent Space

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.

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

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

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

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

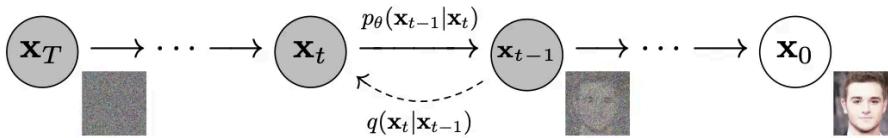
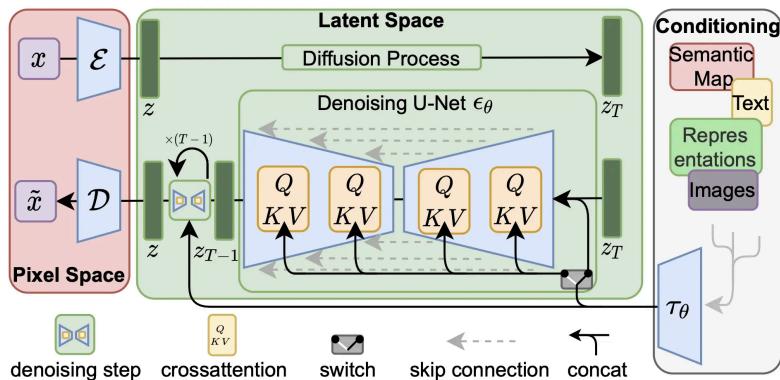


Figure 2: The directed graphical model considered in this work.

Jonathan Ho, Ajay Jain, Pieter Abbeel: “Denoising Diffusion Probabilistic Models”, 2020; [arXiv:2006.11239](https://arxiv.org/abs/2006.11239).



Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, Björn Ommer: “High-Resolution Image Synthesis with Latent Diffusion Models”, 2021; [arXiv:2112.10752](https://arxiv.org/abs/2112.10752).

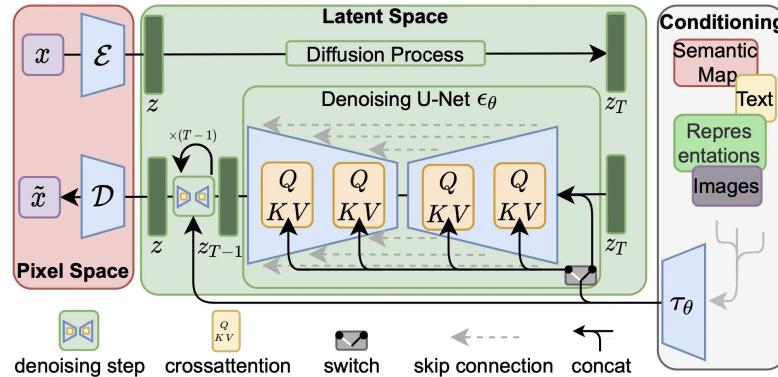
```

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](https://arxiv.org/abs/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

## Qpipeline.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?

Q pipeline.py#L41-L57

```
41 def encode_prompt(self, prompt: str):
42     """
43         Encode the text prompt into embeddings using the t
44         nnn
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         nnn
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.

## Q pipeline.py#L21-L39

```
@classmethod
def from_pretrained(cls, model_name, device=torch.device("cpu",
    # Omit comments
    tokenizer = CLIPTokenizer.from_pretrained(model_name, subfolder="tokenizer")
    text_encoder = CLIPTextModel.from_pretrained(model_name, subfolder="text_encoder")
    scheduler = PareDDIMScheduler.from_config(model_name, subfolder="scheduler")
    unet = PareUNet2DConditionModel.from_pretrained(model_name, subfolder="unet")
    vae = PareAutoencoderKL.from_pretrained(model_name, subfolder="vae")
    return cls(tokenizer, text_encoder, scheduler, unet, vae)
```

## Q pipeline.py#L50-L57

```
50 def get_embs(self, prompt, max_length):
51     """
52         Encode the text prompt into embeddings using the text encoder
53     """
54     text_inputs = self.tokenizer(prompt, padding="max_length")
55     text_input_ids = text_inputs.input_ids.to(self.device)
56     prompt_embeds = self.text_encoder(text_input_ids)[0]
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.

## Qpipeline.py#L34-L35

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

## Qpipeline.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)
```

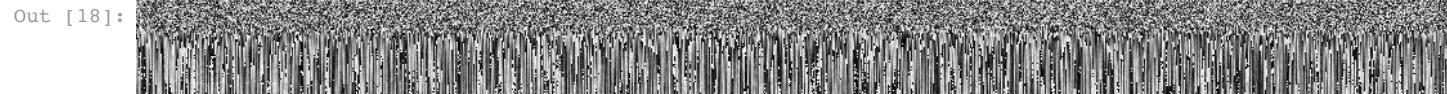
Out [15]:

```
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]])
```

```
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))
```

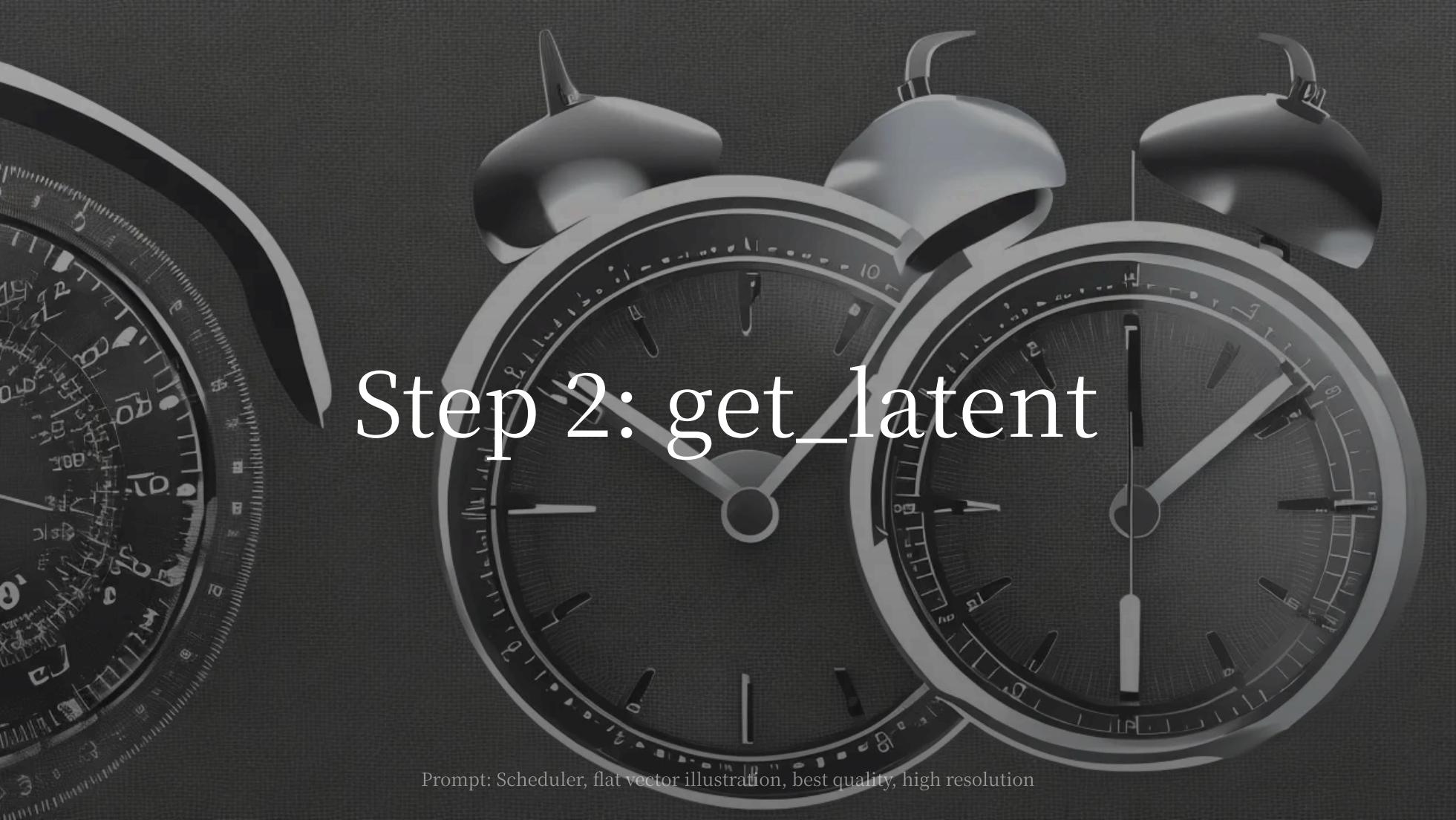
```
Out [18]:  
    torch.Size([1, 77, 1024])  
  
    tensor([[[[-0.3135, -0.4475, -0.0083, ..., 0.2544, -0.0327, -0.2959],  
             [ 0.6030, -0.9849, -0.6772, ..., 1.6953, -0.1359, -0.3333],  
             [-0.1290, -0.3606, -0.1376, ..., 0.5244, -0.8164,  0.8135],  
             ...,  
             [ 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>)
```



## 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-pac  
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) (f  
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 importlit  
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 (f  
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour  
Obtaining file:///br/>ERROR: file:/// does not appear to be a Python project: neither 'setup.py' nor 'pyproject.toml' fo
```



# 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     training.  
63     """  
64     return torch.randn((4, width // 8, height // 8),  
65                        device=self.device, dtype=self.dtype)  
66 
```

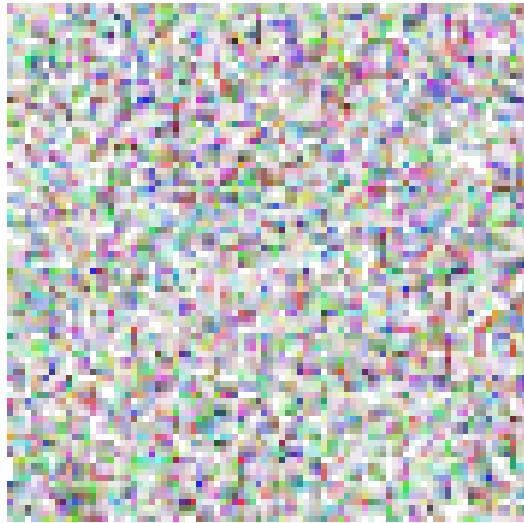
A vibrant watercolor painting of a forest scene. The composition features several large, dark tree trunks and branches in the foreground and middle ground. Behind them, the landscape is filled with a dense array of colorful foliage in shades of red, orange, yellow, green, and blue. The brushwork is visible and expressive, creating a textured and dynamic feel.

# Step 3: denoise

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

Step 3: denoise

Denoising by using Scheduler and UNet.



Index: 0

[Qunderstand-stable-diffusion-slidev-notebooks/denoise.ipynb](#)



Index: 0

[Qunderstand-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

## Qpipeline.py#L75-L93

```
75     @torch.no_grad()
76     def denoise(self, latents, prompt_embeds, num_inference_
77     nnn
78         Iteratively denoise the latent space using the diffi
79     nnn
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

## Qpipeline.py#L21-L39

```
@classmethod
def from_pretrained(cls, model_name, device=torch.device("c:
    # Omit 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,
```

## Qpipeline.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_residu
88         guided_noise_residual = uncond_residual + guidance
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: Denoise 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.

## Qpipeline.py#L82-L93

```
75     @torch.no_grad()
76     def denoise(self, latents, prompt_embeds, num_inference_steps):
77         """
78             Iteratively denoise the latent space using the diffusion
79             model.
80
81             :param latents: torch.FloatTensor of shape (batch_size, num_channels,
82             num_height, num_width) for unconditioned latent
83             :param prompt_embeds: torch.FloatTensor of shape (batch_size, 77, 77)
84             :param num_inference_steps: int. Number of denoising steps. More
85             steps usually mean a better result, but they increase
86             computational cost. Must between 1 and 100.
87             :return: torch.FloatTensor of shape (batch_size, num_channels,
88             num_height, num_width).
89
90             Guidance is provided by text_cond_residual. It is added to
91             the noise_residua before it is passed to the model. This
92             allows for controlling the output of the denoising process.
93
94             Note that latents is modified in place to become the
95             denoised latent representation.
96         """
97         timesteps, num_inference_steps = self.retrieve_timesteps(
98             latents, num_inference_steps)
99
100        for t in reversed(timesteps):
101            latent_model_input = torch.cat([latents] * 2)
102
103            # Predict the noise residual for the current time step
104            noise_residual = self.unet(latent_model_input, t,
105                uncond_residual, text_cond_residual=prompt_embeds)
106            guided_noise_residual = uncond_residual + guidance
107
108            # Update latents by reversing the diffusion process
109            latents = self.scheduler.step(guided_noise_residual,
110                latents)
111
112        return latents
```

# Scheduler.py

Determine the strength  
of denoising

```
1 import json
2 import numpy as np
3 import torch
4 from huggingface_hub import hf_hub_download
5 from .utils import DotDict
6
7
8 class ParedDDIMScheduler:
9     def __init__(self, config_dict: dict):
10         """Initialize beta and alpha values for the scheduler."""
11         self.config = DotDict(**config_dict)
12         self.betas = (
13             torch.linspace(
14                 self.config.beta_start**0.5,
15                 self.config.beta_end**0.5,
16                 self.config.num_train_timesteps,
17                 dtype=torch.float32,
18             )
19             ** 2
20         )
21         self.alphas = 1.0 - self.betas
22         self.alphas_cumprod = torch.cumprod(self.alphas, dim=0)
23         self.final_alpha_cumprod = torch.tensor(1.0)
24
25     @classmethod
26     def from_config(
27         cls,
28         model_name: str,
29         subfolder: str = "scheduler",
30         filename: str = "scheduler_config.json",
31     ) -> "ParedDDIMScheduler":
32         """Create scheduler instance from configuration file."""
33         config_file = hf_hub_download(
34             model_name, filename=filename, subfolder=subfolder
35         )
36         with open(config_file, "r", encoding="utf-8") as reader:
37             text = reader.read()
38         config_dict = json.loads(text)
39         return cls(config_dict)
40
41     def set_timesteps(
42         self, num_inference_steps: int, device: torch.device = None
43     ) -> None:
44         """Set the timesteps for the scheduler based on the number of inference steps."""
45         self.num_inference_steps = num_inference_steps
46         step_ratio = self.config.num_train_timesteps // self.num_inference_steps
47         timesteps = (
48             (np.arange(0, num_inference_steps) * step_ratio)
49             .round()[:-1]
50             .copy()
51             .astype(np.int64)
52         )
53         timesteps += self.config.steps_offset
54         self.timesteps = torch.from_numpy(timesteps).to(device)
55
56     def step(
57         self,
58         model_output: torch.FloatTensor,
59         timestep: int,
60         sample: torch.FloatTensor,
61     ) -> list:
62         """Perform a single step of denoising in the diffusion process."""
63         new_timestep = t
```

# Scheduler

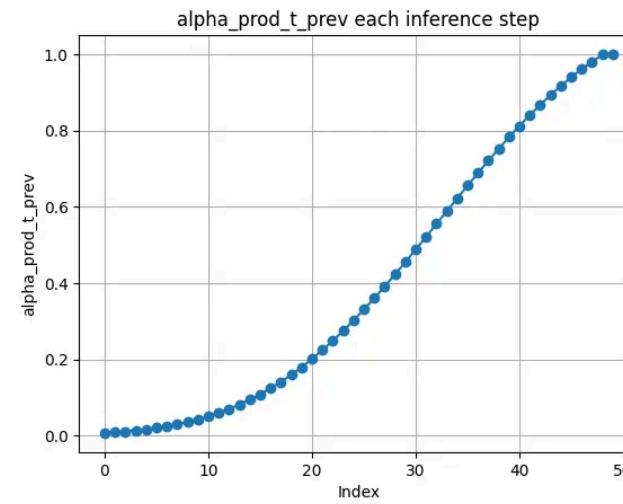
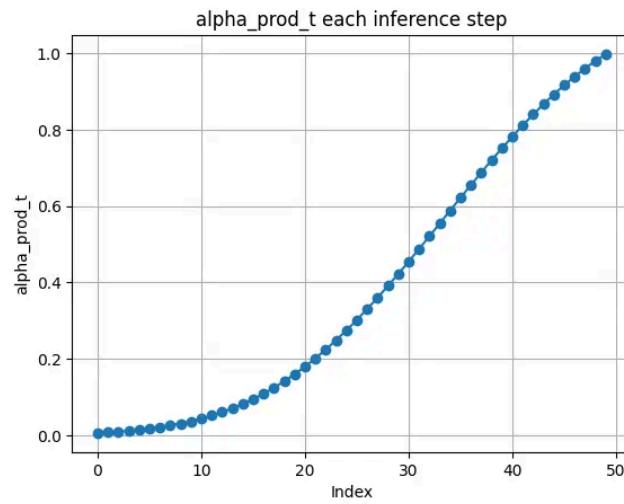
- L49: Get alpha\_prod\_t(0~1.0)  
(Indicates how much the original data is retained.)
- L50: Get alpha\_prod\_t\_prev(0~1.0)
- L52:  $\alpha_{prod\_t} + \beta_{prod\_t} = 1$
- L53: Estimate the original sample from the current sample and model output.
- L54: Calculate the estimated value of the added 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 process.
47         prev_timestep = timestep - self.config.num_train_timesteps
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 - (beta
54         pred_epsilon = (alpha_prod_t**0.5) * model_output + (beta
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
```

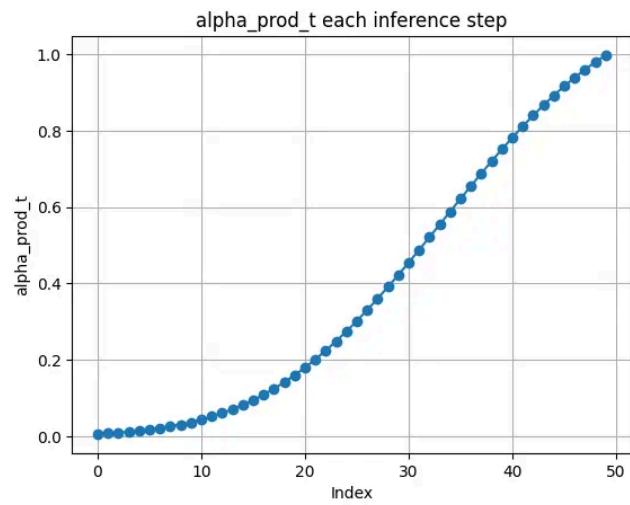
## Scheduler.py#L40-L59

```
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 process."""
47         prev_timestep = timestep - self.config.num_train_timesteps // self.num_inference_steps
48
49         alpha_prod_t = self.alphas_cumprod[timestep]
50         alpha_prod_t_prev = self.alphas_cumprod[prev_timestep] if prev_timestep >= 0 else self.final_alpha_cumprod
51
52         beta_prod_t = 1 - alpha_prod_t
53         pred_original_sample = (alpha_prod_t**0.5) * sample - (beta_prod_t**0.5) * model_output
54         pred_epsilon = (alpha_prod_t**0.5) * model_output + (beta_prod_t**0.5) * sample
55
56         pred_sample_direction = (1 - alpha_prod_t_prev)**(0.5) * pred_epsilon
57         prev_sample = alpha_prod_t_prev**(0.5) * pred_original_sample + pred_sample_direction
58
59         return prev_sample, pred_original_sample
```

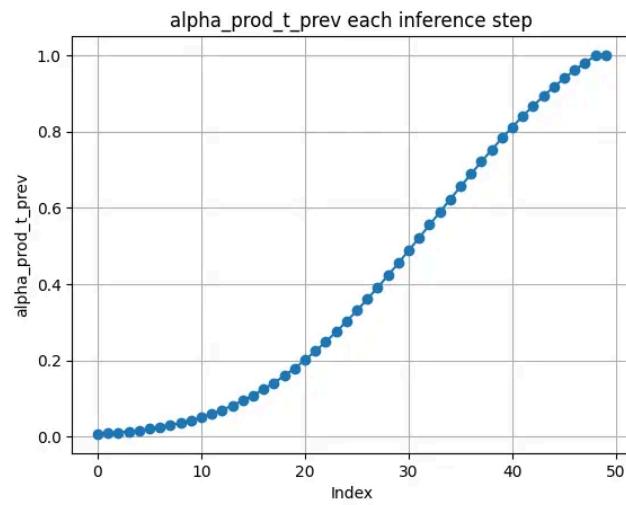


 understand-stable-diffusion-slidev-notebooks/scheduler.ipynb

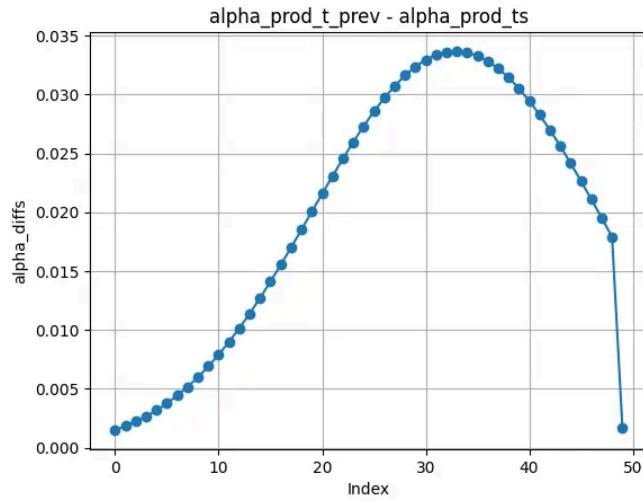
-



+



[Qunderstand-stable-diffusion-slidev-notebooks/scheduler.ipynb](#)



[Qunderstand-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=pipe.device)
        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 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)

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, dtype

```



```

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

```



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



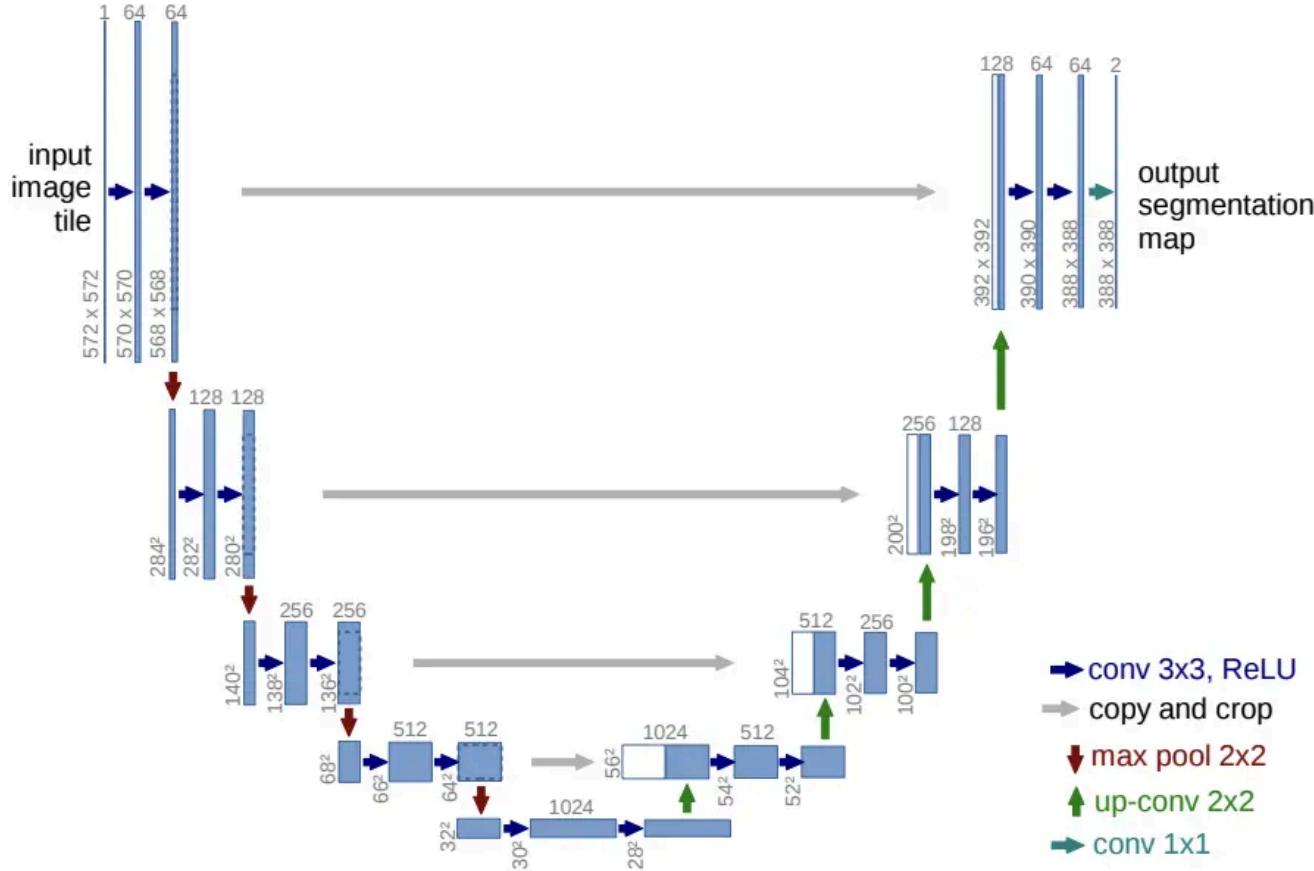
0.5000

[Qunderstand-stable-diffusion-slidev-notebooks/scheduler\\_necessity.ipynb](#)



## Using for Denoising

```
1 import torch
2 from torch import nn
3 from typing import List, Union
4 import json
5 from huggingface_hub import hf_hub_download
6 from .utils import DotDict, get_activation
7 from .defaults import DEFAULT_UNET_CONFIG
8 from .models.embeddings import PareTimestepEmbedding, PareTimesteps
9 from .models.unet_2d_get_blocks import pare_get_down_block, pare_get_up_block
10 from .models.unet_2d_mid_blocks import PareUNetMidBlock2DCrossAttn
11
12
13 class PareUNet2DConditionModel(nn.Module):
14     def __init__(self, **kwargs):
15         super().__init__()
16         self.config = DotDict(DEFAULT_UNET_CONFIG)
17         self.config.update(kwargs)
18         self.config.only_cross_attention = [self.config.only_cross_attention] * len(
19             self.config.down_block_types
20         )
21         self.config.num_attention_heads = (
22             self.config.num_attention_heads or self.config.attention_head_dim
23         )
24         self._setup_model_parameters()
25
26         self._build_input_layers()
27         self._build_time_embedding()
28         self._build_down_blocks()
29         self._build_mid_block()
30         self._build_up_blocks()
31         self._build_output_layers()
32
33     def _setup_model_parameters(self) -> None:
34         if isinstance(self.config.num_attention_heads, int):
35             self.config.num_attention_heads = (self.config.num_attention_heads,) * len(
36                 self.config.down_block_types
37             )
38         if isinstance(self.config.attention_head_dim, int):
39             self.config.attention_head_dim = (self.config.attention_head_dim,) * len(
40                 self.config.down_block_types
41             )
42         if isinstance(self.config.cross_attention_dim, int):
43             self.config.cross_attention_dim = (self.config.cross_attention_dim,) * len(
44                 self.config.down_block_types
45             )
46         if isinstance(self.config.layers_per_block, int):
47             self.config.layers_per_block = [self.config.layers_per_block] * len(
48                 self.config.down_block_types
49             )
50         if isinstance(self.config.transformer_layers_per_block, int):
51             self.config.transformer_layers_per_block = [
52                 self.config.transformer_layers_per_block
53             ] * len(self.config.down_block_types)
54
55     def _build_input_layers(self) -> None:
56         conv_in_padding = (self.config.conv_in_kernel - 1) // 2
57         self.conv_in = nn.Conv2d(
58             self.config.in_channels,
59             self.config.block_out_channels[0],
60             kernel_size=self.config.conv_in_kernel,
61             padding=conv_in_padding,
62         )
63
```



# Create UNet using Resnet and Transformer

```
114     for i in range(num_layers):
115         in_channels = in_channels if i == 0 else out_channels
116         resnets.append(
117             PareResnetBlock2D(
118                 in_channels=in_channels,
119                 out_channels=out_channels,
120                 temb_channels=temb_channels,
121                 eps=resnet_eps,
122                 groups=resnet_groups,
123                 dropout=dropout,
124                 non_linearity=resnet_act_fn,
125                 output_scale_factor=output_scale_factor,
126             )
127         )
128     if not dual_cross_attention:
129         attentions.append(
130             PareTransformer2DModel(
131                 num_attention_heads,
132                 out_channels // num_attention_heads,
133                 in_channels=out_channels,
134                 num_layers=transformer_layers_per_block[i],
135                 cross_attention_dim=cross_attention_dim,
136                 norm_num_groups=resnet_groups,
137                 use_linear_projection=use_linear_projection,
138                 only_cross_attention=only_cross_attention,
139                 upcast_attention=upcast_attention,
140             )
141         )
```

The background of the image is a vibrant, abstract pattern resembling stained glass or a geometric tessellation. It features a variety of colors including red, blue, green, yellow, purple, and orange, all enclosed within numerous thin, black-outlined triangles and quadrilaterals. The overall effect is organic and fluid, despite the geometric nature of the shapes.

# 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

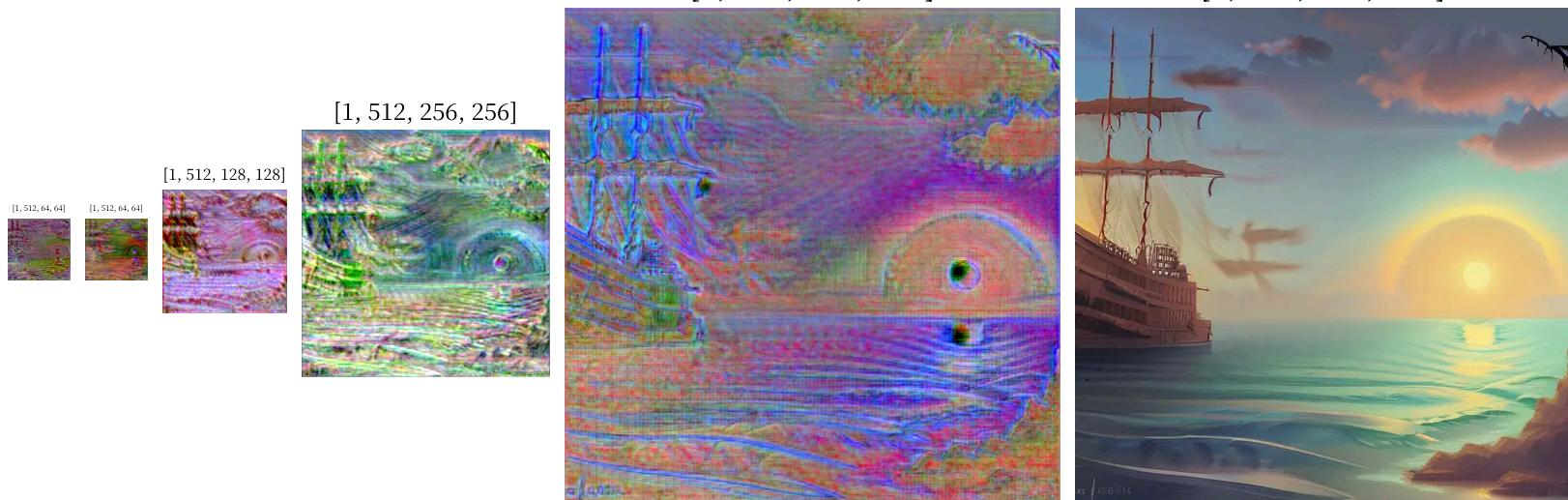
[pipeline.py#L107-L105](#)

```
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_factor)
113         image = self.denormalize(image)
114         image = self.tensor_to_image(image)
115         return image
```

# Variational Autoencoder (VAE)

# vae.py

```
1 import json
2 from typing import List
3 import torch
4 import torch.nn as nn
5 from huggingface_hub import hf_hub_download
6 from .utils import DotDict
7 from .defaults import DEFAULT_VAE_CONFIG
8 from .models.vae_blocks import (
9     PareEncoder,
10    PareDecoder,
11    PareDiagonalGaussianDistribution,
12 )
13
14
15 class PareAutoencoderKL(nn.Module):
16     def __init__(self, **kwargs):
17         super().__init__()
18         self.config = DotDict(DEFAULT_VAE_CONFIG)
19         self.config.update(kwargs)
20
21         # pass init params to Encoder
22         self.encoder = PareEncoder(
23             in_channels=self.config.in_channels,
24             out_channels=self.config.latent_channels,
25             down_block_types=self.config.down_block_types,
26             block_out_channels=self.config.block_out_channels,
27             layers_per_block=self.config.layers_per_block,
28         )
29
30         # pass init params to Decoder
31         self.decoder = PareDecoder(
32             in_channels=self.config.latent_channels,
33             out_channels=self.config.out_channels,
34             up_block_types=self.config.up_block_types,
35             block_out_channels=self.config.block_out_channels,
36             layers_per_block=self.config.layers_per_block,
37         )
38
39         self.quant_conv = nn.Conv2d(
40             2 * self.config.latent_channels, 2 * self.config.latent_channels, 1
41         )
42         self.post_quant_conv = nn.Conv2d(
43             self.config.latent_channels, self.config.latent_channels, 1
44         )
45
46         self.use_slicing = False
47         self.use_tiling = False
48
49         # only relevant if vae tiling is enabled
50         self.tile_sample_min_size = self.config.sample_size
51         sample_size = (
52             self.config.sample_size[0]
53             if isinstance(self.config.sample_size, (list, tuple))
54             else self.config.sample_size
55         )
56         self.tile_latent_min_size = int(
57             sample_size / (2 ** (len(self.config.block_out_channels) - 1))
58         )
59         self.tile_overlap_factor = 0.25
60
61     @classmethod
62     def _get_config(
63         cls, model_name: str, filename: str = "config.json", subfolder: str = "unet"
64     ):
65         return hf_hub_download(model_name, filename, subfolder=subfolder)
```



[Q](#)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

The screenshot shows a code editor interface with the following details:

- Header:** Shows the path: diffusers / src / diffusers / pipelines / stable\_diffusion / pipeline\_stable\_diffusion.py.
- Toolbar:** Includes buttons for Code (selected), Blame, Raw, and various file operations.
- Search Dialog:** A modal window titled "Find" is open, with the instruction "Press ⌘ f again to open the browser's find menu". It contains a search input field with the text "arxiv", a page number "1/6", and navigation arrows.
- Code Content:** The code itself is a Python function named `def rescale_noise_cfg`. The code includes a note about rescaling noise configurations based on findings from a paper available at <https://arxiv.org/pdf/2305.08891.pdf>.

```
54         >>> image = pipe(prompt).images[0]
55         ...
56     """
57
58
59     def rescale_noise_cfg(noise_cfg, noise_pred_text, guidance_rescale=0.0):
60         """
61             Rescale `noise_cfg` according to `guidance_rescale`. Based on findings of [Common Diffusion Noise
62             Schedules and
63             Sample Steps are Flawed](https://arxiv.org/pdf/2305.08891.pdf). See Section 3.4
64
65             std_text = noise_pred_text.std(dim=list(range(1, noise_pred_text.ndim)), keepdim=True)
66             std_cfg = noise_cfg.std(dim=list(range(1, noise_cfg.ndim)), keepdim=True)
67             # rescale the results from guidance (fixes overexposure)
68             noise_pred_rescaled = noise_cfg * (std_text / std_cfg)
69             # mix with the original results from guidance by factor guidance_rescale to avoid "plain looking" images
70             noise_cfg = guidance_rescale * noise_pred_rescaled + (1 - guidance_rescale) * noise_cfg
71
72     return noise_cfg
```

## 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

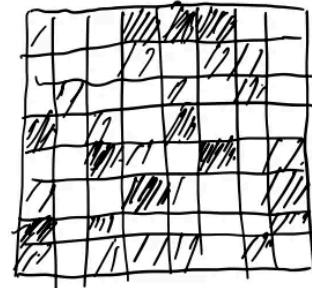
## pipeline.py#L117-L135

```
1 def __call__(self, prompt: str, height: int = 512, width: int = 512, ...):
2     prompt_embeds = self.encode_prompt(prompt)
3     latents = self.get_latent(width, height).unsqueeze(dim=0)
4     latents = self.denoise(latents, prompt_embeds, ...)
5     image = self.vae_decode(latents)
6     return image
```

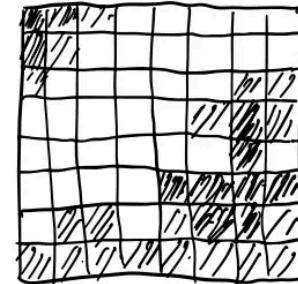
encode\_prompt

$\begin{bmatrix} 0.13, 0.17, 0.79, \dots \\ 0.0, -0.1, -0.1 \\ 0.1, -0.1, 0.1 \\ 0.0, 0.0, 0.0 \\ 0.1, -0.1, 0.1 \\ 0.0, 0.0, 0.0 \\ 0.1, 0.1, 0.1 \end{bmatrix}$

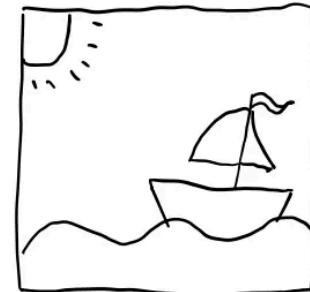
get\_latent



denoise



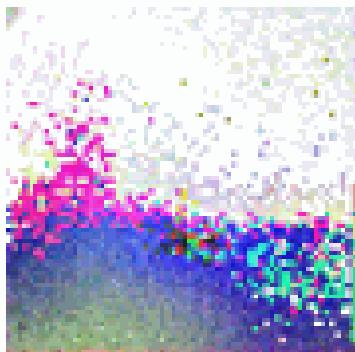
vae\_decode



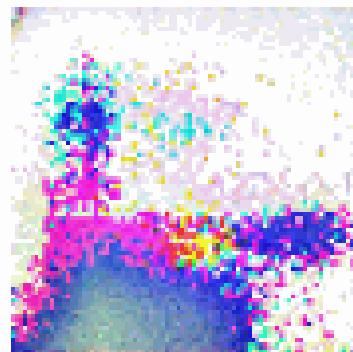
# Appendix

1. Appendix
  1. Other denoising samples
  2. Other decoded denoising samples
  3. UNet is really U form?

# Other denoising samples



Index: 0



Index: 0

[Qunderstand-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?

```
1  init          torch.Size([2, 4, 64, 64])
2  conv_in       torch.Size([2, 320, 64, 64])
3
4  down_blocks_0 torch.Size([2, 320, 32, 32])
5  down_blocks_1 torch.Size([2, 640, 16, 16])
6  down_blocks_2 torch.Size([2, 1280, 8, 8])
7  down_blocks_3 torch.Size([2, 1280, 8, 8])
8
9  mid_block    torch.Size([2, 1280, 8, 8])
10
11 up_blocks0   torch.Size([2, 1280, 16, 16])
12 up_blocks1   torch.Size([2, 1280, 32, 32])
13 up_blocks2   torch.Size([2, 640, 64, 64])
14 up_blocks3   torch.Size([2, 320, 64, 64])
15
16 conv_out     torch.Size([2, 4, 64, 64])
```

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