Sketch-to-Image

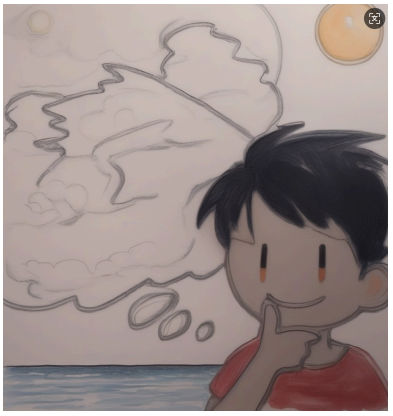
Some models recording:

1. **ControlNet × (ineffective)**
2. **FluxFill × (Need Mask Image)**
3. **Qwen3 √ (effective but need large memory — maybe use web interface..)**

```
import os
from PIL import Image
import torch

from diffusers import QwenImageEditPipeline

pipeline = QwenImageEditPipeline.from_pretrained("Qwen/Qwen-Image-Ed
it")
print("pipeline loaded")
pipeline.to(torch.bfloat16)
pipeline.to("cuda")
pipeline.set_progress_bar_config(disable=None)
image = Image.open("sketch.png").convert("RGB")
prompt = "Change the sketch into realistic image."
inputs = {
    "image": image,
    "prompt": prompt,
    "generator": torch.manual_seed(0),
```

```
    "true_cfg_scale": 4.0,
    "negative_prompt": " ",
    "num_inference_steps": 50,
}

with torch.inference_mode():
    output = pipeline(**inputs)
    output_image = output.images[0]
    output_image.save("output_image_edit.png")
    print("image saved at", os.path.abspath("output_image_edit.png"))
```

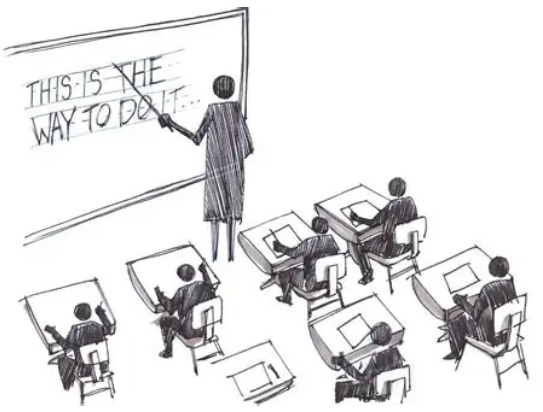
4. **T2I-Adapter √ (can’t transform into real photo but can make pic looks more realistic..)**



```
import os
from diffusers import StableDiffusionXLAdapterPipeline, T2IAdapter, EulerA
ncestralDiscreteScheduler, AutoencoderKL
from diffusers.utils import load_image, make_image_grid
from controlnet_aux.pidi import PidiNetDetector
import torch

# load adapter
adapter = T2IAdapter.from_pretrained(
  "TencentARC/t2i-adapter-sketch-sdxl-1.0", torch_dtype=torch.float16, vari
ent="fp16"
).to("cuda")

# load euler_a scheduler
```

```
model_id = 'stabilityai/stable-diffusion-xl-base-1.0'
euler_a = EulerAncestralDiscreteScheduler.from_pretrained(model_id, subf
older="scheduler")
vae=AutoencoderKL.from_pretrained("madebyollin/sdxl-vae-fp16-fix", torc
h_dtype=torch.float16)
pipe = StableDiffusionXLAdapterPipeline.from_pretrained(
    model_id, vae=vae, adapter=adapter, scheduler=euler_a, torch_dtype=to
rch.float16, variant="fp16",
).to("cuda")
pipe.enable_xformers_memory_efficient_attention()

pidinet = PidiNetDetector.from_pretrained("lllyasviel/Annotators").to("cud
a")

# Image
url = "sketch_cartoon.png"
image = load_image(url)
image = pidinet(
  image, detect_resolution=1024, image_resolution=1024, apply_filter=True
)

# generation
prompt = "A boy is thinking his holiday"
negative_prompt = "extra digit, fewer digits, cropped, worst quality, low qu
ality, glitch, deformed, mutated, ugly, disfigured"

gen_images = pipe(
    prompt=prompt,
    negative_prompt=negative_prompt,
    image=image,
    num_inference_steps=30,
    adapter_conditioning_scale=0.9,
    guidance_scale=7.5,
).images[0]
gen_images.save('out_sketch.png')
```

5. **Nano-Banana √ (Best Performance)**




```
import requests
import json
import re
```

```
import base64
import os
import mimetypes
from datetime import datetime

def load_local_image(image_path):
    """read local image and convert to base64"""
    try:
        with open(image_path, 'rb') as f:
            image_data = f.read()

        # convert to base64
        img_base64 = base64.b64encode(image_data).decode('utf-8')

        # acquire picture's mime type
        mime_type, _ = mimetypes.guess_type(image_path)
        if mime_type is None:
            mime_type = 'image/png'  # default to png

        return img_base64, mime_type
    except Exception as e:
        print(f"Read Image {image_path} Fails: {e}")
        return None, None

def save_base64_image(base64_data, filename=None):
    """write base64 image data to a file"""
    if filename is None:
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        # filename = f"generated_image_{timestamp}.png"
        filename = f"generated_image.png"

    try:
        # decode base64 data
        image_data = base64.b64decode(base64_data)

        # write to file
        with open(filename, 'wb') as f:
            f.write(image_data)
```

```
        print(f"\n📸 Image has been writen to {filename}")
        print(f"📂 Full Path: {os.path.abspath(filename)}")

        return filename
    except Exception as e:
        print(f"\n❌ Image saved failed: {e}")
        return None

def extract_base64_from_markdown(text):
    """extract base64 data from markdown image format"""
    # match ![image](...) format
    pattern = r'!\[image\]\(([A-Za-z0-9+/=]+)\)'
    match = re.search(pattern, text)
    if match:
        return match.group(1)
    return None

def chat():
    api_url = "https://new.12ai.org"
    # set API_KEY
    api_key = "sk-TD18zZ2DIvt0mYDfRpxXXinFHUCviiUehkiaEHRjYYgOTt1i"
    model = "gemini-2.5-flash-image-preview"

    # English Prompt
    user_question = "Please help me transform this comic picture into a real-
life scene picture."

    # Image List
    images = ["/hdd0/zl/EchoInk_New/Imgae_2_Image/sketch_class.png"]

    # Construct request payload
    parts = [{"text": user_question}]
    for image_path in images:
        img_base64, mime_type = load_local_image(image_path)
        parts.append({
            "inline_data": {
                "mime_type": mime_type,
```

```
            "data": img_base64
        }
    })

data = {
    "contents": [{"parts": parts}],
    "generationConfig": {
        "temperature": 0.7,
        "maxOutputTokens": 4096,
        "topP": 1.0
    },
    "stream": False
}

try:
    response = requests.post(
        f"{api_url}/v1beta/models/{model}:generateContent?key={api_key}",
        headers={"Content-Type": "application/json"},
        json=data,
        timeout=30,
        verify=False
    )

    if response.status_code != 200:
        print(f"Error: {response.status_code}")
        print(response.text[:1000])  # truncate long output
        return

    result = response.json()
    print(f"Question: {user_question}")
    print("Response:")
    print("-" * 50)

    # Simply print the text part of the response
    for candidate in result.get('candidates', []):
        for part in candidate.get('content', {}).get('parts', []):
            if 'text' in part:
                text_content = part['text']
```

```
                print(text_content[:2000])  # Truncate original output

                # check and save base64 image if exists
                base64_data = extract_base64_from_markdown(text_content)
                if base64_data:
                    save_base64_image(base64_data)

    except requests.exceptions.RequestException as e:
        print(f"Error in request: {e}")
    except json.JSONDecodeError as e:
        print(f"JSON analysis error: {e}")

if __name__ == "__main__":
    chat()
```