AI VIET NAM
@aivietnam.edu.vn

Emoji Image Generation

Project

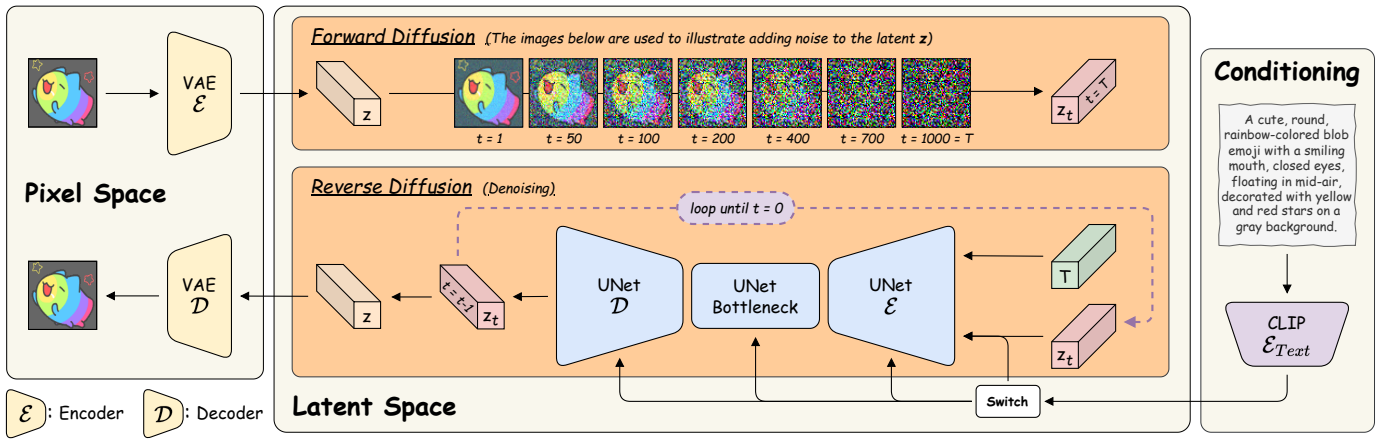
Dinh-Thang Duong – TA

Yen-Linh Vu – STA

Anh-Khoi Nguyen – STA

Getting Started

❖ Content



Our objectives:

- Investigate the task of Image Generation (IG) and Stable Diffusion (SD).
- Implement a Python program to crawl Discord Emojis on website.
- Implement and training a SD model from scratch using PyTorch and Diffusers.
- Implement and training a SD model from a pre-trained model using Diffusers.

Outline

- Introduction
- Data Collection and Preparation
- Training SD from scratch
- Training SD from pre-trained models
- Question

Introduction

Introduction

❖ Getting Started

Forbes

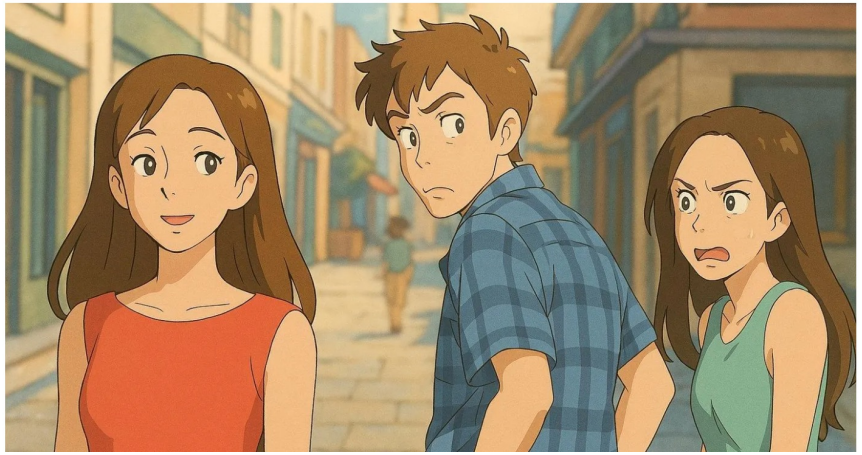
The AI-Generated Studio Ghibli Trend, Explained

OpenAI's new image generator has released a surge of Studio Ghibli inspired images that many believe go against the philosophy of the legendary animation studio and director Hayao Miyazaki.

By Dani Di Placido, Senior Contributor. Dani Di Placido covers film, televisio... Follow Author

Published Mar 27, 2025, 10:31am EDT, Updated Mar 31, 2025, 12:41pm EDT

Share Save Comment 6



Create image apply studio ghibli style to this image

Image created

Example: An image generation trend with Studio Ghibli generated image.

Ask anything

Search Deep research

https://www.forbes.com/sites/danidiplacido/2025/03/27/the-ai-generated-studio-ghibli-trend-explained/

Introduction

❖ Introduction to the task of Image Generation

(a) Text-to-image generation MirrorGAN [55]

(b) Scene-graph-to-image generation [26]

(c) Sketch-to-image generation EdgeGAN [97]

(d) Layout-to-image generation OC-GAN [74]

(e) Pose-Guided-image generation MsCGAN [94]

(f) Image-to-image Translation C2GAN [39]

(g) Face generation [62]

(h) Image generation PNAPGAN [89]

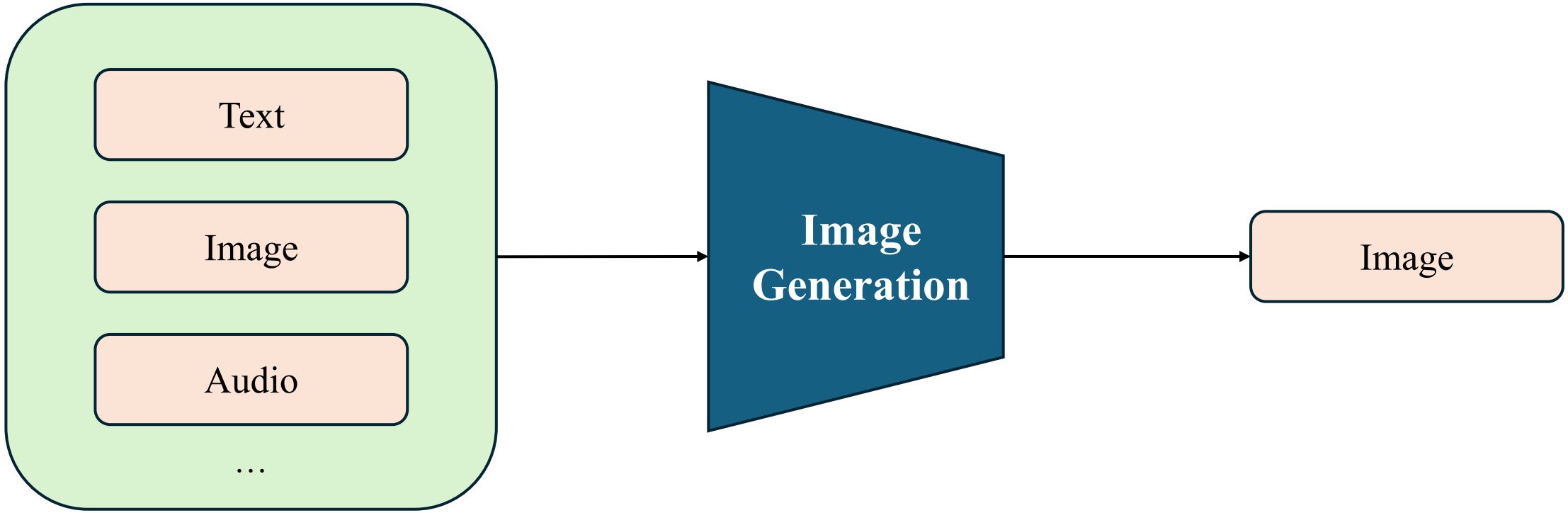
Image generation is the process of synthesizing new images from given inputs or learned representations.

It aims to produce visually coherent, contextually relevant, and high-quality outputs aligned with the intended goal.

https://link.springer.com/article/10.1007/s11063-022-10777-x

Introduction

❖ Examples of Image Generation



Introduction

❖ Image Generation applications

https://heartbeat.comet.ml/super-resolution-580424fb7f7c
https://medium.com/data-science-at-microsoft/introduction-to-image-inpainting-with-a-practical-example-from-the-e-commerce-industry-f81ae6635d5e
https://huggingface.co/docs/hub/models-widgets

Introduction

❖ Introduction to text2img

Sprouts in the shape of text 'Imagen' coming out of a fairytale book.

A photo of a Shiba Inu dog with a backpack riding a bike. It is wearing sunglasses and a beach hat.

A high contrast portrait of a very happy fuzzy panda dressed as a chef in a high end kitchen making dough. There is a painting of flowers on the wall behind him.

Text-to-image generation refers to the task of automatically producing images that are semantically aligned with a given textual description or prompt.

By processing the linguistic content of the prompt and mapping it to visual features, models in this domain transform input text into synthesized images that reflect the conceptual meaning conveyed by the words.

https://arxiv.org/abs/2205.11487

Introduction

❖ Text2img challenges

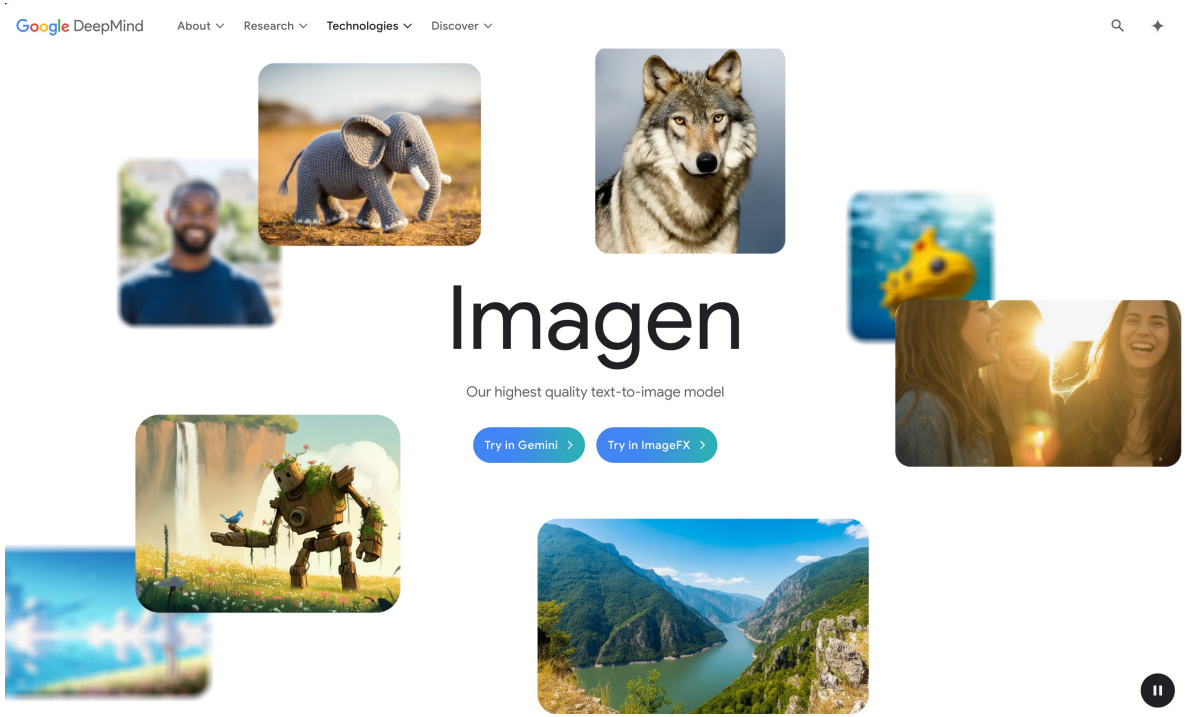
- **Language Nuance**: Interpreting subtle, implied, or context-dependent details from textual descriptions.
- **Visual Accuracy**: Rendering accurate shapes, colors, and spatial relationships aligned with the prompt.
- **Realism and Style**: Achieving photorealism or adhering to a specific artistic style.

https://mmagic.readthedocs.io/en/stable/model_zoo/text2image.html

AI VIET NAM
@aivietnam.edu.vn

Introduction

❖ Text2img popular applications

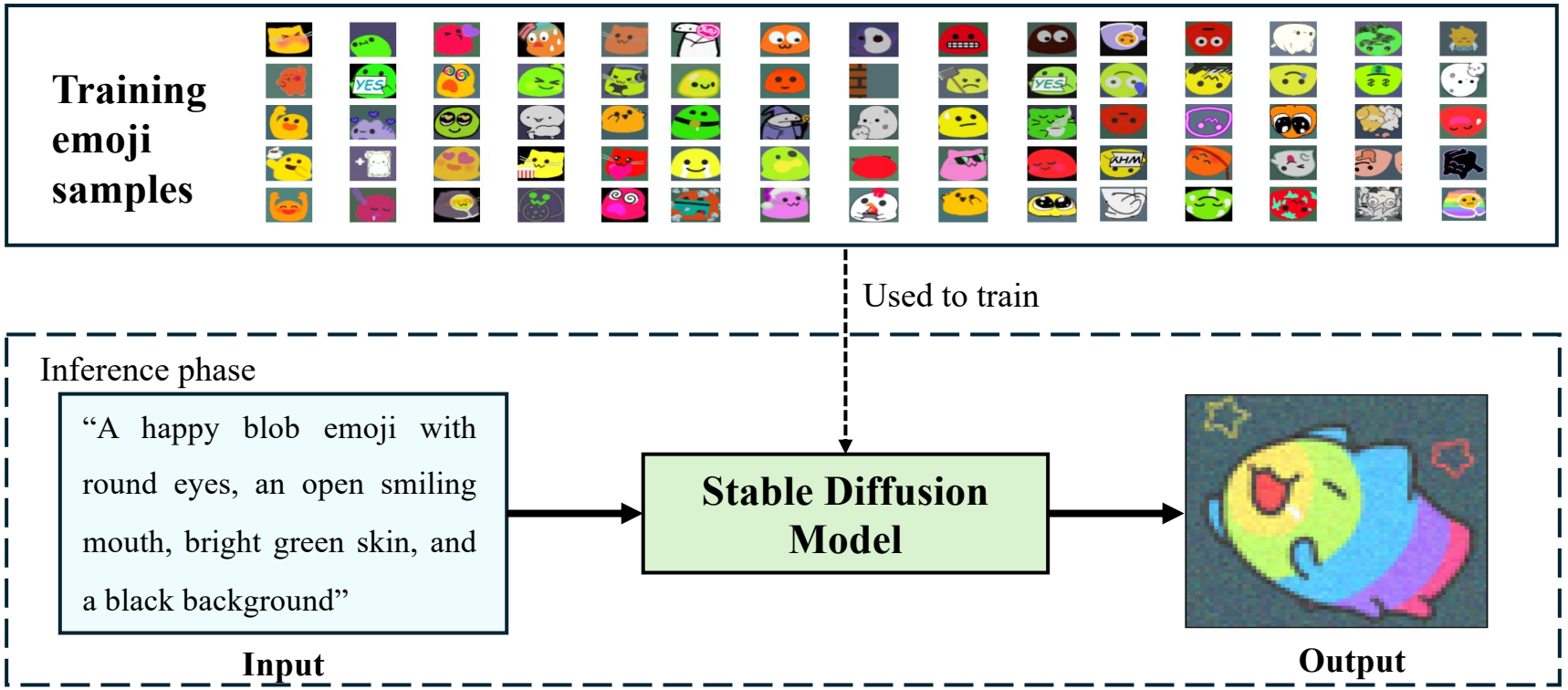


AI VIET NAM
@aivietnam.edu.vn

Training SD from scratch

❖ Problem Statement

Problem Statement: Given a prompt that describes how the emoji looks like, generate a new emoji image using Stable Diffusion model.



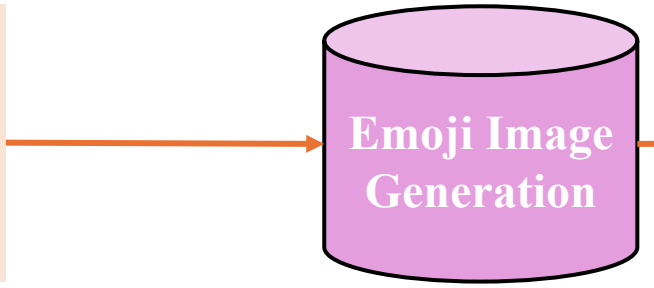
Introduction

❖ Description

Problem Statement: Given a prompt that describes how the emoji looks like, generate a new emoji image using Stable Diffusion model.

Input

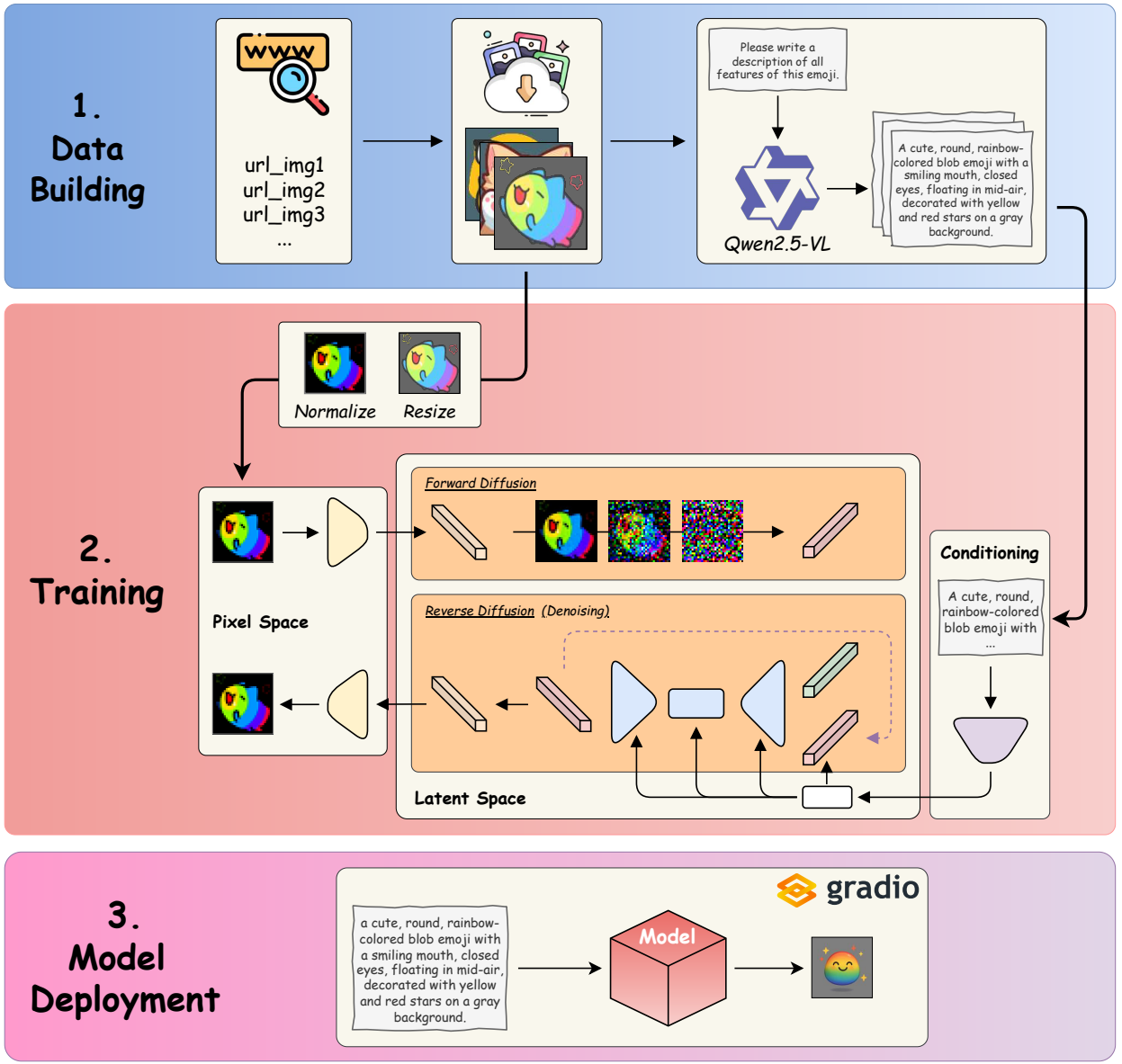
"Blue cat emoji with a round head, blushing cheeks, puffed face, angry expression, tiny clenched fists raised beside the head, steam puffs above both ears, no background."



Output

Introduction

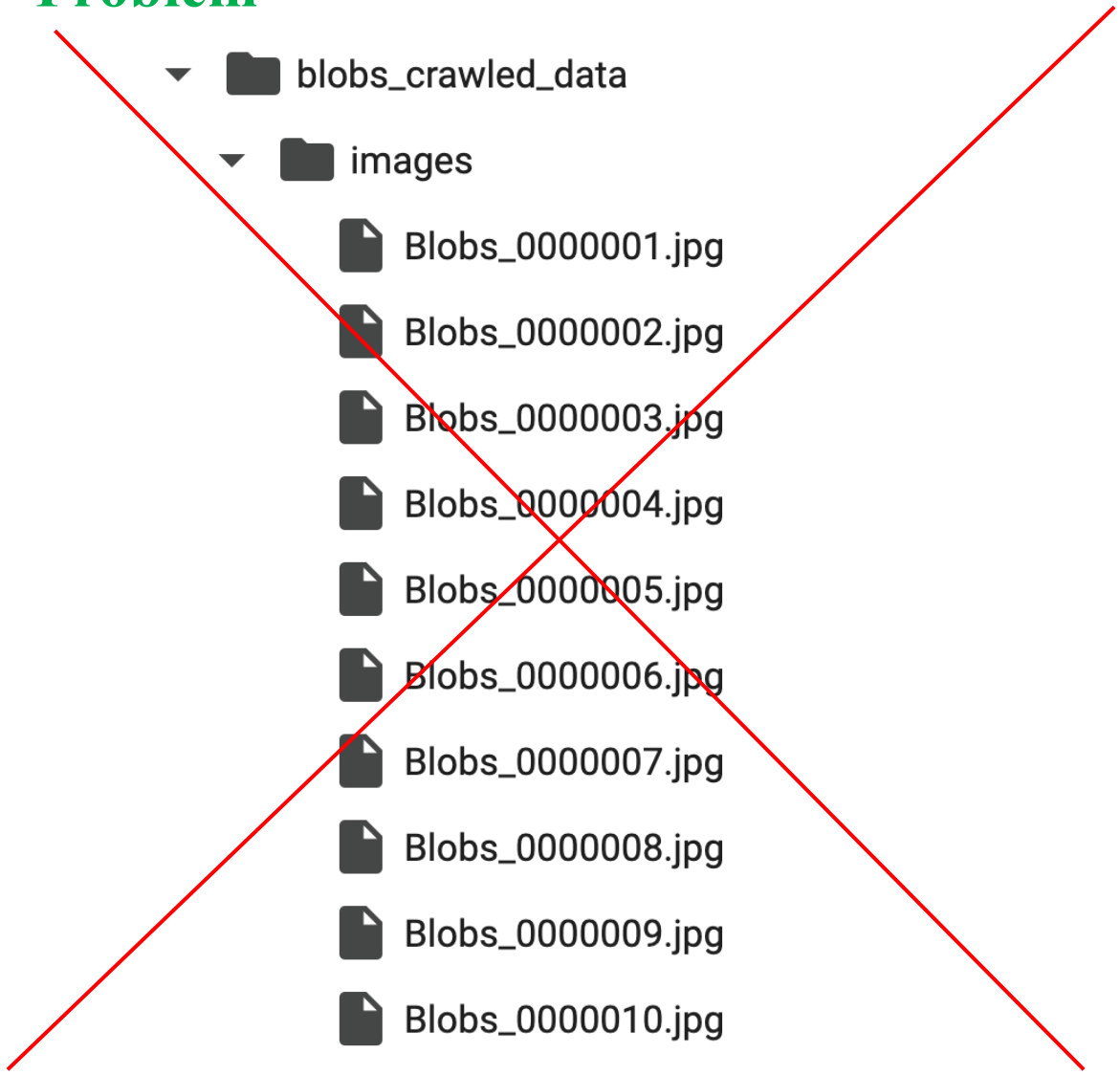
❖ Project Pipeline

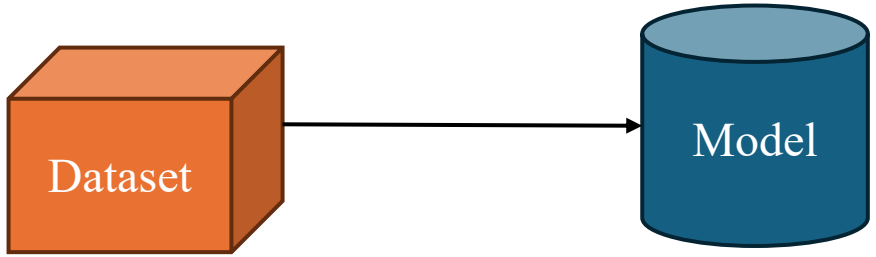


Data Collection and Preparation

Data Collection and Preparation

❖ Problem

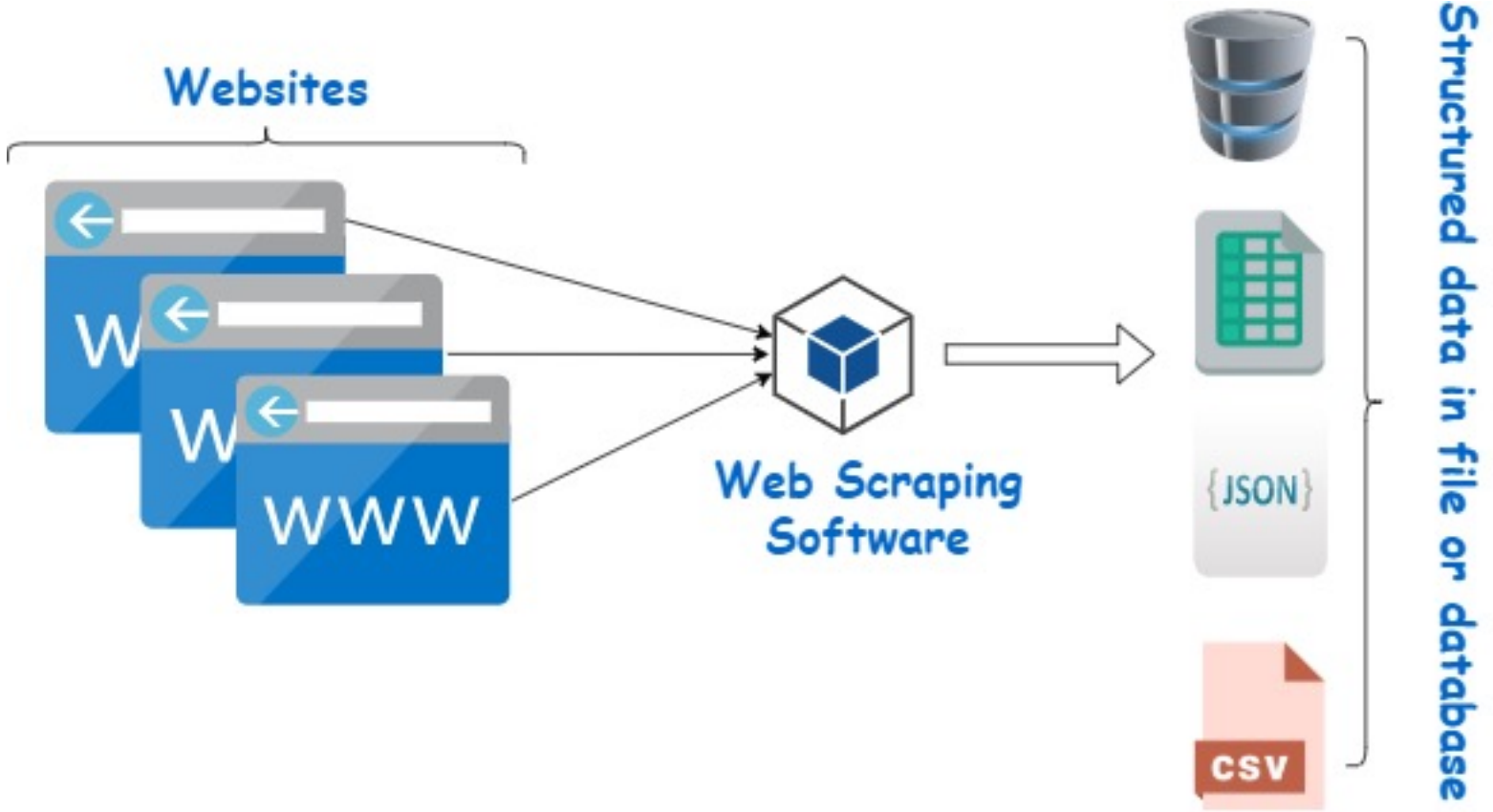




We don’t have data yet.

Data Collection and Preparation

❖ Solution: Data Scraping

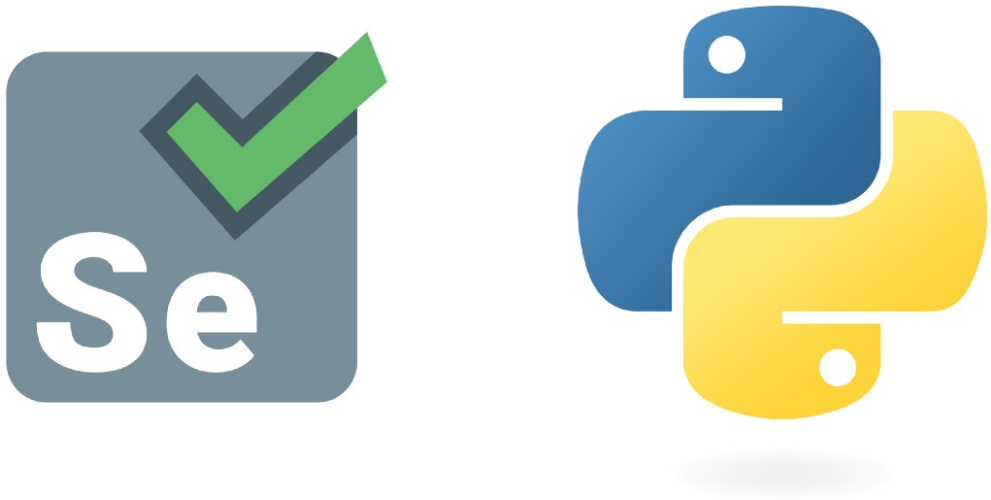


We can get data from websites.

https://www.webharvy.com/articles/what-is-web-scraping.html

Data Collection and Preparation

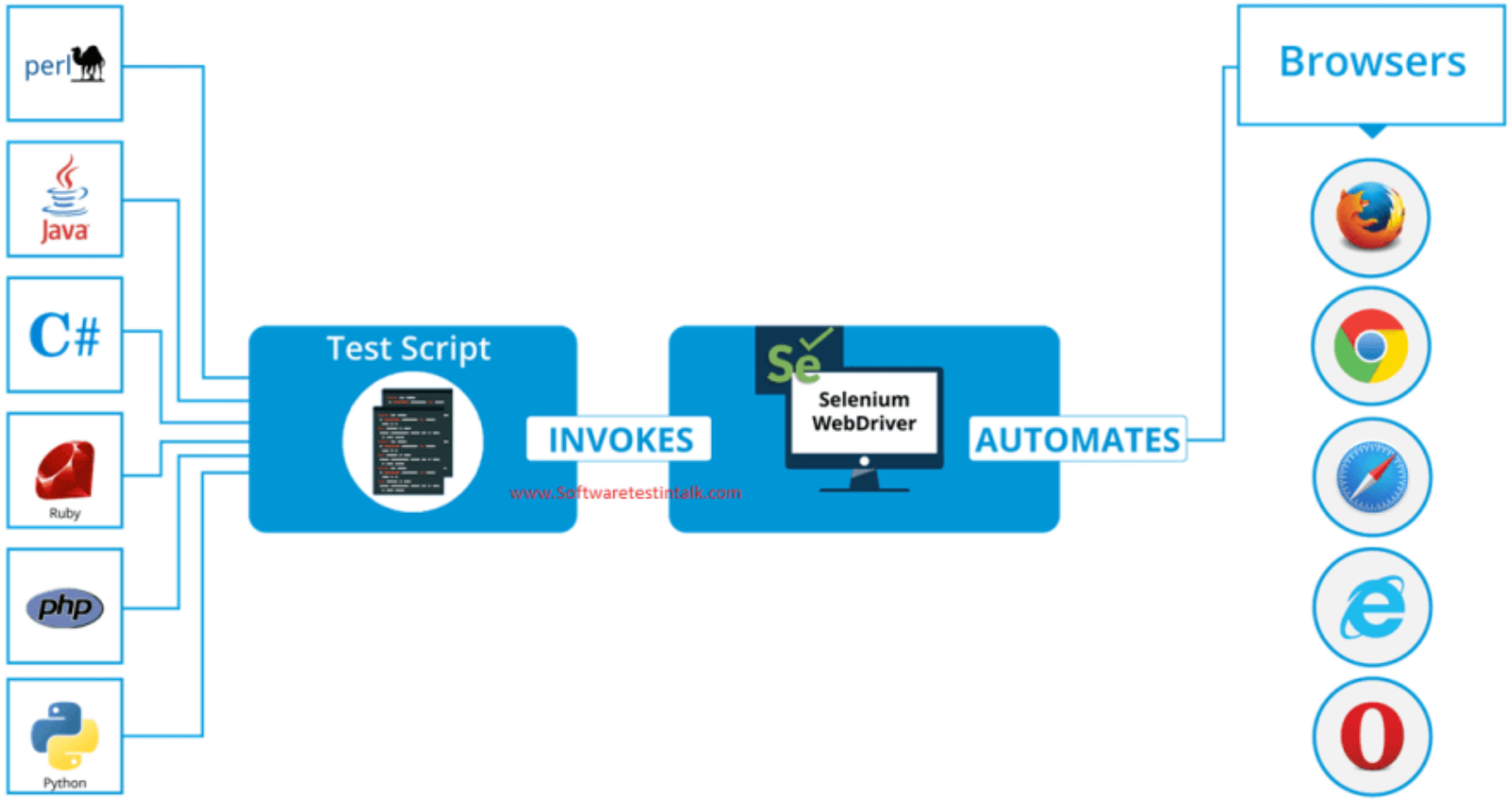
❖ Selenium Package



Selenium Package: Use to automate web browser interaction from Python.

Data Collection and Preparation

❖ Introduction to Selenium

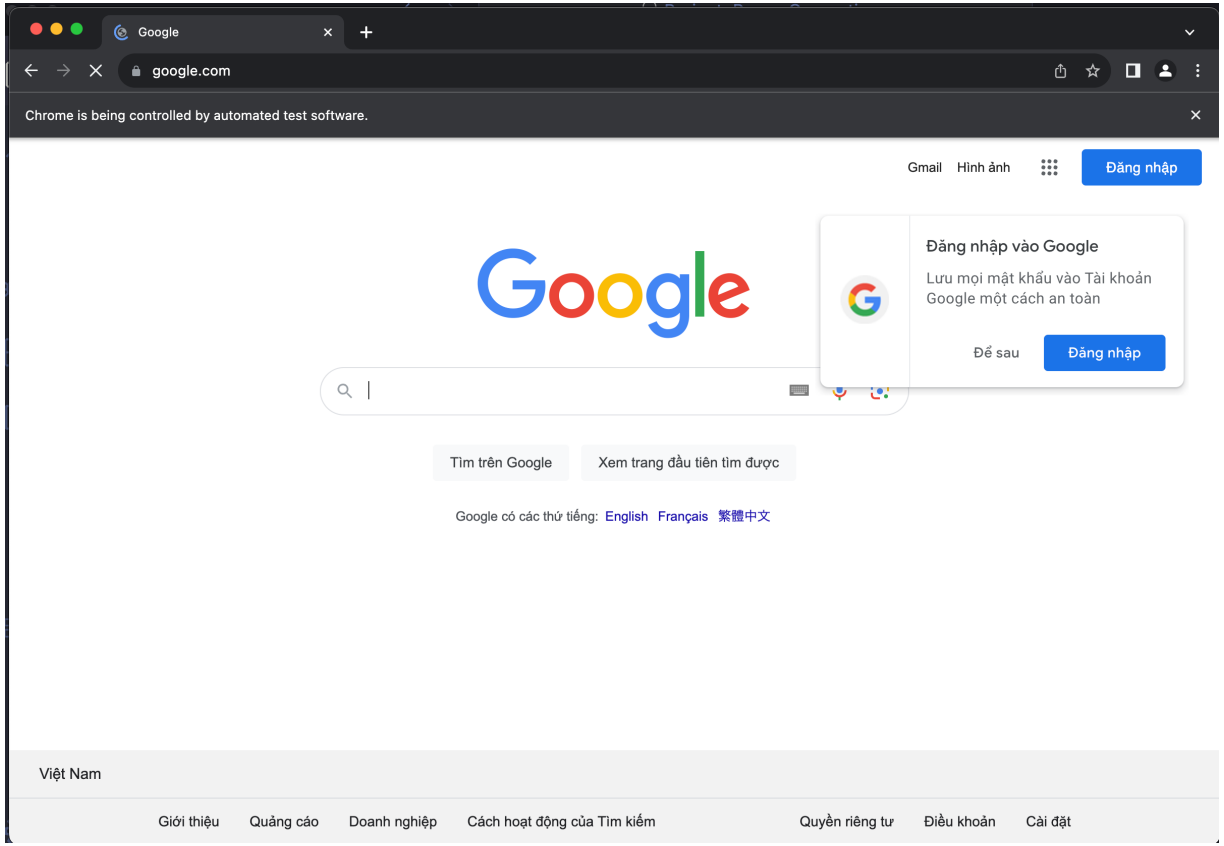


https://dev.to/cuongld2/introduction-to-selenium-19j6

Data Collection and Preparation

❖ Introduction to Selenium: Initialize driver

```
1  from selenium import webdriver
2
3  url = 'http://www.google.com'
4
5  driver = webdriver.Chrome()
6  driver.get(url)
```



Data Collection and Preparation

❖ Introduction to Selenium: HTML

```
<!DOCTYPE html>
<html>
    <!-- Root Element -->
    <head>
        <!-- Contains the header information -->
        <title>Title of the Page</title>
        <!-- Defines the title of the page -->
    </head>
    <body>
        <!-- Holds the content of the page -->
        <!-- Tags related to layout and formatting go here -->
    </body>
</html>
```

Data Collection and Preparation

❖ Coding: Import libraries and initialize webdriver

```python
import os
import requests
import time
import json

from tqdm import tqdm
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
```

```python
WEBDRIVER_DELAY_TIME_INT = 10
TIMEOUT_INT = 10
service = Service(executable_path=r"/usr/bin/chromedriver")
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument("--headless")
chrome_options.add_argument("--no-sandbox")
chrome_options.add_argument("--disable-dev-shm-usage")
chrome_options.add_argument("window-size=1920x1080")
chrome_options.headless = True
driver = webdriver.Chrome(service=service, options=chrome_options)
driver.implicitly_wait(TIMEOUT_INT)
wait = WebDriverWait(driver, WEBDRIVER_DELAY_TIME_INT)
```

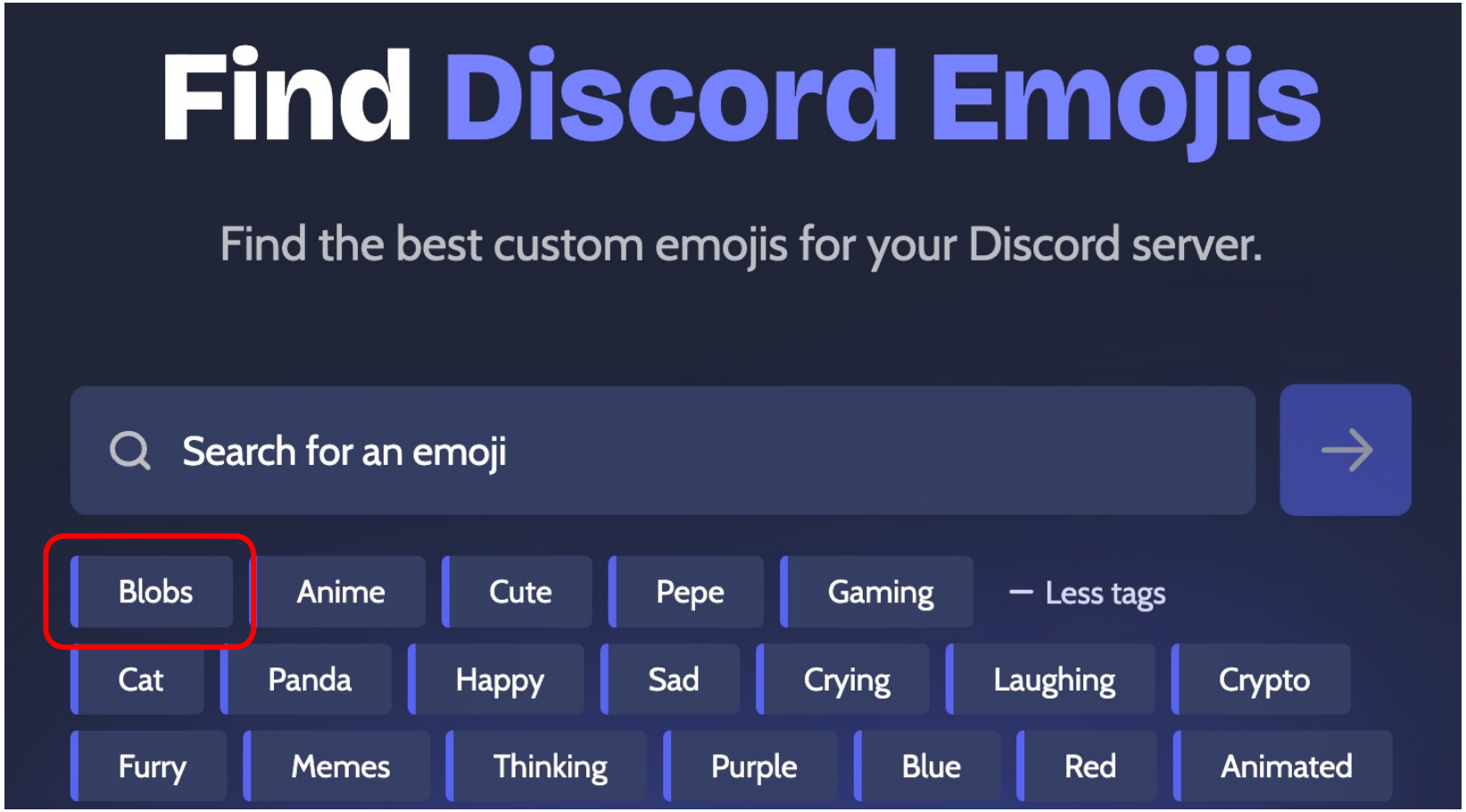
Data Collection and Preparation

❖ Website

Consider an emoji website: Discord Emojis.

Data Collection and Preparation

❖ Website: Tags

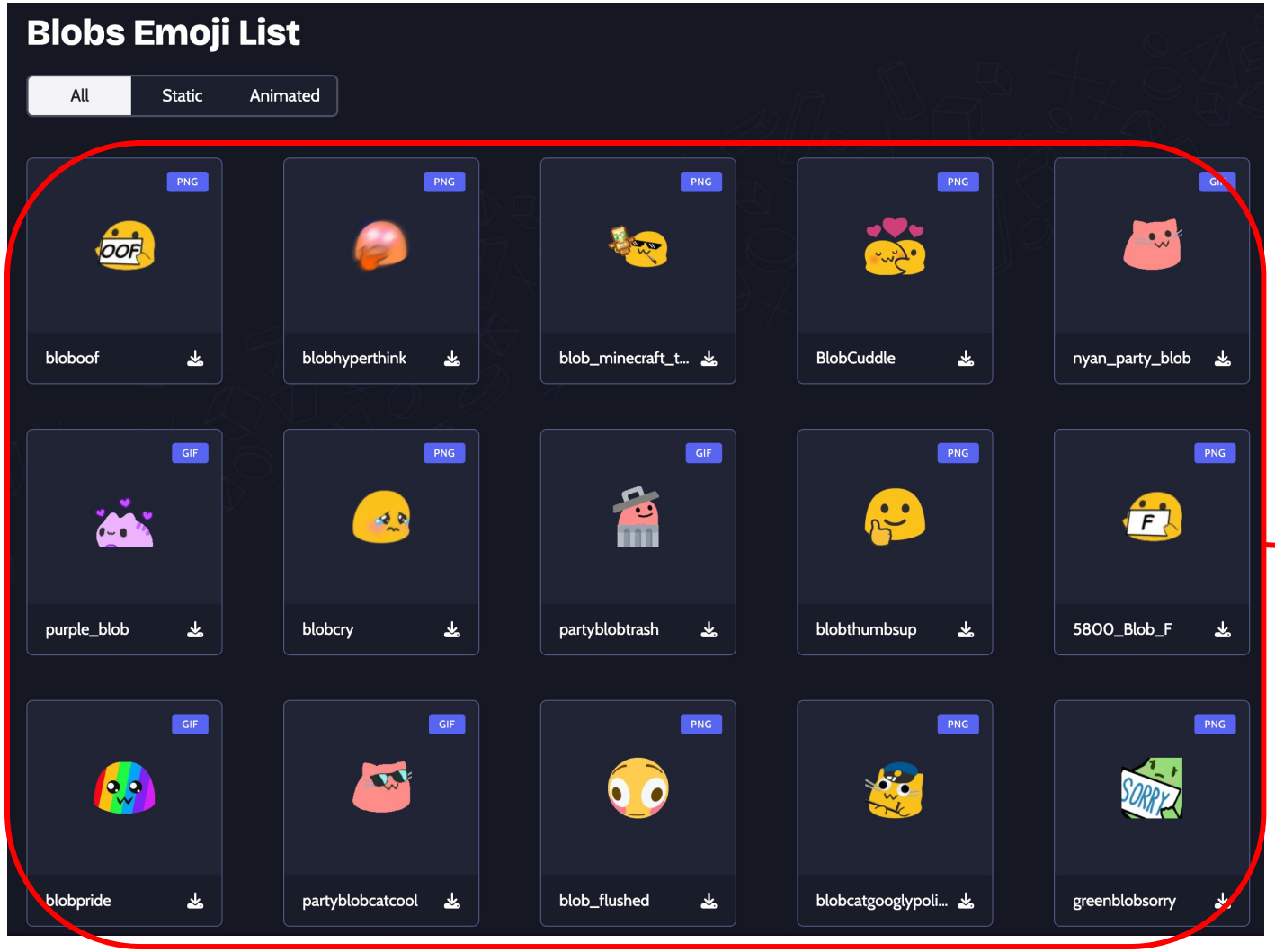


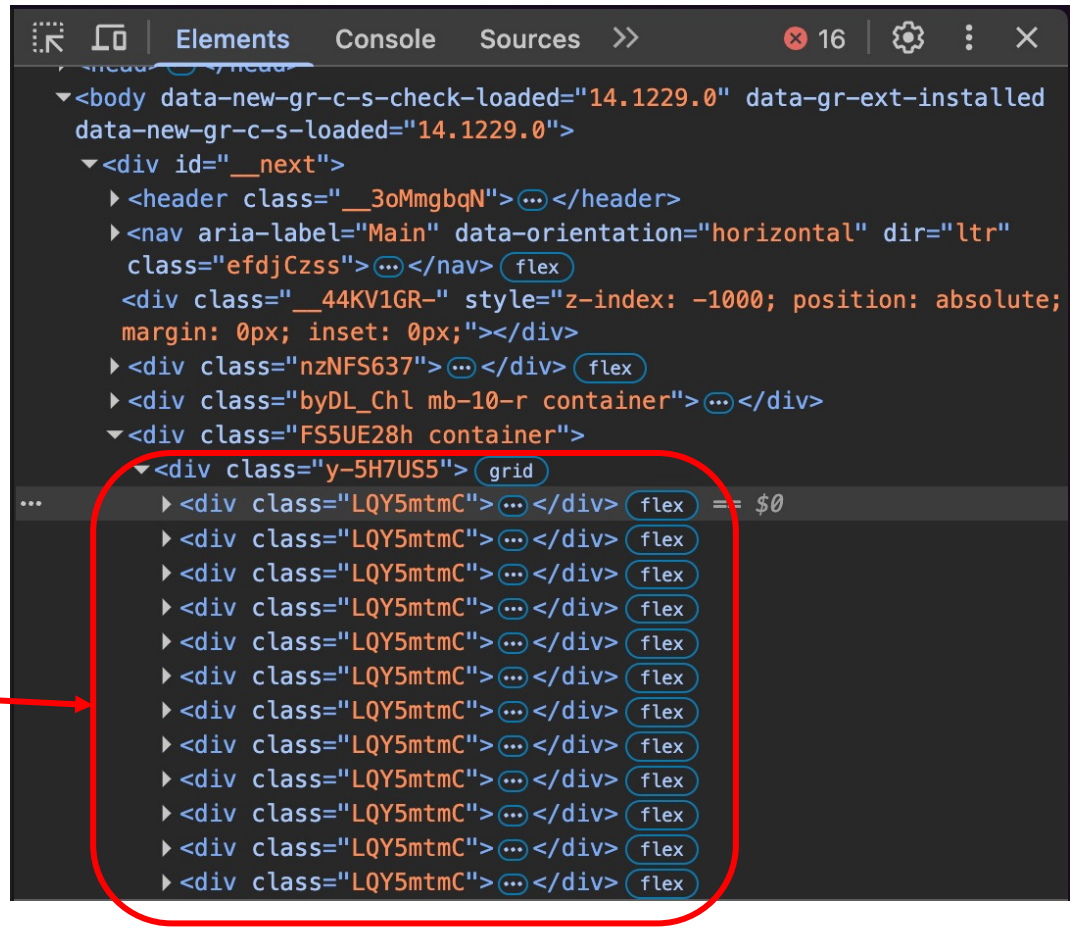
In our scope, **only Blobs** emojis are considered to be used for training the model.

AI VIET NAM
@aivietnam.edu.vn

Data Collection and Preparation

❖ Analyze website

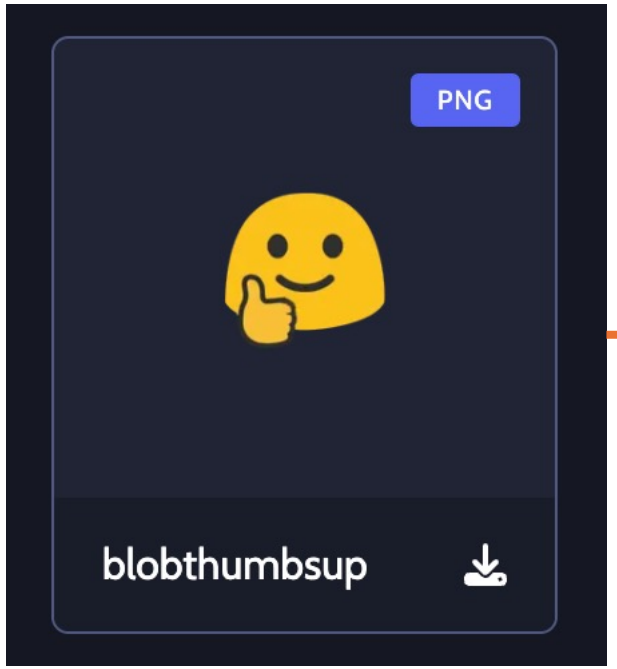


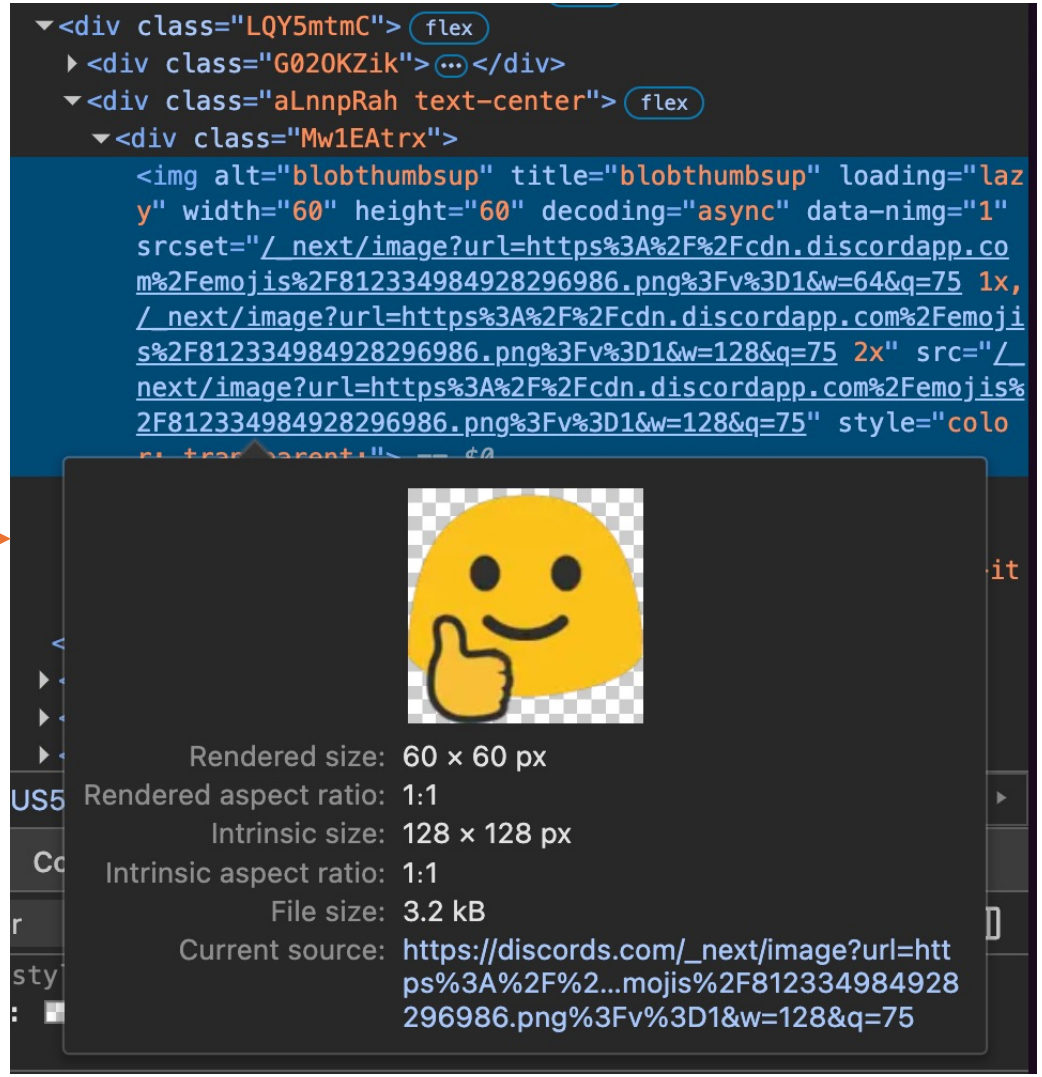


A list of Blobs emojis are represented as div class “y-5H7YS5”.

Data Collection and Preparation

❖ Analyze website



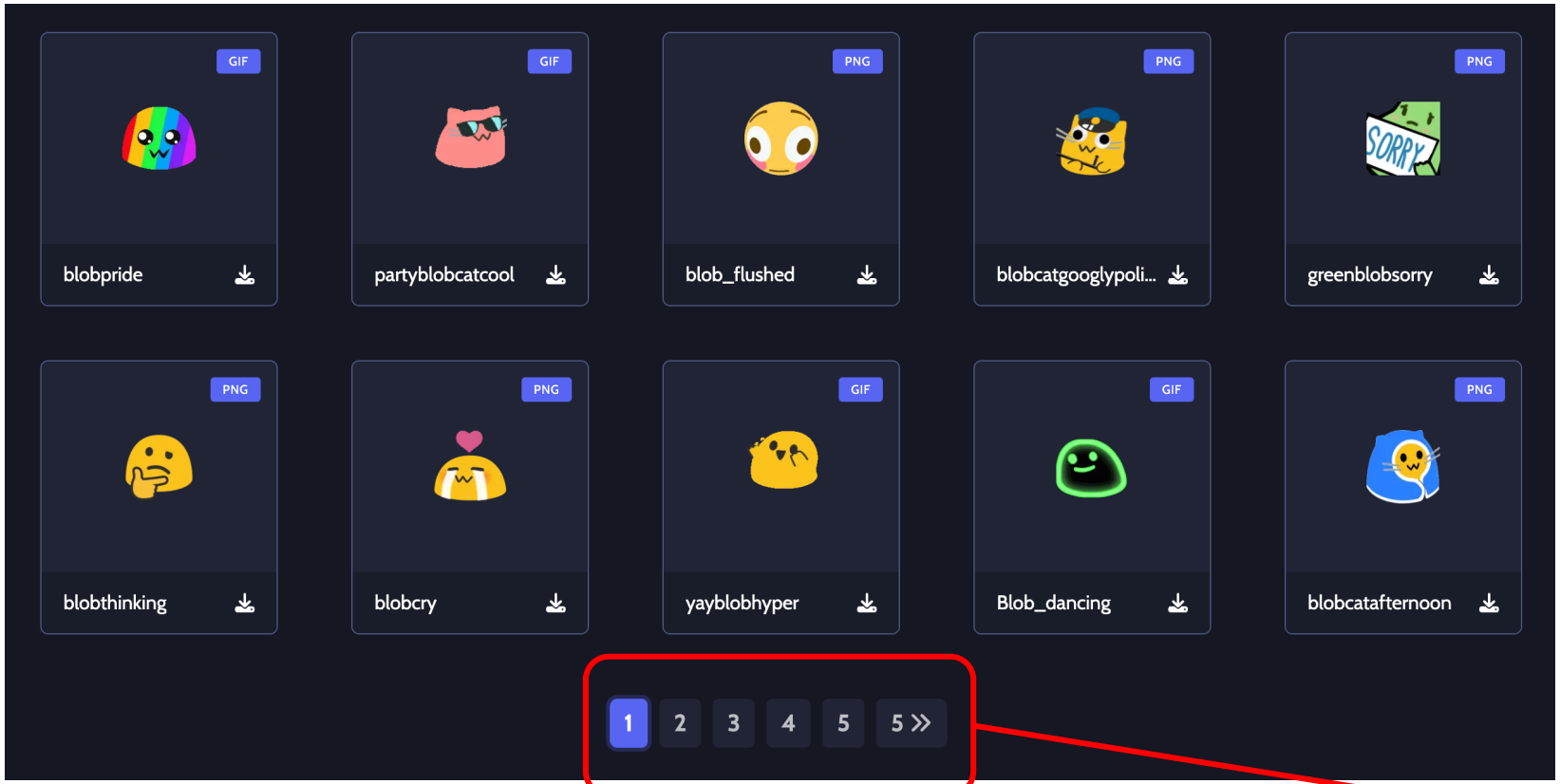


Information of an emoji that we can extract:

1. Image source (URL).
2. Title.

Data Collection and Preparation

❖ Analyze website

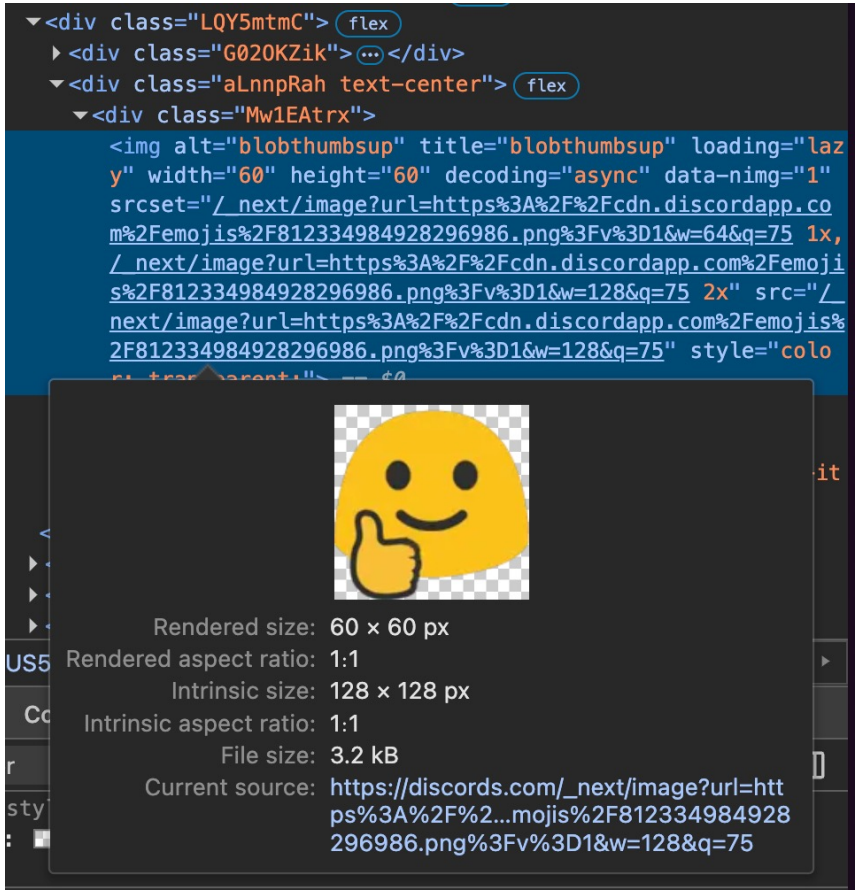


https://discords.com/emoji-list/tag/Blobs?page=1

Data Collection and Preparation

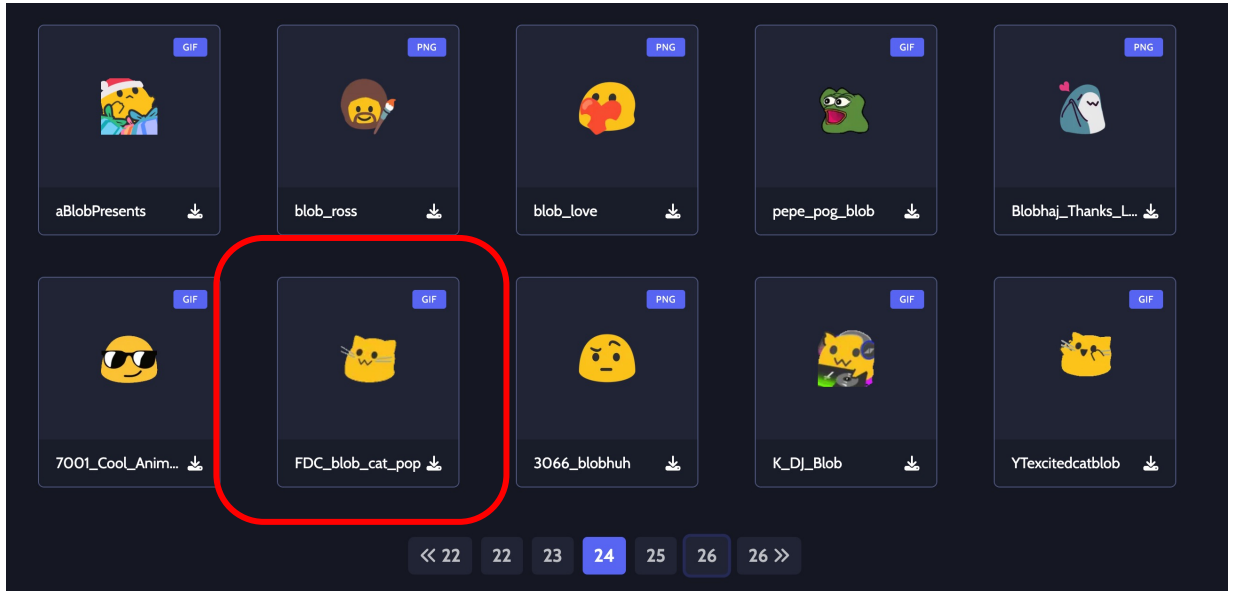
❖ Coding: Extract Image URLs

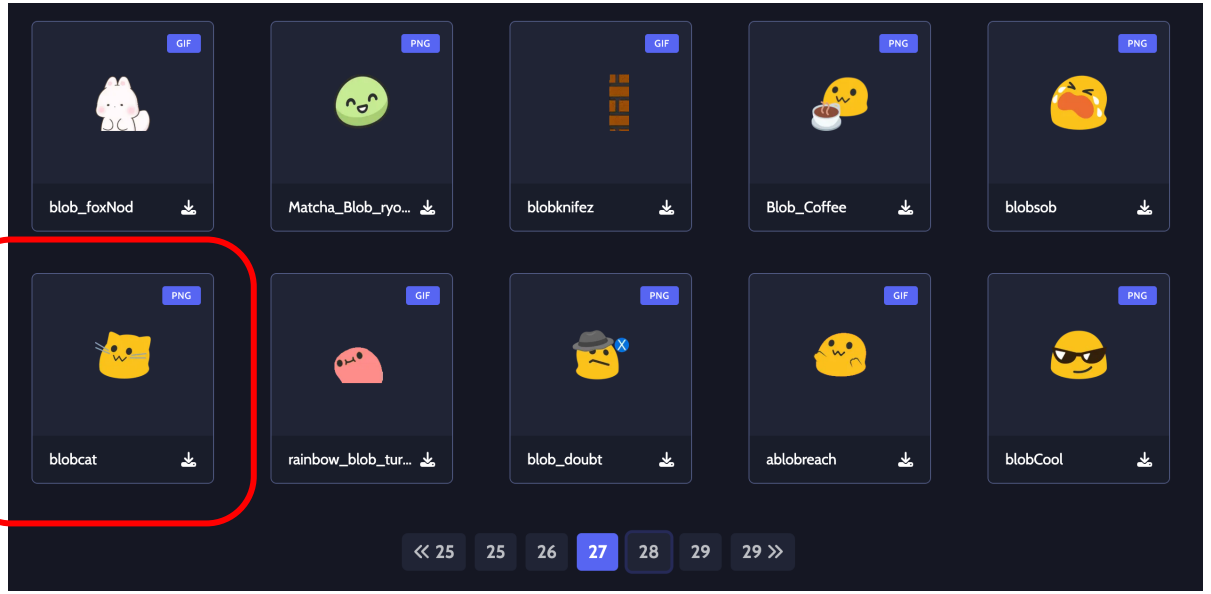
```python
def get_image_links_from_page(page_url, driver):
    driver.get(page_url)
    try:
        container = wait.until(EC.presence_of_element_located(
            (By.CSS_SELECTOR, "div.FS5UE28h.container")
        ))
        image_items = wait.until(EC.presence_of_all_elements_located(
            (By.CSS_SELECTOR, "div.LQY5mtmC div.aLnnpRah.text-center"))
        )

        image_links = []
        for img_elem in image_items:
            img_div = img_elem.find_element(By.CSS_SELECTOR, "div.Mw1EAtrx img, img")

            img_url = img_div.get_attribute("src")
            img_title = img_div.get_attribute("title")
            if img_url:
                image_links.append((img_url, img_title))

        return image_links
    except Exception as e:
        print(f"Error while trying to extract images: {e}")
        return []
```



Data Collection and Preparation

❖ Coding: Check image duplication





On this website, there might be the case where some emojis are duplicated.

Data Collection and Preparation

❖ Coding: Check image duplication

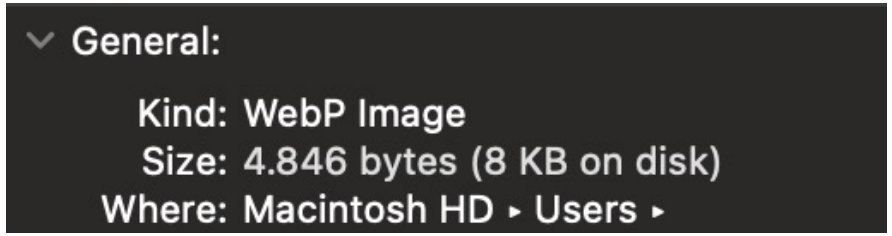
```python
def hash_image_content(url):
    try:
        response = requests.get(url, stream=True)
        if response.status_code == 200:
            return hashlib.md5(response.content).hexdigest()
        else:
            print(f"Error downloading image from {url}; status: {response.status_code}")
            return None
    except requests.exceptions.RequestException as e:
        print(f"Error with the image download for {url}: {e}")
        return None
```

For every image content, convert it into a hexadecimal string that will be used as a **unique identifier** to check for duplication.

Data Collection and Preparation

❖ Coding: Convert image format

```python
def convert_webp_to_jpg(webp_data):
    try:
        img = Image.open(BytesIO(webp_data))
        if img.format == 'WEBP':
            if img.mode == 'RGBA':
                img = img.convert('RGB')
            buffer = BytesIO()
            img.save(buffer, format="JPEG")
            return buffer.getvalue()
        else:
            return webp_data
    except Exception as e:
        print(f"Error converting WebP to JPG: {e}")
        return webp_data
```



Data Collection and Preparation

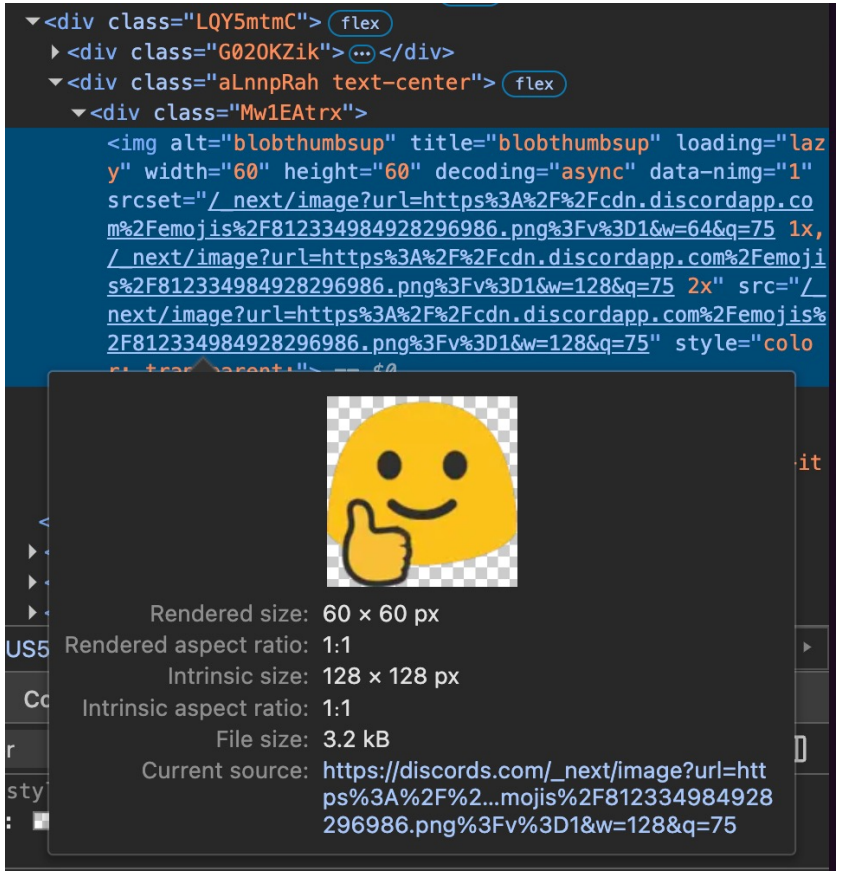
❖ Coding: Download image function

```python
def download_image(img_url, img_name, folder_path):
    try:
        response = requests.get(img_url, stream=True)
        if response.status_code == 200:
            img_path = os.path.join(folder_path, f"{img_name}")
            img_data = convert_webp_to_jpg(response.content)
            with open(img_path, "wb") as f:
                f.write(img_data)
        else:
            print(f"Error downloading image from {img_url}; status: {response.status_code}")
    except requests.exceptions.RequestException as e:
        print(f"Error with the image download for {img_url}: {e}")
```

Data Collection and Preparation

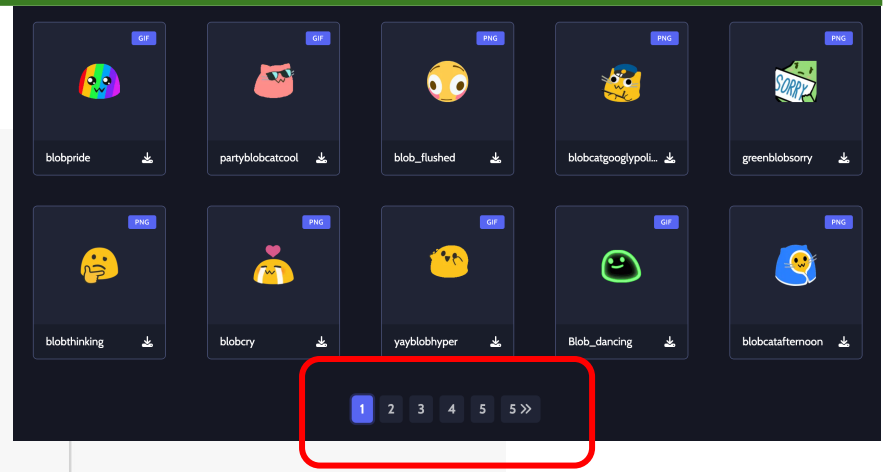
❖ Coding: Process image function

```python
def process_image_page(image_url, img_title, folder_path, idx, tag, seen_hashes):
    img_hash = hash_image_content(image_url)
    if img_hash and img_hash not in seen_hashes:
        seen_hashes.add(img_hash)
        new_file_name = f"{tag}_{idx:07d}.jpg"
        download_image(image_url, new_file_name, folder_path)
        metadata = {
            "file_name": new_file_name,
            "image_url": image_url,
            "image_title": img_title,
            "tag": tag
        }
        return metadata
    else:
        return None
```



Data Collection and Preparation

❖ Coding: Process image function




```python
def loop_over_pages(base_url, tags, total_pages, driver, folder_path):
    os.makedirs(folder_path, exist_ok=True)
    all_metadata = []
    seen_hashes = set()

    for tag in tags:
        all_images = []

        for page in tqdm(range(1, total_pages + 1), desc=f"Extracting Images for {tag}", unit="page"):
            page_url = f"{base_url}/emoji-list/tag/{tag}?page={page}"
            images = get_image_links_from_page(page_url, driver)
            all_images.extend(images)

            time.sleep(1)

        metadata_list = []
        for idx, (img_url, img_title) in enumerate(all_images, start=1):
            metadata = process_image_page(img_url, img_title, folder_path, idx, tag, seen_hashes)
            if metadata:
                metadata_list.append(metadata)

        all_metadata.extend(metadata_list)

    return all_metadata
```

```python
def save_metadata(metadata_list, metadata_file):
    df = pd.DataFrame(metadata_list)
    df.to_csv(metadata_file,
              index=False,
              encoding="utf-8")
```

Data Collection and Preparation

❖ Coding: Start crawling

```python
os.makedirs("crawled_data", exist_ok=True)
folder_path = os.path.join("crawled_data", "images")
metadata_file = os.path.join("crawled_data", "metadata.csv")

base_url = "https://discords.com"
tags = ["Panda"]
total_pages = 1000

metadata_list = loop_over_pages(base_url, tags, total_pages, driver, folder_path)
save_metadata(metadata_list, metadata_file)

print("Start downloading images...")
with tqdm(total=len(metadata_list), desc="Downloading Images", unit="image") as pbar:
    for metadata in metadata_list:
        img_url = metadata['image_url']
        file_name = metadata['file_name']
        download_image(img_url, file_name, folder_path)
        pbar.update(1)

print("Download images completed.")

total_crawled_images = len(os.listdir(folder_path))
print(f"Total crawled images: {total_crawled_images}.")

driver.quit()
```

```
#11 0x592a2065bce7 <unknown>
#12 0x592a2065b9e4 <unknown>
#13 0x592a2066013a <unknown>
#14 0x592a2065c5b9 <unknown>
#15 0x592a20641e00 <unknown>
#16 0x592a206735d2 <unknown>
#17 0x592a20673778 <unknown>
#18 0x592a2068ba1f <unknown>
#19 0x7ae3c963fac3 <unknown>
#20 0x7ae3c96d1850 <unknown>

Extracting Images for Panda: 100%|[redacted]|
Error downloading image from https://discords.
Error downloading image from https://discords.
Error downloading image from https://discords.
Error downloading image from https://discords.
Error downloading image from https://discords.
Start downloading images...
Downloading Images: 100%|[redacted]| 3942/3942
Download images completed.
Total crawled images: 3942.
```

Data Collection and Preparation

❖ Crawled emojis

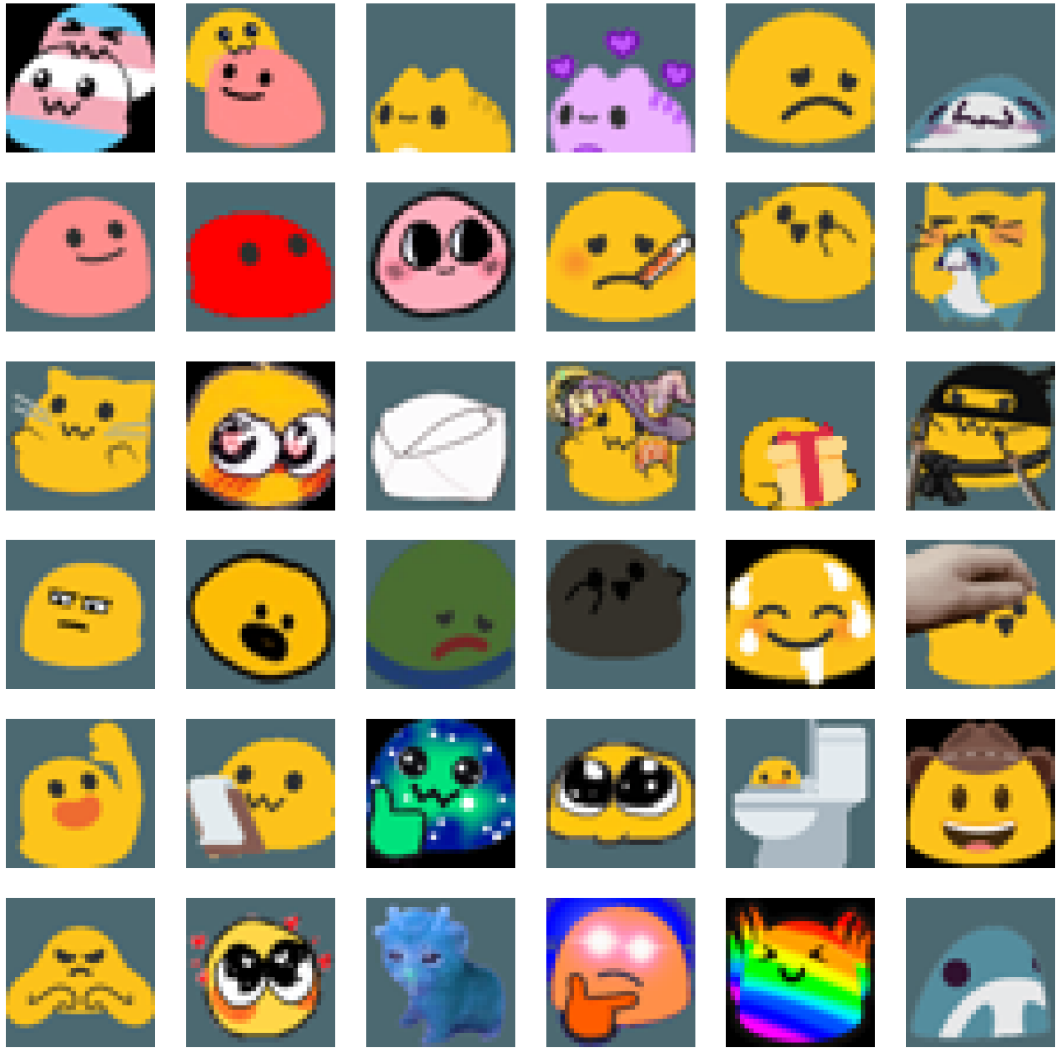
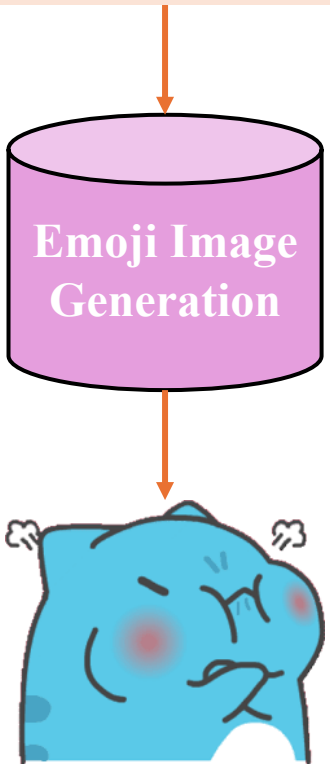


Figure: A bunch of emojis crawled using our code.

Data Collection and Preparation

❖ Crawled emojis

"Blue cat emoji with a round head, blushing cheeks, puffed face, angry expression, tiny clenched fists raised beside the head, steam puffs above both ears, no background."



Input

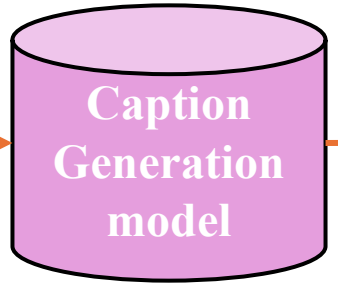
Output

Currently, there is no complete prompt/description for each image of our dataset.

Data Collection and Preparation

❖ Caption for each emoji

Input



Output

"Blue cat emoji with a round head, blushing cheeks, puffed face, angry expression, tiny clenched fists raised beside the head, steam puffs above both ears, no background."

Thus, to complete the dataset where each sample consists of an image and a prompt, a model to create pseudo label for image is required.

Data Collection and Preparation

❖ Idea: Use LVLM?

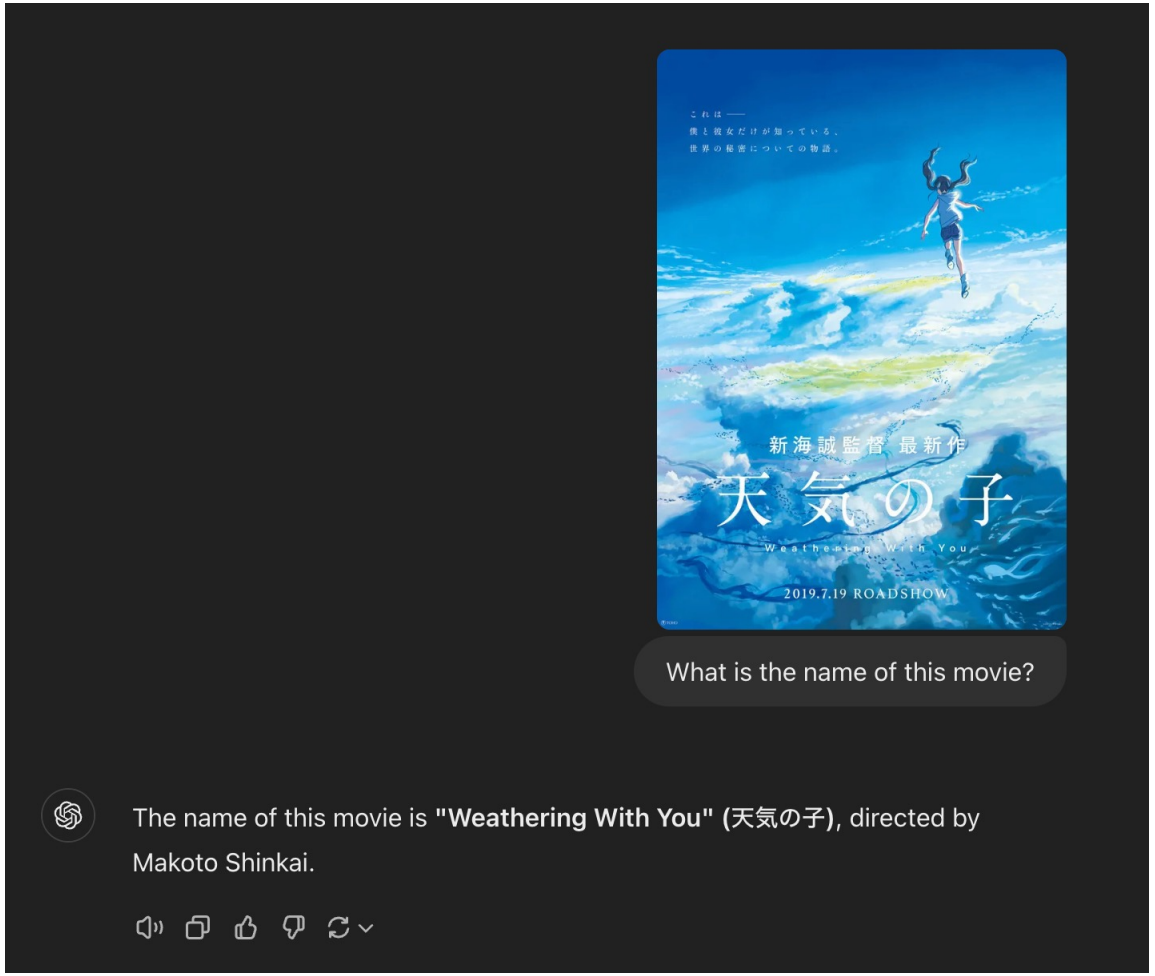


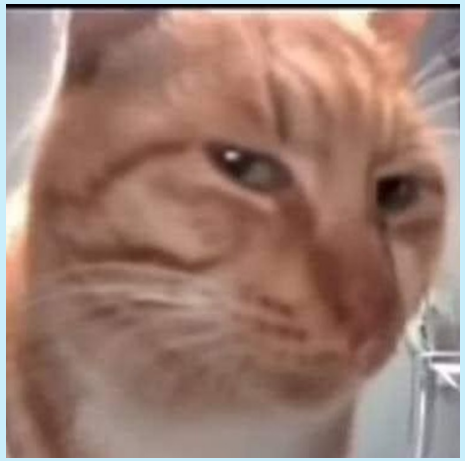
Figure: ChatGPT, one of the most famous application that utilizing LVLMs.

Large Vision-Language Models (LVLMs) are models that integrate large language models (LLMs) with pre-trained vision encoders, enabling them to process image inputs, understand various queries, and perform reasoning tasks.

Data Collection and Preparation

❖ Idea: Use LVLM?

Image:



Question: What is the feeling of the cat in this image?

?

LLM

LLM only accepts text, how can an image can be feeded into?

Figure: Illustration of a pair of (image, text) input.

Data Collection and Preparation

❖ LVLMs Architecture

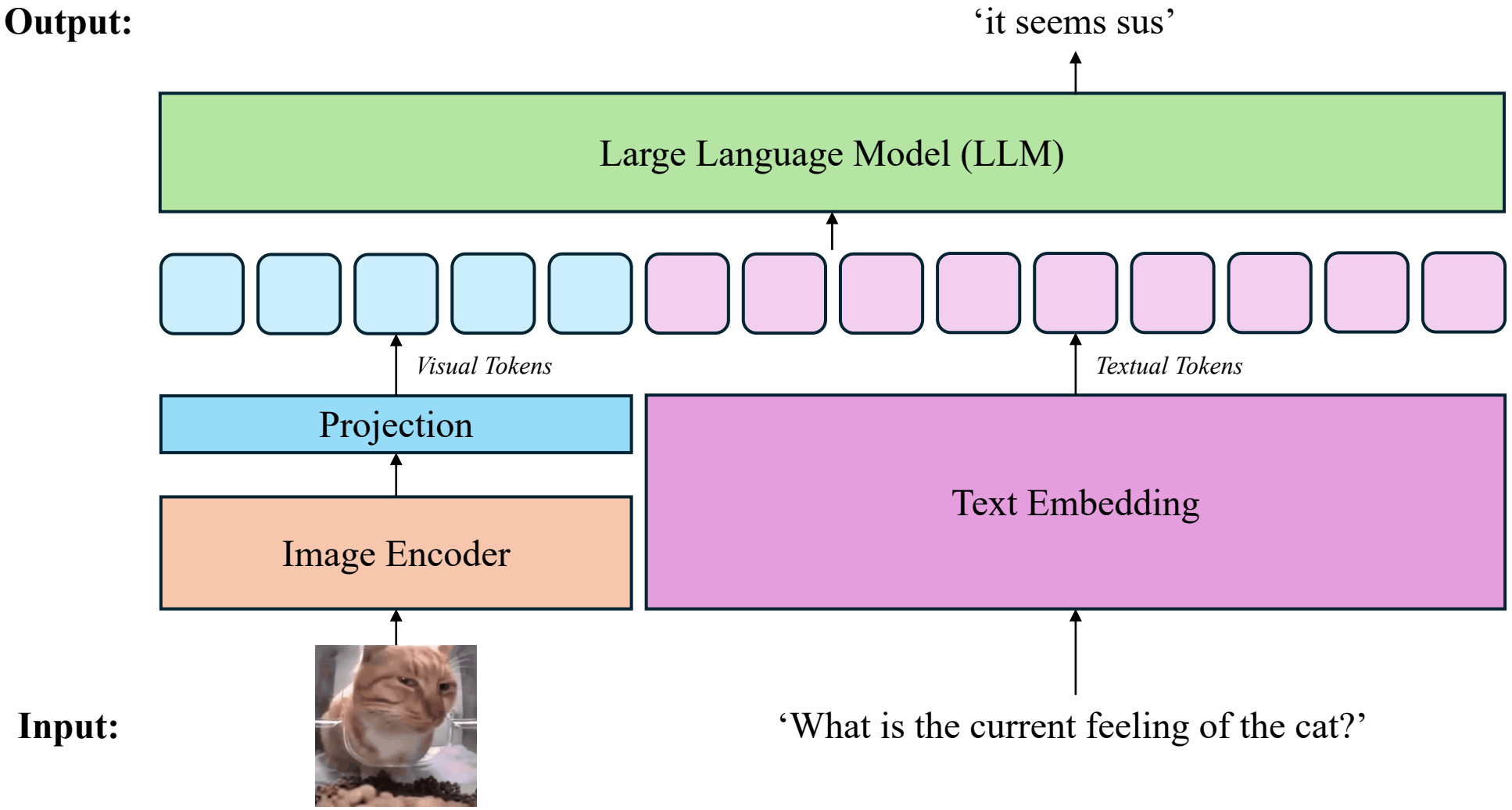


Figure: A typical architecture of LVLMs.

Data Collection and Preparation

❖ Capability of VLMs

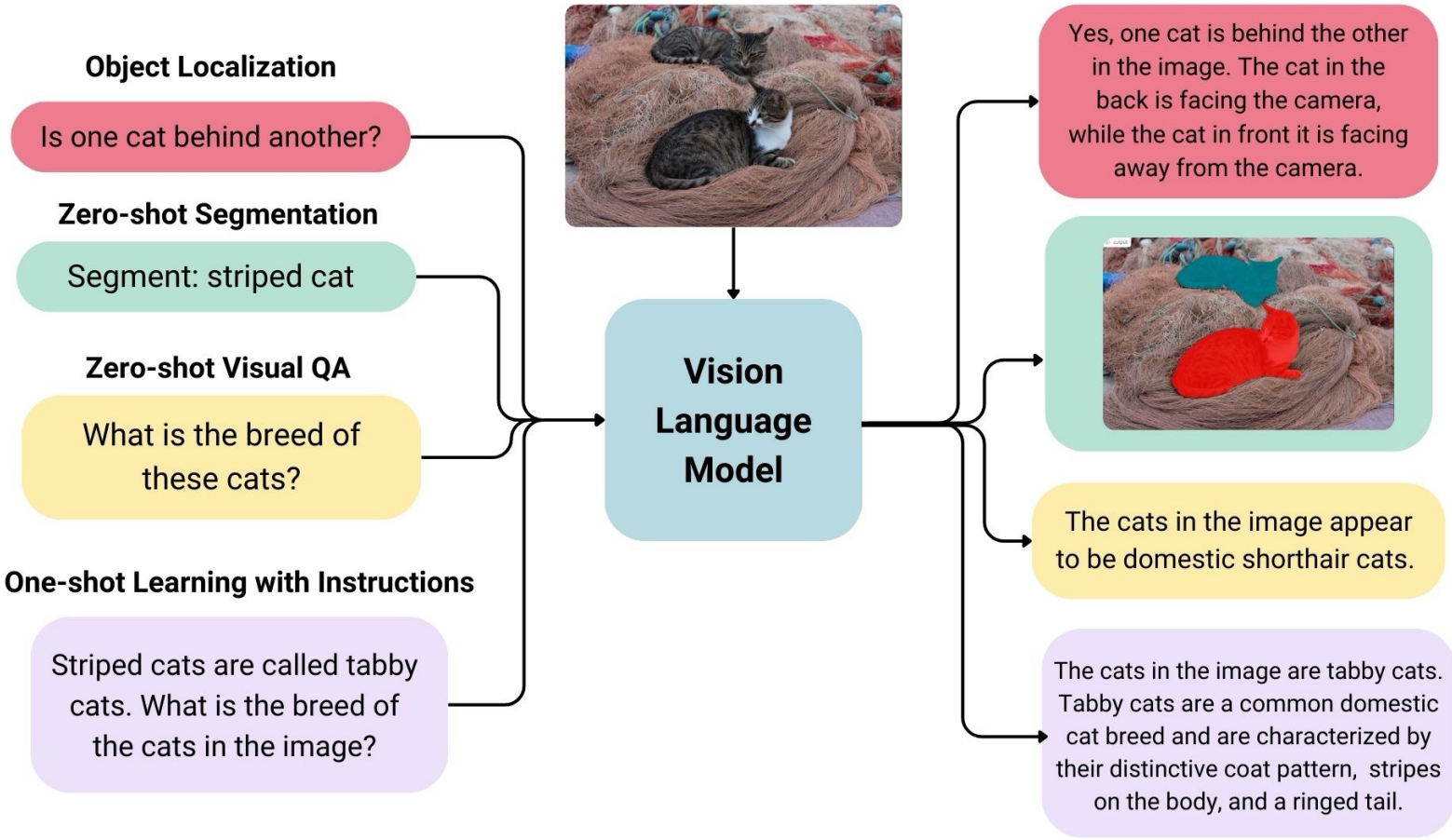
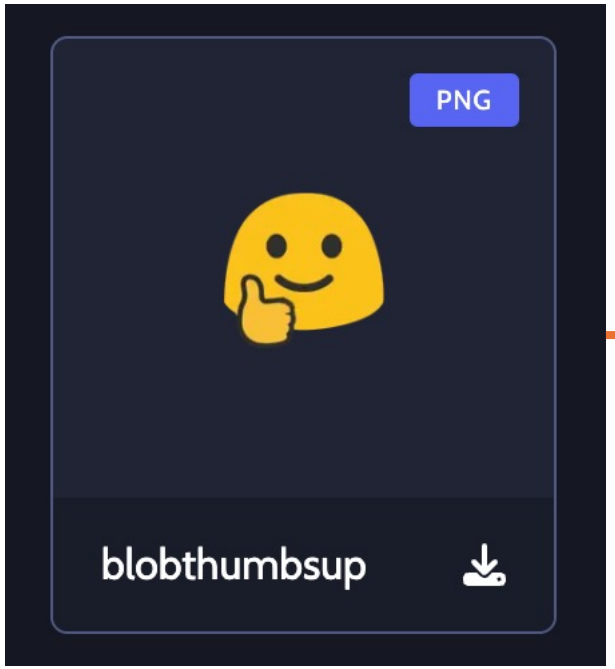


Figure: Illustration of what a Large Vision-Language Model (LVLM) can do nowadays.

https://huggingface.co/blog/vlms

Data Collection and Preparation

❖ Generate caption for emojis



Large Vision Language Model (LVLM)

Blobthumbsup emoji with a round head, small dot eyes, smiling mouth, cheerful expression, white skin color, giving a thumbs-up action, with a white background.

Data Collection and Preparation

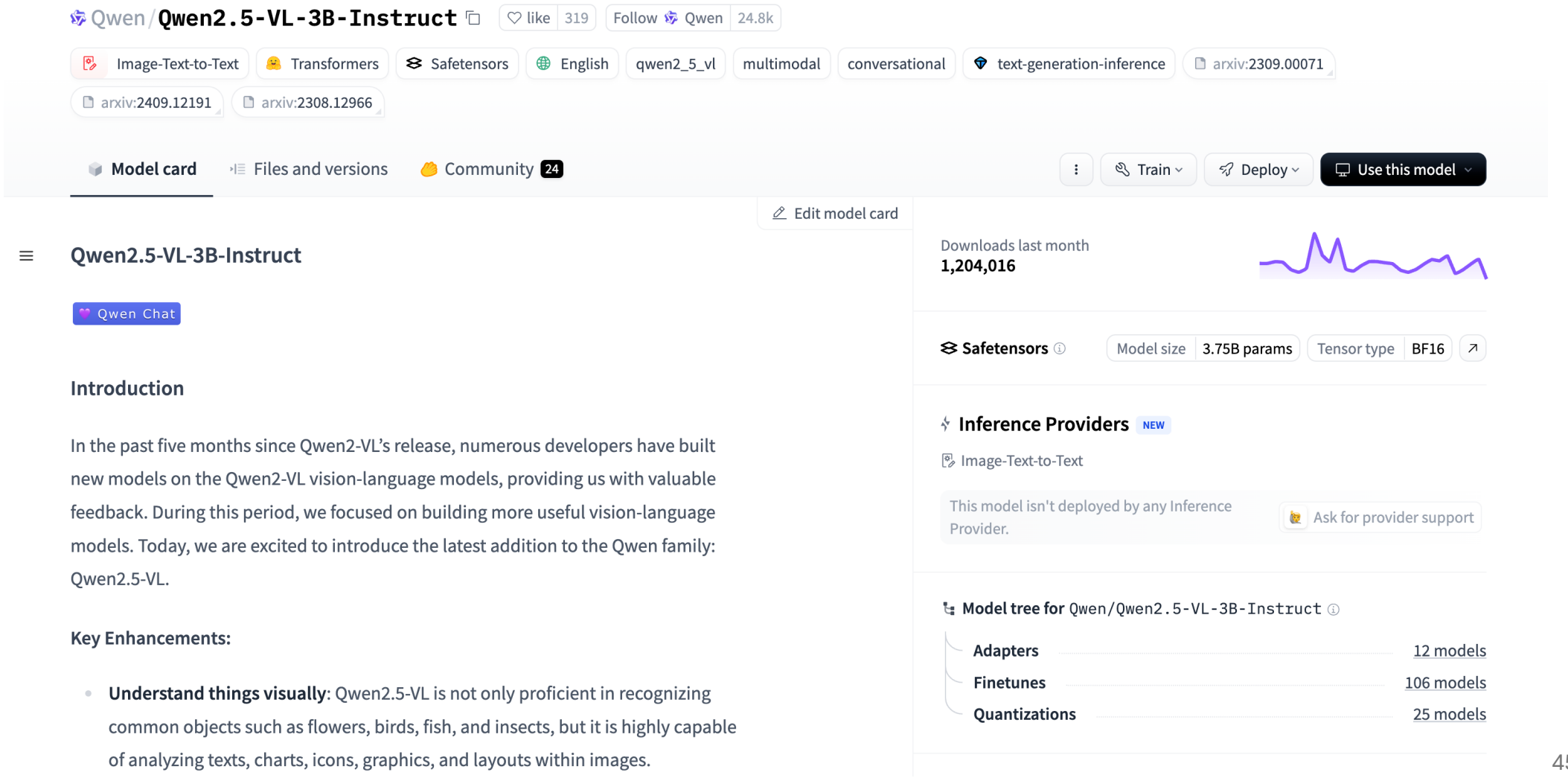
❖ Choosing LVLMs

Qwen-VL: A large-scale vision-language model that integrates image and text understanding for multi-modal tasks, enabling advanced capabilities such as image captioning, visual question answering, and cross-modal reasoning.

It leverages transformer-based architectures and extensive training to generate highly coherent outputs and accurately link textual information with visual content.

Data Collection and Preparation

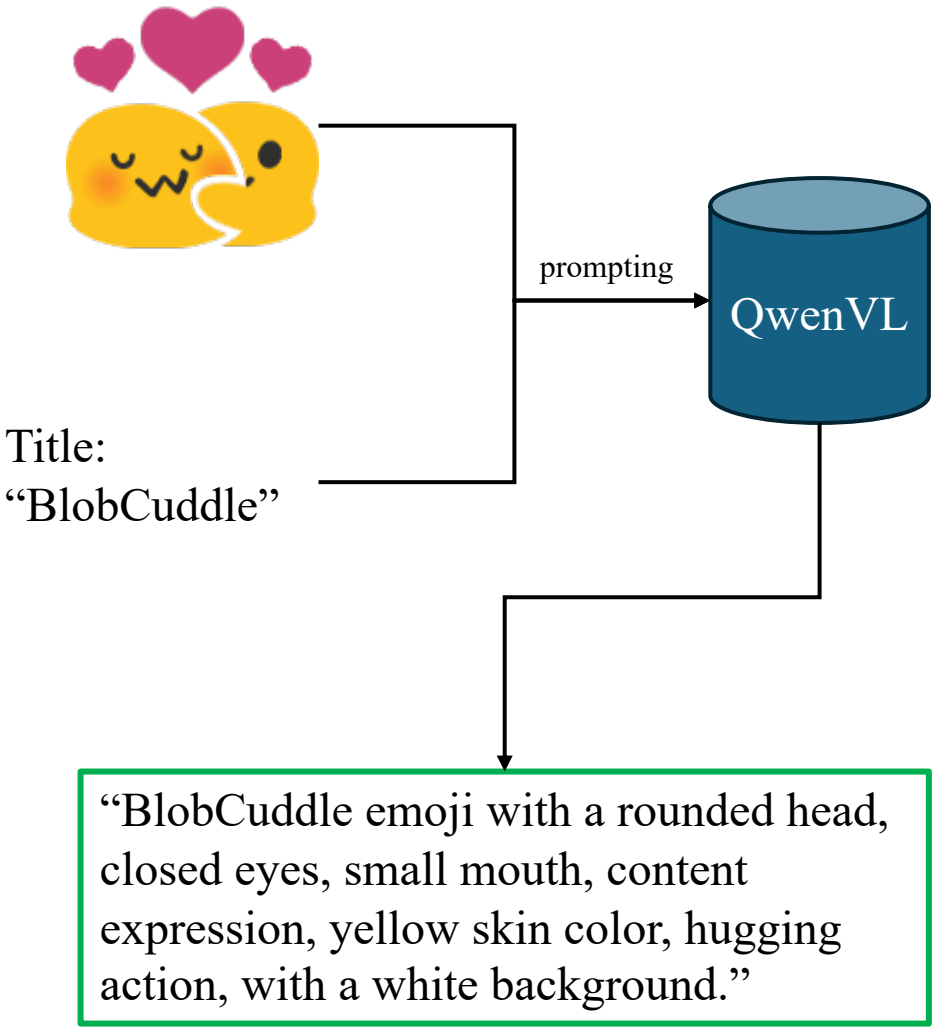
❖ Choosing LVLMs



Data Collection and Preparation

❖ Prompting QwenVL for emoji captioning

Instruction: Given the emoji title and the corresponding image, generate a natural language description of the emoji's key features. The description should include: 1) the head shape, 2) eye characteristics, 3) mouth characteristics, 4) facial expression, 5) skin color, 6) any action (if present), and 7) background color. The description should be concise and structured like this: '{emoji_title} emoji with a {head shape}, {eye description} eyes, {mouth description} mouth, {expression description} expression, {skin color} skin color, {action description}, with a {background color} background.'
Example: For an emoji titled 'Pepe the Frog', the description should be like: 'Pepe the frog emoji with a round head, big eyes, smiling mouth, happy expression, green skin color, no action, with a green background.'
Input: '{emoji_title}'t



Data Collection and Preparation

❖ Coding

```python
import os
import pandas as pd
import torch
from transformers import Qwen2_5_VLForConditionalGeneration
from transformers import AutoProcessor
from PIL import Image
from qwen_vl_utils import process_vision_info
from tqdm import tqdm
```

```python
model_id = "Qwen/Qwen2.5-VL-3B-Instruct"

model = Qwen2_5_VLForConditionalGeneration.from_pretrained(
    model_id,
    torch_dtype=torch.bfloat16,
    device_map="auto"
)

processor = AutoProcessor.from_pretrained(model_id)
```

Data Collection and Preparation

❖ Coding: Create prompt function

```python
def create_prompt(emoji_title):
    instruction = (
        "Instruction: Given the emoji title and the corresponding image, generate a natural language description "
        "of the emoji's key features. The description should include: 1) the head shape, 2) eye characteristics, "
        "3) mouth characteristics, 4) facial expression, 5) skin color, 6) any action (if present), and 7) background color. "
        "The description should be concise and structured like this: "
        "'{emoji_title} emoji with a {head shape}, {eye description} eyes, {mouth description} mouth, "
        "{expression description} expression, {skin color} skin color, {action description}, "
        "with a {background color} background.'"
    )

    example = (
        "Example: For an emoji titled 'Pepe the Frog', the description should be like: "
        "'Pepe the frog emoji with a round head, big eyes, smiling mouth, happy expression, "
        "green skin color, no action, with a green background.'"
    )

    prompt = f"{instruction}\n{example}\nInput: '{emoji_title}'"
    return prompt
```

Data Collection and Preparation

❖ Coding: Setup input

```
for _, row in tqdm(df.iterrows(), total=len(df), desc=f"Processing {subfolder}"):
    image_path = os.path.join(images_folder, row["file_name"])
    prompt = create_prompt(row["image_title"])

    image = Image.open(image_path).convert("RGBA")

    messages = [
        {
            "role": "user",
            "content": [
                {"type": "image", "image": image},
                {"type": "text", "text": prompt},
            ],
        }
    ]

    text = processor.apply_chat_template(
        messages, tokenize=False, add_generation_prompt=True
    )
    image_inputs, video_inputs = process_vision_info(messages)
```

Data Collection and Preparation

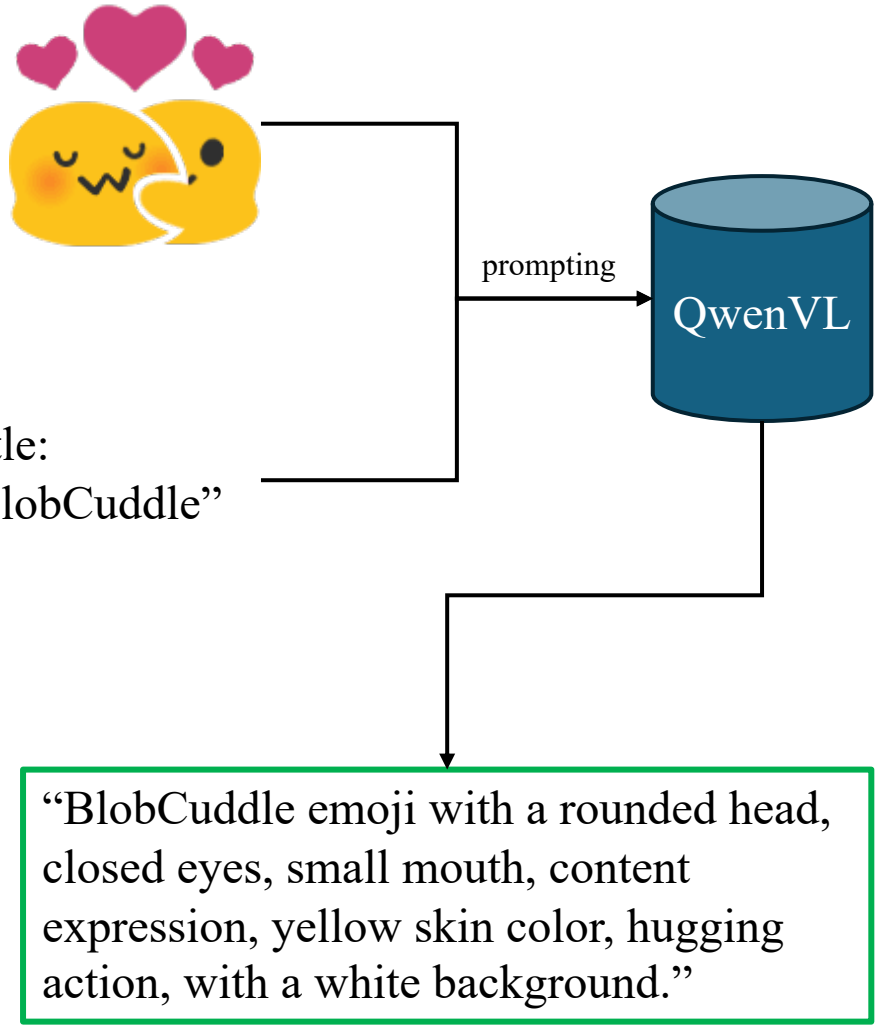
❖ Coding: Generate caption

```
    inputs = processor(
        text=[text],
        images=image_inputs,
        videos=video_inputs,
        padding=True,
        return_tensors="pt"
    ).to("cuda")

    generated_ids = model.generate(**inputs, max_new_tokens=128)
    generated_ids_trimmed = [
        out_ids[len(in_ids):] for in_ids, out_ids in zip(inputs.input_ids, generated_ids)
    ]
    result_text = processor.batch_decode(
        generated_ids_trimmed, skip_special_tokens=True, clean_up_tokenization_spaces=False
    )[0]

    df.loc[_, "prompt"] = result_text
```

Data Collection and Preparation

❖ Results

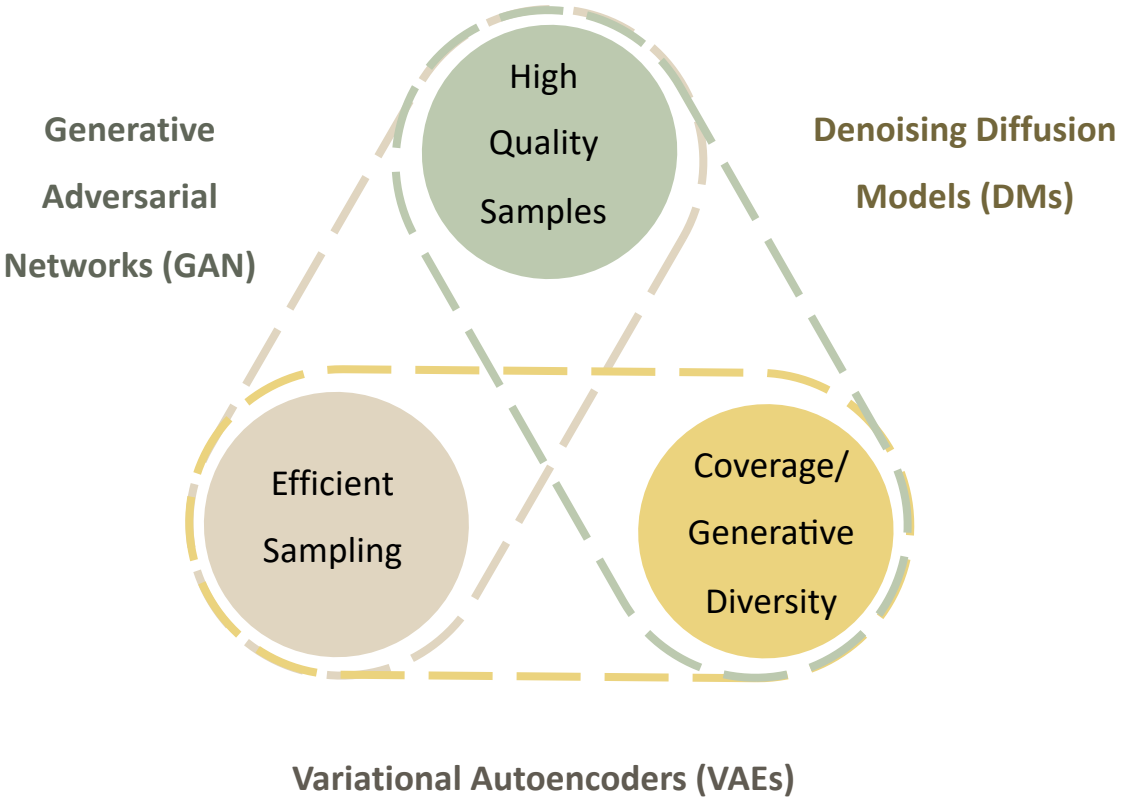
image_title	tag	prompt
bloboof	Blobs	bloboof emoji with a square head, small eyes, open mouth, surprised expression, white skin color, no action, with a white background.
blobhyperthink	Blobs	blobhyperthink emoji with a round head, large circular eyes, wide open mouth, surprised expression, orange skin color, no action, with a white background.
blob_minecraft_totem_of_undying	Blobs	blob_minecraft_totem_of_undying emoji with a blob-like head, large eyes, wide open mouth, surprised expression, yellow skin color, no action, with a white background.
BlobCuddle	Blobs	BlobCuddle emoji with a rounded head, closed eyes, small mouth, content expression, yellow skin color, hugging action, with a white background.
nyan_party_blob	Blobs	Nyan Party Blob emoji with a rounded head, small black eyes, squiggly mouth, surprised expression, pink skin color, no action, with a white background.
purple_blob	Blobs	Purple Blob emoji with a round head, small black eyes, smiling mouth, happy expression, purple skin color, waving action, with a white background.



Training SD from scratch

Training SD from scratch

❖ Introduction: Generative models

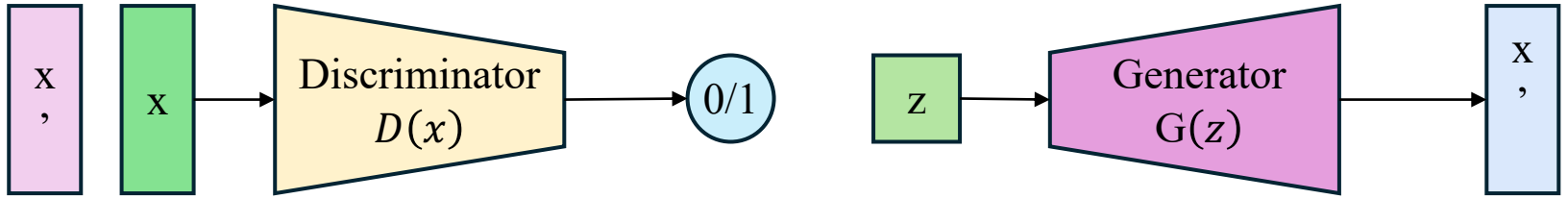


FEATURES	GAN	VAE	DIFFUSION MODEL
IMAGE QUALITY	High-quality, sharp images	Often blurry, less realistic	Very high and realistic
SAMPLE DIVERSITY	Lower	Moderate diversity	High diversity
SAMPLING SPEED	Fast	Fast	Slow

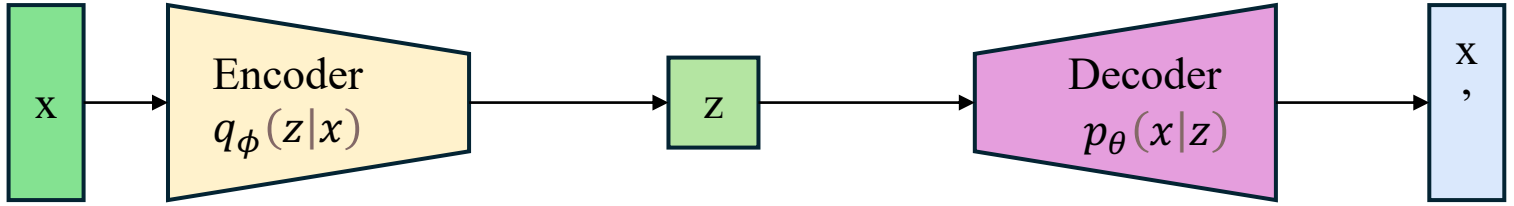
Training SD from scratch

❖ Introduction: Generative models

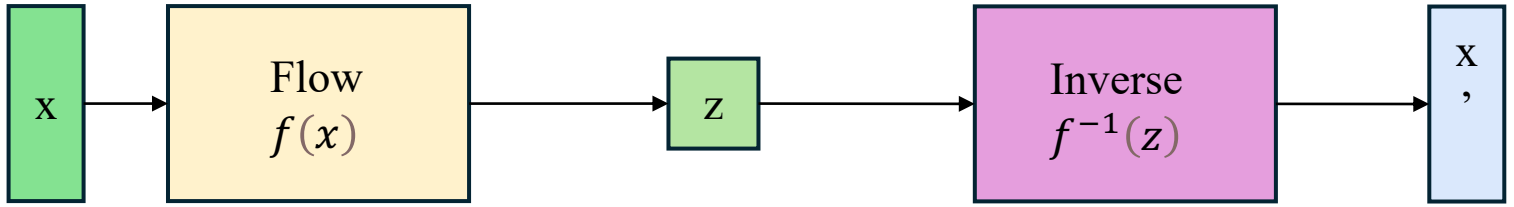
GAN: Adversarial training.



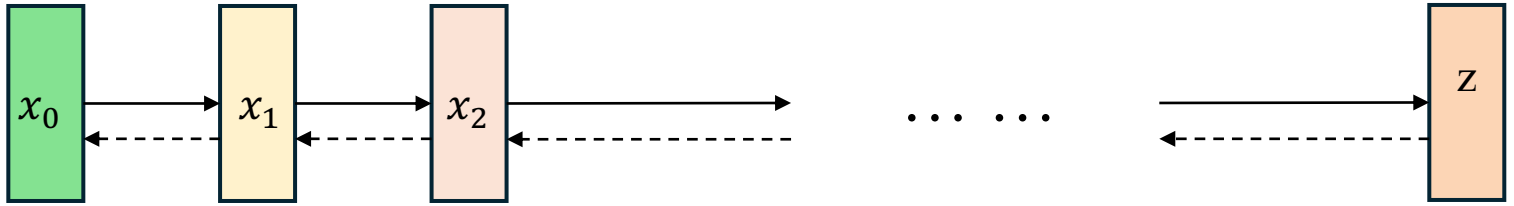
VAE: maximize variational lower bound.



Flow-based models: Invertible transform of distributions.



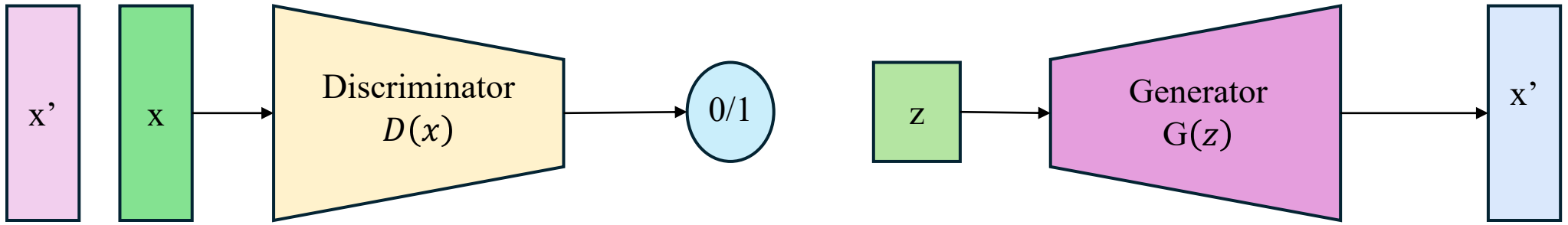
Diffusion models: Gradually add Gaussian noise and then reverse.



Training SD from scratch

❖ Introduction: GANs

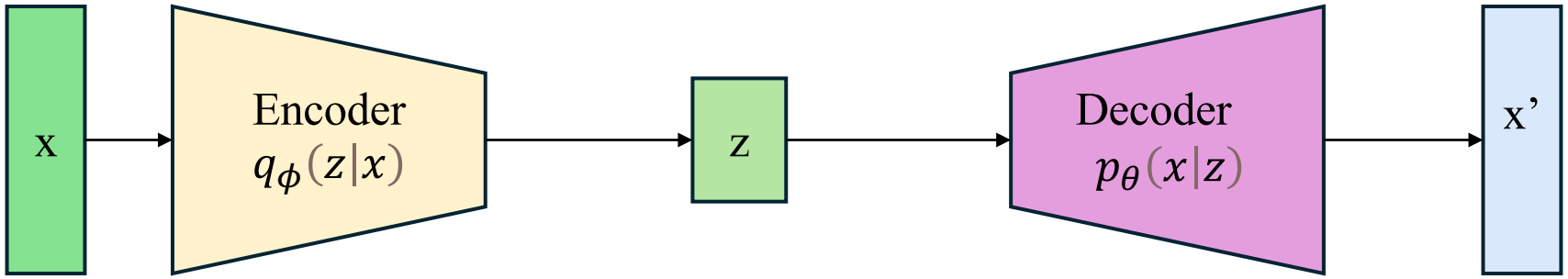
GAN: Adversarial training



Training SD from scratch

❖ Introduction: VAE

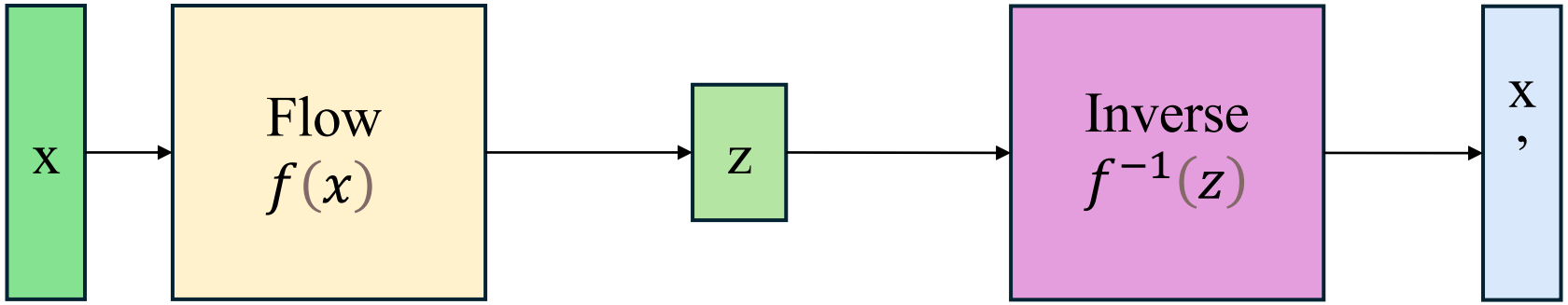
VAE: maximize variational lower bound.



Training SD from scratch

❖ Introduction: Flow Matching

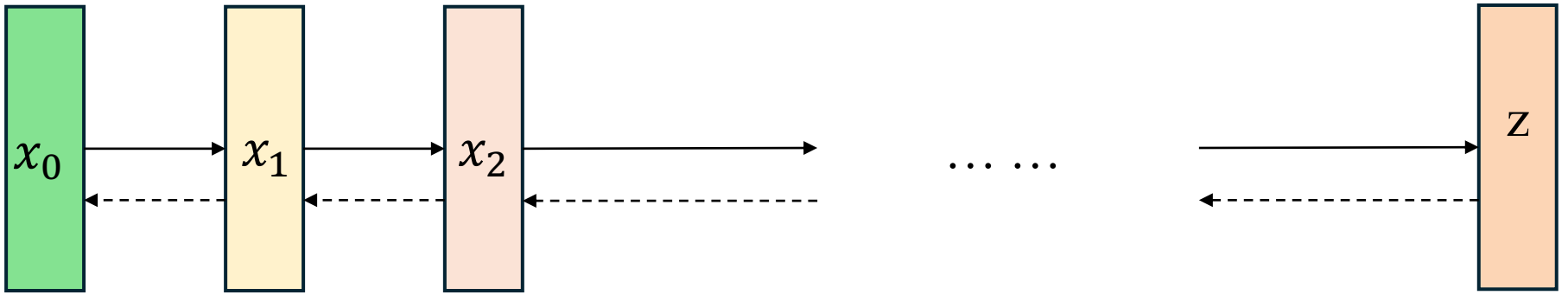
Flow-based models:
Invertible transform of distributions.



Training SD from scratch

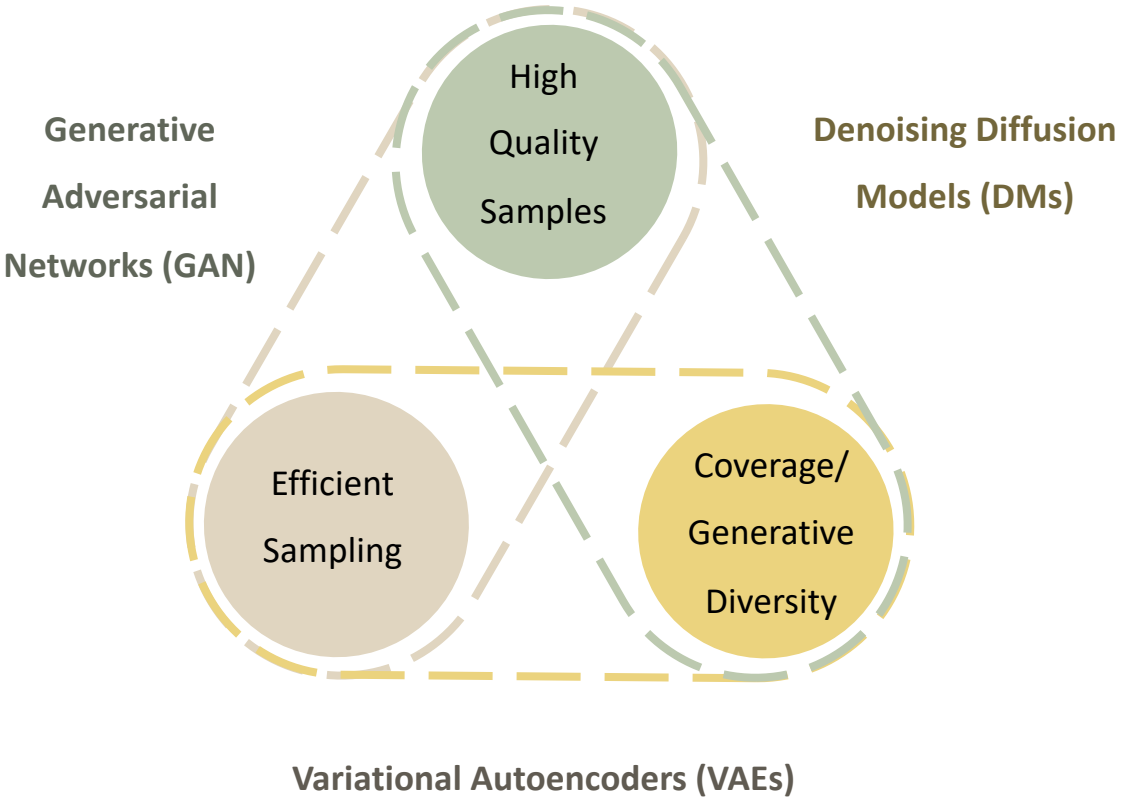
❖ Introduction: Diffusion models

Diffusion models: Gradually add Gaussian noise and then reverse.



Training SD from scratch

❖ Introduction: Generative models

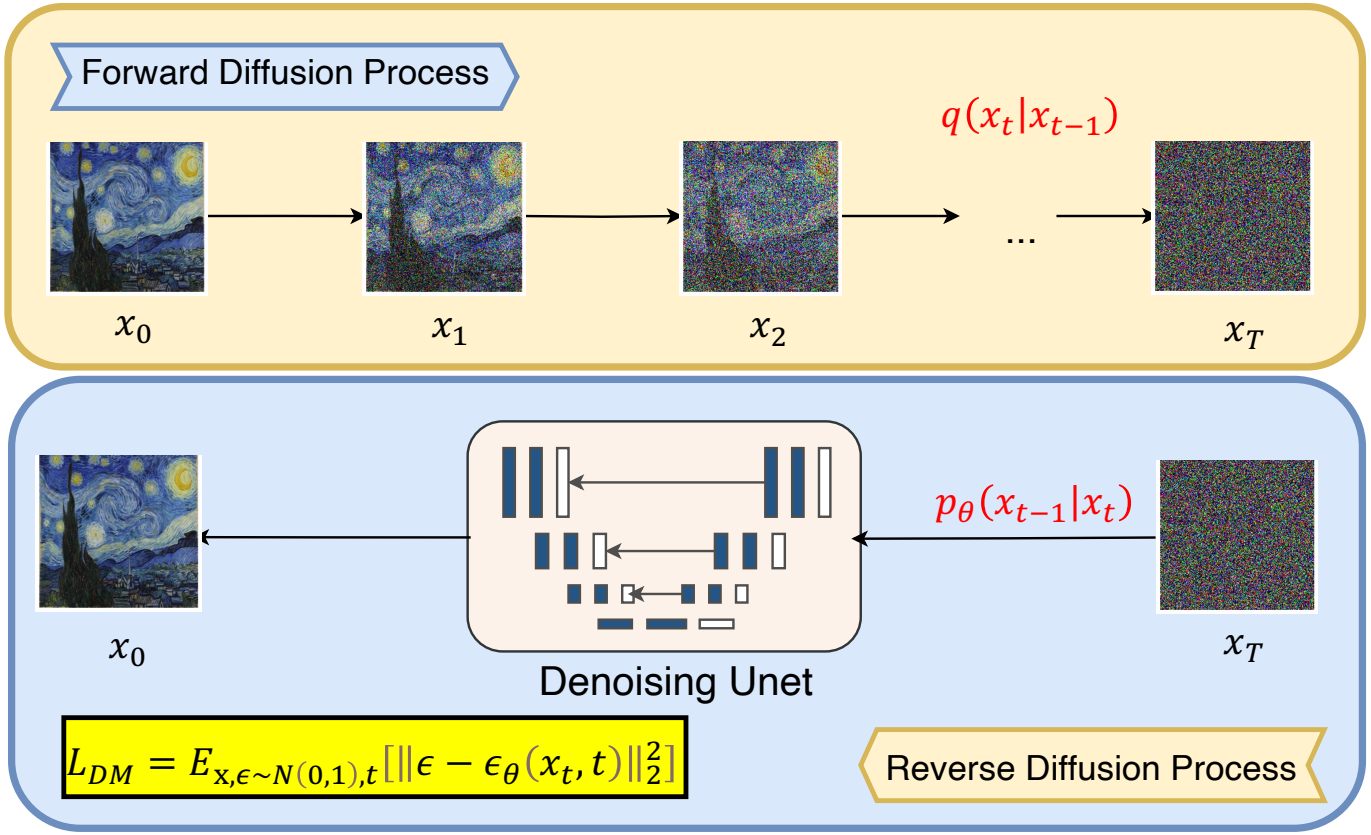


FEATURES	GAN	VAE	DIFFUSION MODEL
IMAGE QUALITY	High-quality, sharp images	Often blurry, less realistic	Very high and realistic
SAMPLE DIVERSITY	Lower	Moderate diversity	High diversity
SAMPLING SPEED	Fast	Fast	Slow

Why?

Training SD from scratch

❖ Introduction: Pure Diffusion Models



Limitations:

- Requires **hundreds of sampling steps.**
- **Resource-intensive** and **very slow.**

This speed issue led to the development of

Stable Diffusion!

Training SD from scratch

❖ Introduction: An example of Stable Diffusion

An original card from the Rider-Waite Tarot deck (1909).
(Source: Wikipedia)

A fantasy-inspired style, drawing from watercolor paintings and mythological elements.*(Generated by AI – Stable Diffusion 3.5)*

Stable diffusion model

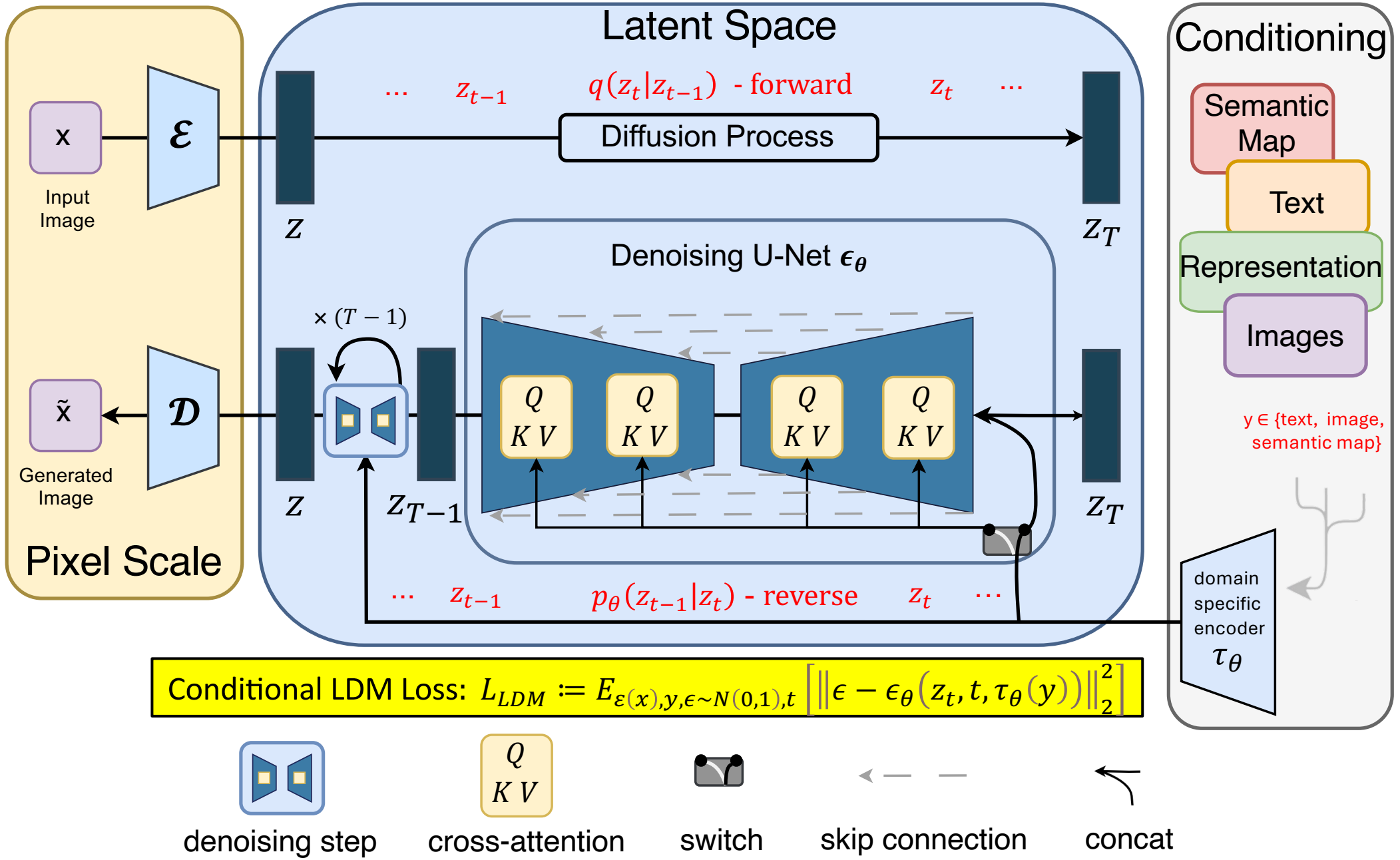
Prompt

"An illustration of the Tarot card *The Empress*, preserving the composition and symbols of the Rider-Waite deck, but reimagined in a fantasy style inspired by watercolor art and mythical elements."

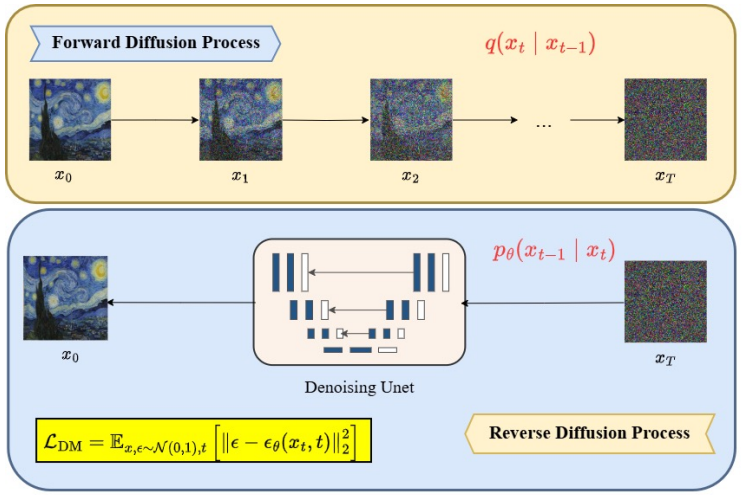
The new illustrated version of the card adopts a fantasy and mythological art style.

(Generated by AI - Stable Diffusion 3.5)

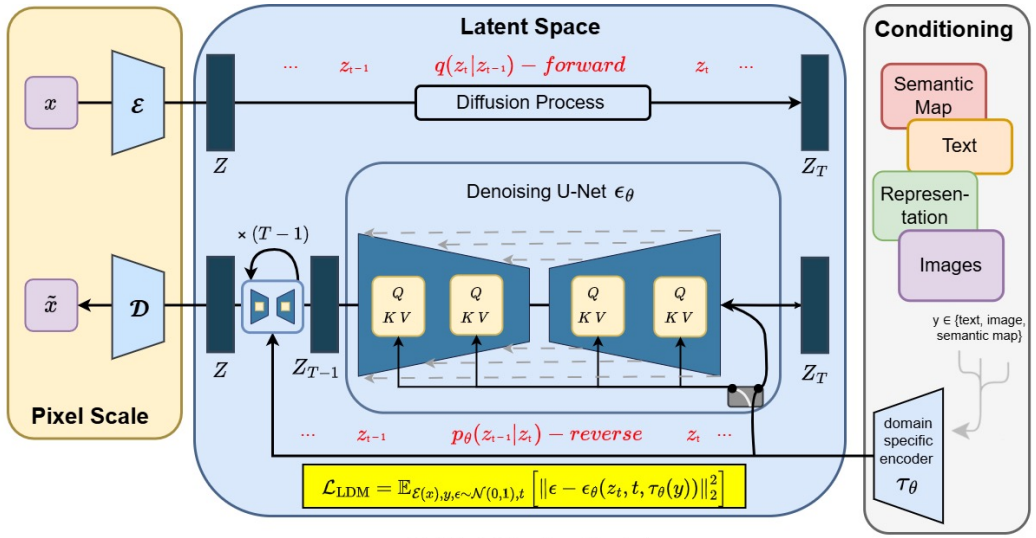
❖ Stable Diffusion Models Architecture



❖ Pure Diffusion Models vs Stable Diffusion Models



Diffusion Models

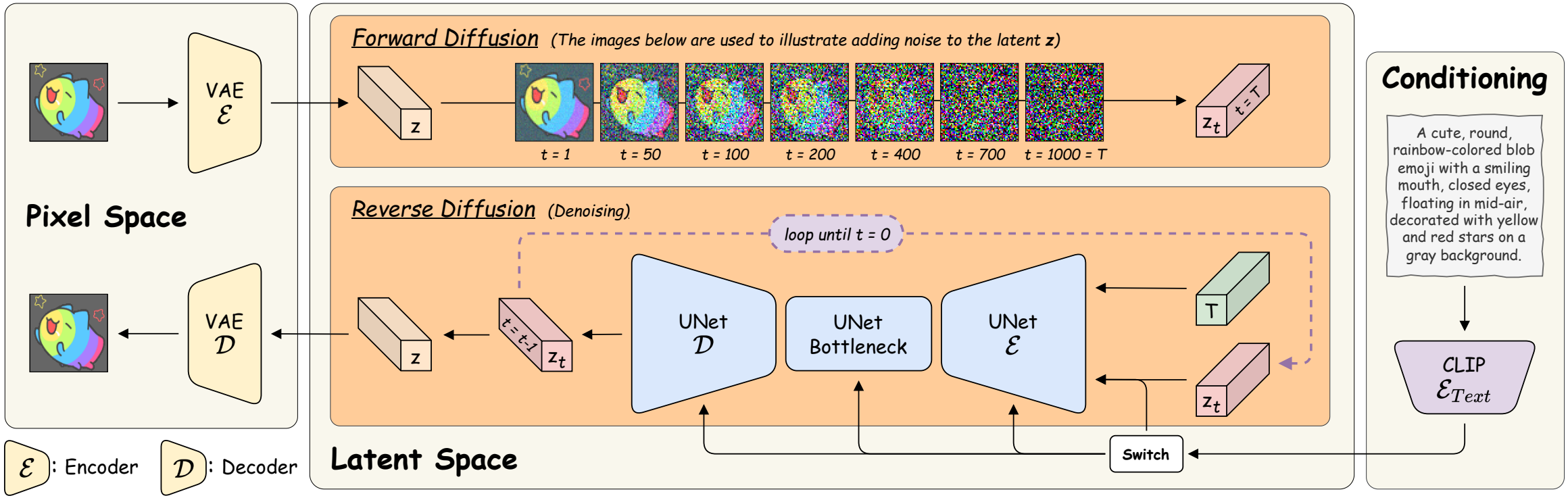


Stable Diffusion Models

Features	Pure Diffusion Models	Stable Diffusion Models
Processing Space	Pixel space (e.g. $x \in R^{H\times W\times 3}$)	Latent space $z \in R^d$ (via VAE encoder)
Inference Speed	Slower	Faster
Architecture Highlight	Denoising U-Net only	Denoising U-Net with Cross-Attention for prompts

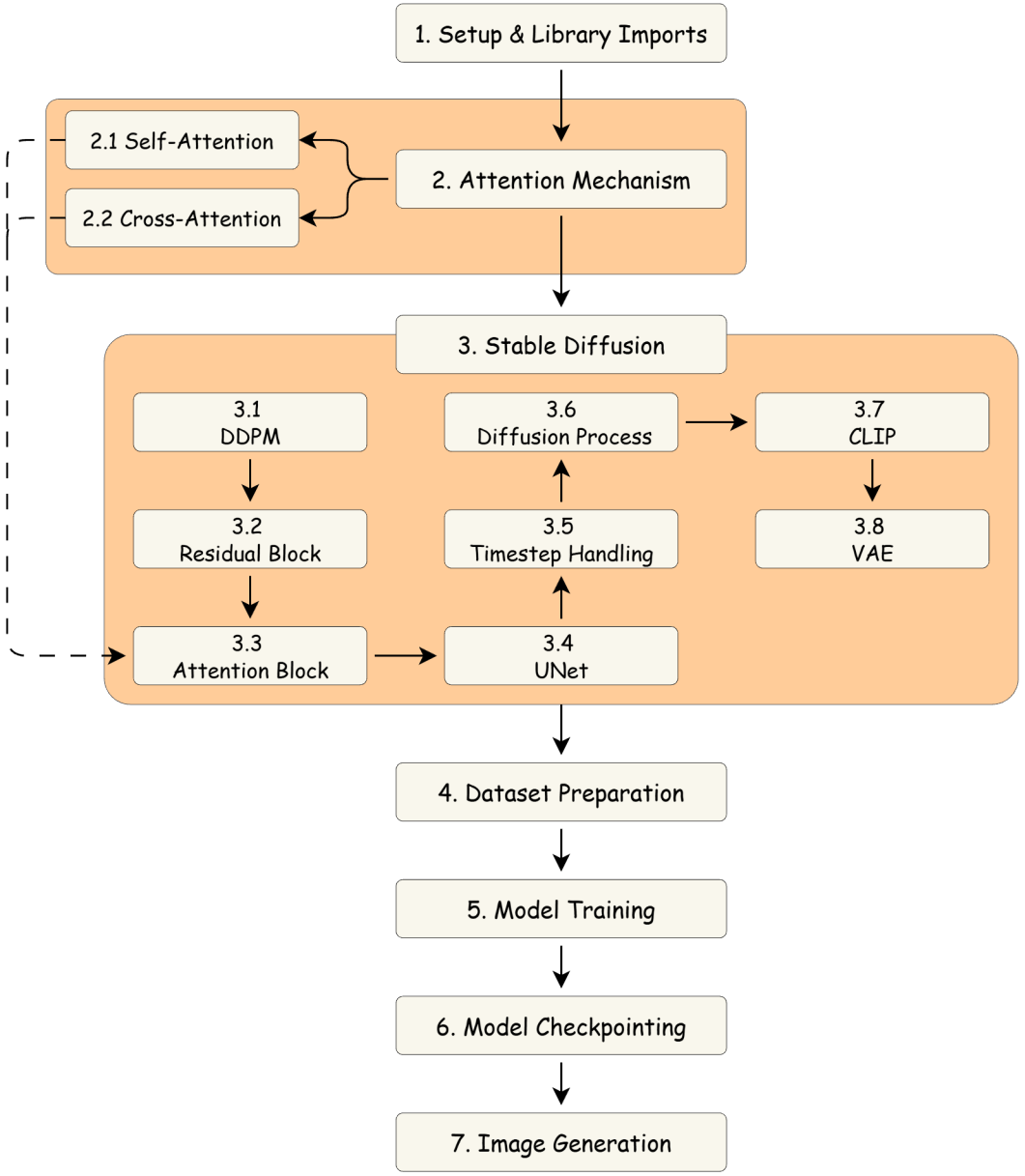
Training SD from scratch

❖ Overall architecture



Training SD from scratch

❖ Workflow



Training SD from scratch

❖ Setup & Library Imports

🤗 Diffusers

🤗 Transformers

```python
import os
import math
import torch
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from torch import nn
from PIL import Image
from tqdm import tqdm
from torchvision import transforms
from diffusers import AutoencoderKL
from torch.nn import functional as F
from torch.amp import GradScaler, autocast
import torch.optim.lr_scheduler as lr_scheduler
from torch.utils.data import Dataset, DataLoader
from transformers import CLIPTokenizer, CLIPTextModel
```

Training SD from scratch

❖ Diffuser

🤗 Diffusers

Popular Tasks & Pipelines

Task	Pipeline	🤗 Hub
Unconditional Image Generation	DDPM	google/ddpm-ema-church-256
Text-to-Image	Stable Diffusion Text-to-Image	stable-diffusion-v1-5/stable-diffusion-v1-5
Text-to-Image	unCLIP	kakaobrain/karlo-v1-alpha
Text-to-Image	DeepFloyd IF	DeepFloyd/IF-I-XL-v1.0
Text-to-Image	Kandinsky	kandinsky-community/kandinsky-2-2-decoder
Text-guided Image-to-Image	ControlNet	lllyasviel/sd-controlnet-canny
Text-guided Image-to-Image	InstructPix2Pix	timbrooks/instruct-pix2pix
Text-guided Image-to-Image	Stable Diffusion Image-to-Image	stable-diffusion-v1-5/stable-diffusion-v1-5
Text-guided Image Inpainting	Stable Diffusion Inpainting	runwayml/stable-diffusion-inpainting
Image Variation	Stable Diffusion Image Variation	lambdalabs/sd-image-variations-diffusers
Super Resolution	Stable Diffusion Upscale	stabilityai/stable-diffusion-x4-upscaler
Super Resolution	Stable Diffusion Latent Upscale	stabilityai/sd-x2-latent-upscaler

https://github.com/huggingface/diffusers

InstructPix2Pix

https://huggingface.co/timbrooks/instruct-pix2pix

x4 Upscaler

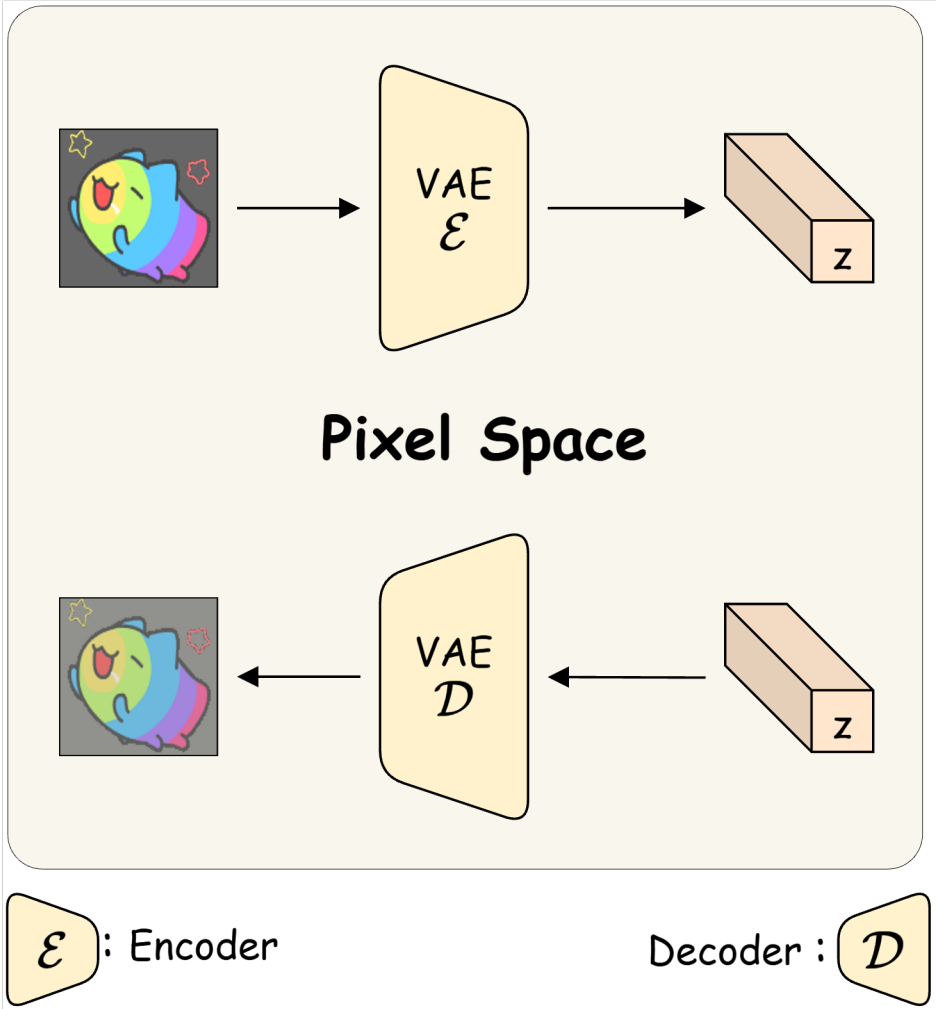
https://huggingface.co/stabilityai/stable-diffusion-x4-upscaler

Stable Diffusion Image Variations

https://huggingface.co/lambdalabs/sd-image-variations-diffusers

Training SD from scratch

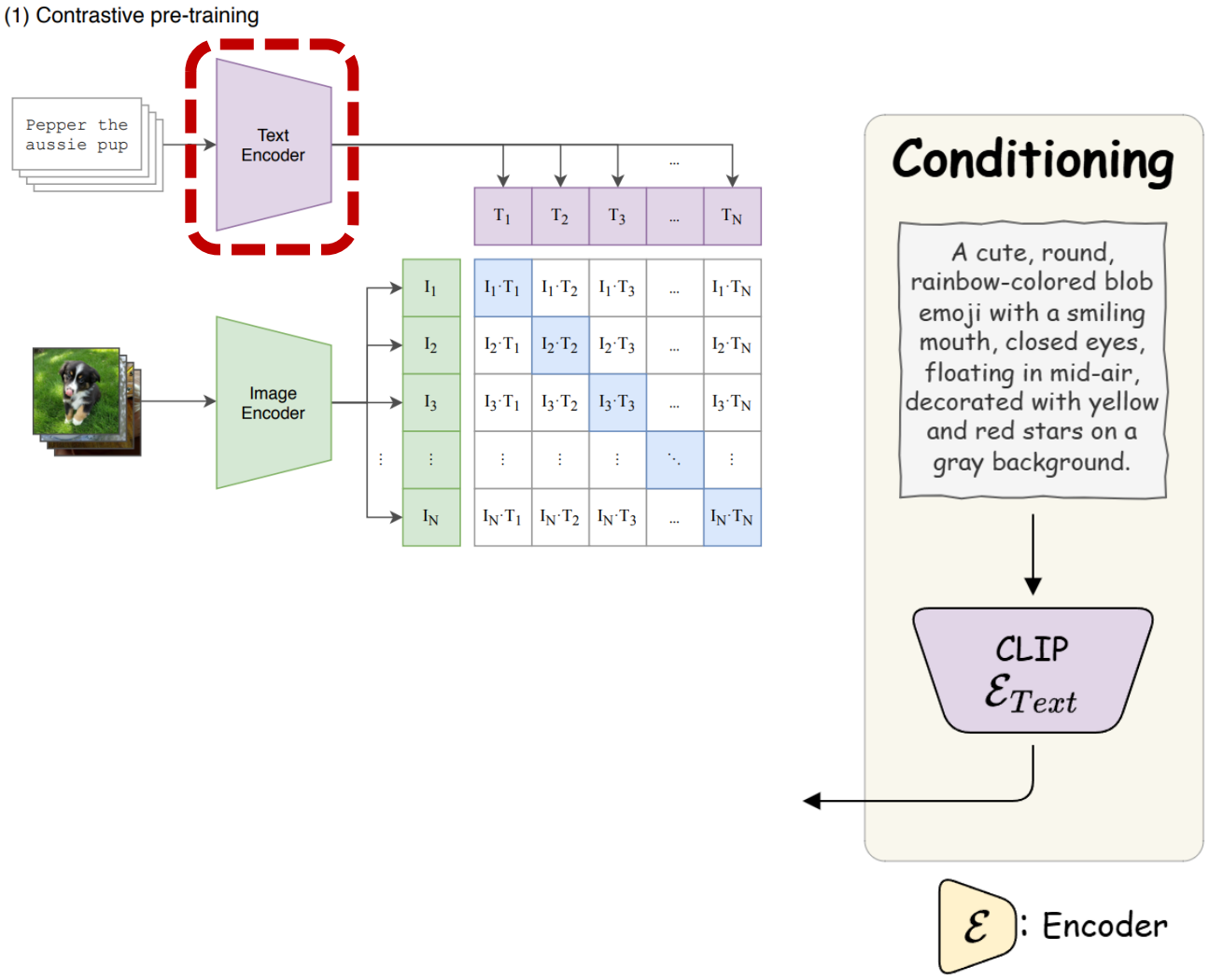
❖ VAE Review




```
1 VAE_id = "stabilityai/sd-vae-ft-mse"
2 vae = AutoencoderKL.from_pretrained(VAE_id)
3 vae.requires_grad_(False)
4 vae.eval()
```

Training SD from scratch

❖ CLIP Review




```python
class CLIPTextEncoder(nn.Module):
    def __init__(self):
        super().__init__()
        CLIP_id = "openai/clip-vit-base-patch32"
        self.tokenizer = CLIPTokenizer.from_pretrained(CLIP_id)
        self.text_encoder = CLIPTextModel.from_pretrained(CLIP_id)
        self.device = "cuda" if torch.cuda.is_available() else "cpu"

        for param in self.text_encoder.parameters():
            param.requires_grad = False

        self.text_encoder.eval()
        self.text_encoder.to(self.device)

    def forward(self, prompts):
        inputs = self.tokenizer(
            prompts,
            padding="max_length",
            truncation=True,
            max_length=self.text_encoder.config.max_position_embeddings,
            return_tensors="pt"
        )
        input_ids = inputs.input_ids.to(self.device)
        attention_mask = inputs.attention_mask.to(self.device)

        with torch.no_grad():
            text_encoder_output = self.text_encoder(
                input_ids=input_ids,
                attention_mask=attention_mask
            )
        last_hidden_states = text_encoder_output.last_hidden_state

        return last_hidden_states
```

Training SD from scratch

❖ DDMP

β_t

```python
class DDPMScheduler:
    def __init__(
        self,
        random_generator,
        train_timesteps=1000,
        diffusion_beta_start=0.00085,
        diffusion_beta_end=0.012
    ):

        self.betas = torch.linspace(
            diffusion_beta_start ** 0.5, diffusion_beta_end ** 0.5,
            train_timesteps, dtype=torch.float32) ** 2
        self.alphas = 1.0 - self.betas
        self.alphas_cumulative_product = torch.cumprod(self.alphas, dim=0)
        self.one_val = torch.tensor(1.0)
        self.prng_generator = random_generator
        self.total_train_timesteps = train_timesteps
        self.schedule_timesteps = torch.from_numpy(
            np.arange(0, train_timesteps)[::-1].copy())

    def set_steps(self, num_sampling_steps=50):
        self.num_sampling_steps = num_sampling_steps
        step_scaling_factor = self.total_train_timesteps // self.num_sampling_steps
        timesteps_for_sampling = (
            np.arange(0, num_sampling_steps) * step_scaling_factor
        ).round()[::-1].copy().astype(np.int64)
        self.schedule_timesteps = torch.from_numpy(timesteps_for_sampling)

    def _get_prior_timestep(self, current_timestep):
        previous_t = current_timestep - self.total_train_timesteps
        previous_t = previous_t // self.num_sampling_steps
        return previous_t
```

Training SD from scratch

❖ DDMP

$$\alpha_t = 1 - \beta_t$$

```python
class DDPMScheduler:
    def __init__(
        self,
        random_generator,
        train_timesteps=1000,
        diffusion_beta_start=0.00085,
        diffusion_beta_end=0.012
    ):

        self.betas = torch.linspace(
            diffusion_beta_start ** 0.5, diffusion_beta_end ** 0.5,
            train_timesteps, dtype=torch.float32) ** 2
        self.alphas = 1.0 - self.betas
        self.alphas_cumulative_product = torch.cumprod(self.alphas, dim=0)
        self.one_val = torch.tensor(1.0)
        self.prng_generator = random_generator
        self.total_train_timesteps = train_timesteps
        self.schedule_timesteps = torch.from_numpy(
            np.arange(0, train_timesteps)[::-1].copy())

    def set_steps(self, num_sampling_steps=50):
        self.num_sampling_steps = num_sampling_steps
        step_scaling_factor = self.total_train_timesteps // self.num_sampling_steps
        timesteps_for_sampling = (
            np.arange(0, num_sampling_steps) * step_scaling_factor
        ).round()[::-1].copy().astype(np.int64)
        self.schedule_timesteps = torch.from_numpy(timesteps_for_sampling)

    def _get_prior_timestep(self, current_timestep):
        previous_t = current_timestep - self.total_train_timesteps
        previous_t = previous_t // self.num_sampling_steps
        return previous_t
```

Training SD from scratch

❖ DDMP

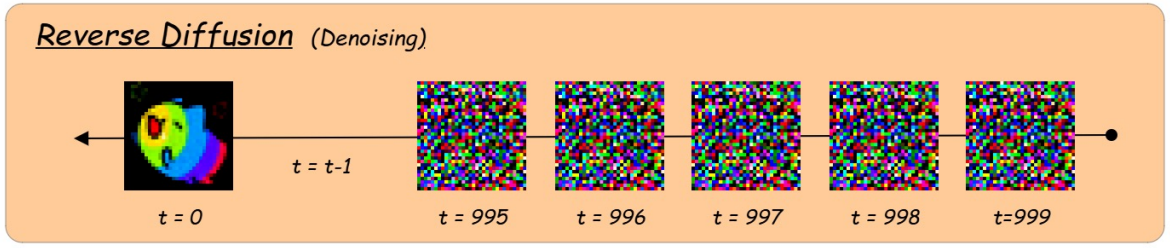
$$\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$$

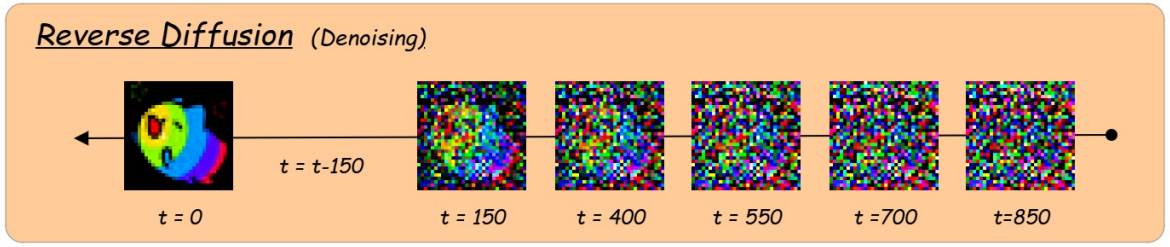
```python
class DDPMScheduler:
    def __init__(
        self,
        random_generator,
        train_timesteps=1000,
        diffusion_beta_start=0.00085,
        diffusion_beta_end=0.012
    ):

        self.betas = torch.linspace(
            diffusion_beta_start ** 0.5, diffusion_beta_end ** 0.5,
            train_timesteps, dtype=torch.float32) ** 2
        self.alphas = 1.0 - self.betas
        self.alphas_cumulative_product = torch.cumprod(self.alphas, dim=0)
        self.one_val = torch.tensor(1.0)
        self.prng_generator = random_generator
        self.total_train_timesteps = train_timesteps
        self.schedule_timesteps = torch.from_numpy(
            np.arange(0, train_timesteps)[::-1].copy())

    def set_steps(self, num_sampling_steps=50):
        self.num_sampling_steps = num_sampling_steps
        step_scaling_factor = self.total_train_timesteps // self.num_sampling_steps
        timesteps_for_sampling = (
            np.arange(0, num_sampling_steps) * step_scaling_factor
        ).round()[::-1].copy().astype(np.int64)
        self.schedule_timesteps = torch.from_numpy(timesteps_for_sampling)

    def _get_prior_timestep(self, current_timestep):
        previous_t = current_timestep - self.total_train_timesteps
        previous_t = previous_t // self.num_sampling_steps
        return previous_t
```

AI VIET NAM
@aivietnam.edu.vn

Training SD from scratch

❖ DDMP






```python
class DDPMScheduler:
    def __init__(
        self,
        random_generator,
        train_timesteps=1000,
        diffusion_beta_start=0.00085,
        diffusion_beta_end=0.012
    ):

        self.betas = torch.linspace(
            diffusion_beta_start ** 0.5, diffusion_beta_end ** 0.5,
            train_timesteps, dtype=torch.float32) ** 2
        self.alphas = 1.0 - self.betas
        self.alphas_cumulative_product = torch.cumprod(self.alphas, dim=0)
        self.one_val = torch.tensor(1.0)
        self.prng_generator = random_generator
        self.total_train_timesteps = train_timesteps
        self.schedule_timesteps = torch.from_numpy(
            np.arange(0, train_timesteps)[::-1].copy())

    def set_steps(self, num_sampling_steps=50):
        self.num_sampling_steps = num_sampling_steps
        step_scaling_factor = self.total_train_timesteps // self.num_sampling_steps
        timesteps_for_sampling = (
            np.arange(0, num_sampling_steps) * step_scaling_factor
        ).round()[::-1].copy().astype(np.int64)
        self.schedule_timesteps = torch.from_numpy(timesteps_for_sampling)

    def _get_prior_timestep(self, current_timestep):
        previous_t = current_timestep - self.total_train_timesteps
        previous_t = previous_t // self.num_sampling_steps
        return previous_t
```

Training SD from scratch

❖ DDMP

$$\sigma_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t$$

```
34 def _calculate_variance(self, timestep):
35     prev_t = self._get_prior_timestep(timestep)
36     alpha_cumprod_t = self.alphas_cumulative_product[timestep]
37     alpha_cumprod_t_prev = self.alphas_cumulative_product[prev_t] if prev_t >= 0 else self.one_val
38     beta_t_current = 1 - alpha_cumprod_t / alpha_cumprod_t_prev
39     variance_value = (1 - alpha_cumprod_t_prev) / (1 - alpha_cumprod_t) * beta_t_current
40     variance_value = torch.clamp(variance_value, min=1e-20)
41     return variance_value
```

Training SD from scratch

❖ DDMP

$$x_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1-\bar{\alpha}_t}\epsilon_\theta)$$

```python
def step(self, current_t, current_latents, model_prediction):
    t = current_t
    prev_t = self._get_prior_timestep(t)

    alpha_cumprod_t = self.alphas_cumulative_product[t]
    alpha_cumprod_t_prev = self.alphas_cumulative_product[prev_t] if prev_t >= 0 else self.one_val
    beta_cumprod_t = 1 - alpha_cumprod_t
    beta_cumprod_t_prev = 1 - alpha_cumprod_t_prev
    alpha_t_current = alpha_cumprod_t / alpha_cumprod_t_prev
    beta_t_current = 1 - alpha_t_current

    predicted_ori = (
        cur_latents - beta_cumprod_t ** 0.5 * model_prediction
        ) / alpha_cumprod_t ** 0.5

    ori_coeff = (alpha_cumprod_t_prev ** 0.5 * beta_t_current) / beta_cumprod_t
    cur_coeff = alpha_t_current ** 0.5 * beta_cumprod_t_prev / beta_cumprod_t

    predicted_prior_mean = ori_coeff * predicted_ori + cur_coeff * cur_latents

    variance_term = 0
    if t > 0:
        target_device = model_prediction.device
        noise_component = torch.randn(
            model_prediction.shape,
            generator=self.prng_generator,
            device=target_device,
            dtype=model_prediction.dtype
        )
        variance_term = (self._calculate_variance(t) ** 0.5) * noise_component

    predicted_prior_sample = predicted_prior_mean + variance_term
    return predicted_prior_sample
```

Training SD from scratch

❖ DDMP

$$\mu_t = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} x_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t} x_0$$

$$\alpha_t = 1 - \beta_t$$
$$\beta_t = 1 - \alpha_t$$

```python
def step(self, current_t, current_latents, model_prediction):
    t = current_t
    prev_t = self._get_prior_timestep(t)

    alpha_cumprod_t = self.alphas_cumulative_product[t]
    alpha_cumprod_t_prev = self.alphas_cumulative_product[prev_t] if prev_t >= 0 else self.one_val
    beta_cumprod_t = 1 - alpha_cumprod_t
    beta_cumprod_t_prev = 1 - alpha_cumprod_t_prev
    alpha_t_current = alpha_cumprod_t / alpha_cumprod_t_prev
    beta_t_current = 1 - alpha_t_current

    predicted_ori = (
        cur_latents - beta_cumprod_t ** 0.5 * model_prediction
        ) / alpha_cumprod_t ** 0.5

    ori_coeff = (alpha_cumprod_t_prev ** 0.5 * beta_t_current) / beta_cumprod_t
    cur_coeff = alpha_t_current ** 0.5 * beta_cumprod_t_prev / beta_cumprod_t

    predicted_prior_mean = ori_coeff * predicted_ori + cur_coeff * cur_latents

    variance_term = 0
    if t > 0:
        target_device = model_prediction.device
        noise_component = torch.randn(
            model_prediction.shape,
            generator=self.prng_generator,
            device=target_device,
            dtype=model_prediction.dtype
        )
        variance_term = (self._calculate_variance(t) ** 0.5) * noise_component

    predicted_prior_sample = predicted_prior_mean + variance_term
    return predicted_prior_sample
```

Training SD from scratch

❖ DDMP

$$\sqrt{\beta_t}\epsilon$$

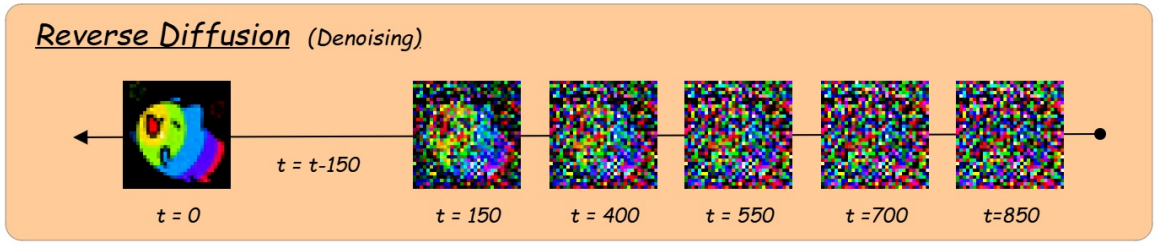
```python
def step(self, current_t, current_latents, model_prediction):
    t = current_t
    prev_t = self._get_prior_timestep(t)

    alpha_cumprod_t = self.alphas_cumulative_product[t]
    alpha_cumprod_t_prev = self.alphas_cumulative_product[prev_t] if prev_t >= 0 else self.one_val
    beta_cumprod_t = 1 - alpha_cumprod_t
    beta_cumprod_t_prev = 1 - alpha_cumprod_t_prev
    alpha_t_current = alpha_cumprod_t / alpha_cumprod_t_prev
    beta_t_current = 1 - alpha_t_current

    predicted_ori = (
        cur_latents - beta_cumprod_t ** 0.5 * model_prediction
        ) / alpha_cumprod_t ** 0.5

    ori_coeff = (alpha_cumprod_t_prev ** 0.5 * beta_t_current) / beta_cumprod_t
    cur_coeff = alpha_t_current ** 0.5 * beta_cumprod_t_prev / beta_cumprod_t

    predicted_prior_mean = ori_coeff * predicted_ori + cur_coeff * cur_latents

    variance_term = 0
    if t > 0:
        target_device = model_prediction.device
        noise_component = torch.randn(
            model_prediction.shape,
            generator=self.prng_generator,
            device=target_device,
            dtype=model_prediction.dtype
        )
        variance_term = (self._calculate_variance(t) ** 0.5) * noise_component

    predicted_prior_sample = predicted_prior_mean + variance_term
    return predicted_prior_sample
```

Training SD from scratch

❖ DDMP

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta\right) + \sqrt{\beta_t}\epsilon$$

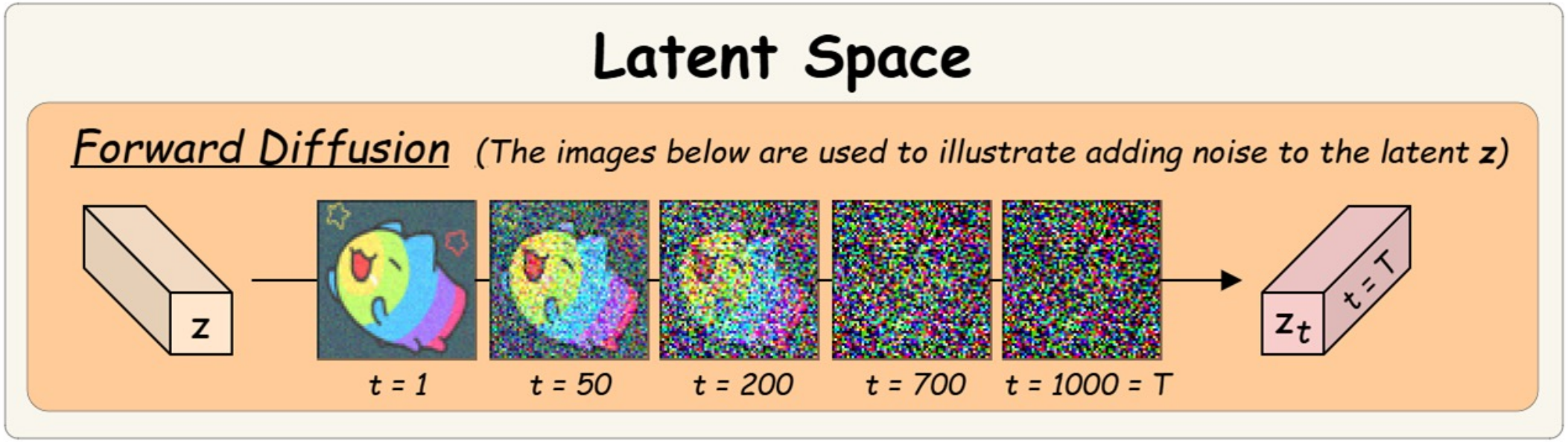
$$\epsilon \sim \mathcal{N}(0,1)$$




```python
def step(self, current_t, current_latents, model_prediction):
    t = current_t
    prev_t = self._get_prior_timestep(t)

    alpha_cumprod_t = self.alphas_cumulative_product[t]
    alpha_cumprod_t_prev = self.alphas_cumulative_product[prev_t] if prev_t >= 0 else self.one_val
    beta_cumprod_t = 1 - alpha_cumprod_t
    beta_cumprod_t_prev = 1 - alpha_cumprod_t_prev
    alpha_t_current = alpha_cumprod_t / alpha_cumprod_t_prev
    beta_t_current = 1 - alpha_t_current

    predicted_ori = (
        cur_latents - beta_cumprod_t ** 0.5 * model_prediction
        ) / alpha_cumprod_t ** 0.5

    ori_coeff = (alpha_cumprod_t_prev ** 0.5 * beta_t_current) / beta_cumprod_t
    cur_coeff = alpha_t_current ** 0.5 * beta_cumprod_t_prev / beta_cumprod_t

    predicted_prior_mean = ori_coeff * predicted_ori + cur_coeff * cur_latents

    variance_term = 0
    if t > 0:
        target_device = model_prediction.device
        noise_component = torch.randn(
            model_prediction.shape,
            generator=self.prng_generator,
            device=target_device,
            dtype=model_prediction.dtype
        )
        variance_term = (self._calculate_variance(t) ** 0.5) * noise_component

    predicted_prior_sample = predicted_prior_mean + variance_term
    return predicted_prior_sample
```

Training SD from scratch

❖ DDMP



$$q(x_t \mid x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I) = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon_t$$

Training SD from scratch

❖ DDMP

$$q(x_t \mid x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I) = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon_t$$

```
77 def add_noise(self, initial_samples, noise_timesteps):
78     alphas_cumprod = self.alphas_cumulative_product.to(
79         device=initial_samples.device,
80         dtype=initial_samples.dtype
81     )
82     noise_timesteps = noise_timesteps.to(initial_samples.device)
83     sqrt_alpha_cumprod = alphas_cumprod[noise_timesteps] ** 0.5
84     sqrt_alpha_cumprod = sqrt_alpha_cumprod.view(
85         sqrt_alpha_cumprod.shape[0], *([1] * (initial_samples.ndim - 1))
86     )
87     sqrt_one_minus_alpha_cumprod = (1 - alphas_cumprod[noise_timesteps]) ** 0.5
88     sqrt_one_minus_alpha_cumprod = sqrt_one_minus_alpha_cumprod.view(
89         sqrt_one_minus_alpha_cumprod.shape[0], *([1] * (initial_samples.ndim - 1))
90     )
91     random_noise = torch.randn(
92         initial_samples.shape, generator=self.prng_generator,
93         device=initial_samples.device, dtype=initial_samples.dtype
94     )
95     a = sqrt_alpha_cumprod * initial_samples
96     b = sqrt_one_minus_alpha_cumprod * random_noise
97     noisy_result = a + b
98     return noisy_result, random_noise
```

Training SD from scratch

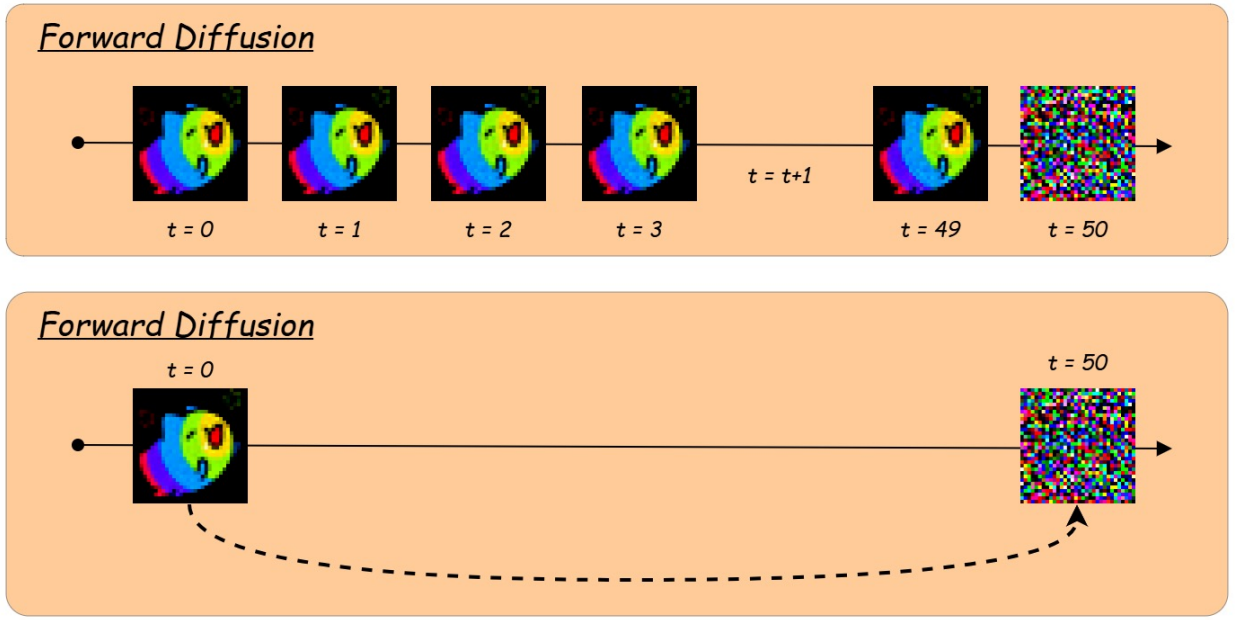
❖ DDMP

$$q(x_t \mid x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I) = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon_t$$

```
77 def add_noise(self, initial_samples, noise_timesteps):
78     alphas_cumprod = self.alphas_cumulative_product.to(
79         device=initial_samples.device,
80         dtype=initial_samples.dtype
81     )
82     noise_timesteps = noise_timesteps.to(initial_samples.device)
83     sqrt_alpha_cumprod = alphas_cumprod[noise_timesteps] ** 0.5
84     sqrt_alpha_cumprod = sqrt_alpha_cumprod.view(
85         sqrt_alpha_cumprod.shape[0], *([1] * (initial_samples.ndim - 1))
86     )
87     sqrt_one_minus_alpha_cumprod = (1 - alphas_cumprod[noise_timesteps]) ** 0.5
88     sqrt_one_minus_alpha_cumprod = sqrt_one_minus_alpha_cumprod.view(
89         sqrt_one_minus_alpha_cumprod.shape[0], *([1] * (initial_samples.ndim - 1))
90     )
91     random_noise = torch.randn(
92         initial_samples.shape, generator=self.prng_generator,
93         device=initial_samples.device, dtype=initial_samples.dtype
94     )
95     a = sqrt_alpha_cumprod * initial_samples
96     b = sqrt_one_minus_alpha_cumprod * random_noise
97     noisy_result = a + b
98     return noisy_result, random_noise
```

Training SD from scratch

❖ DDMP

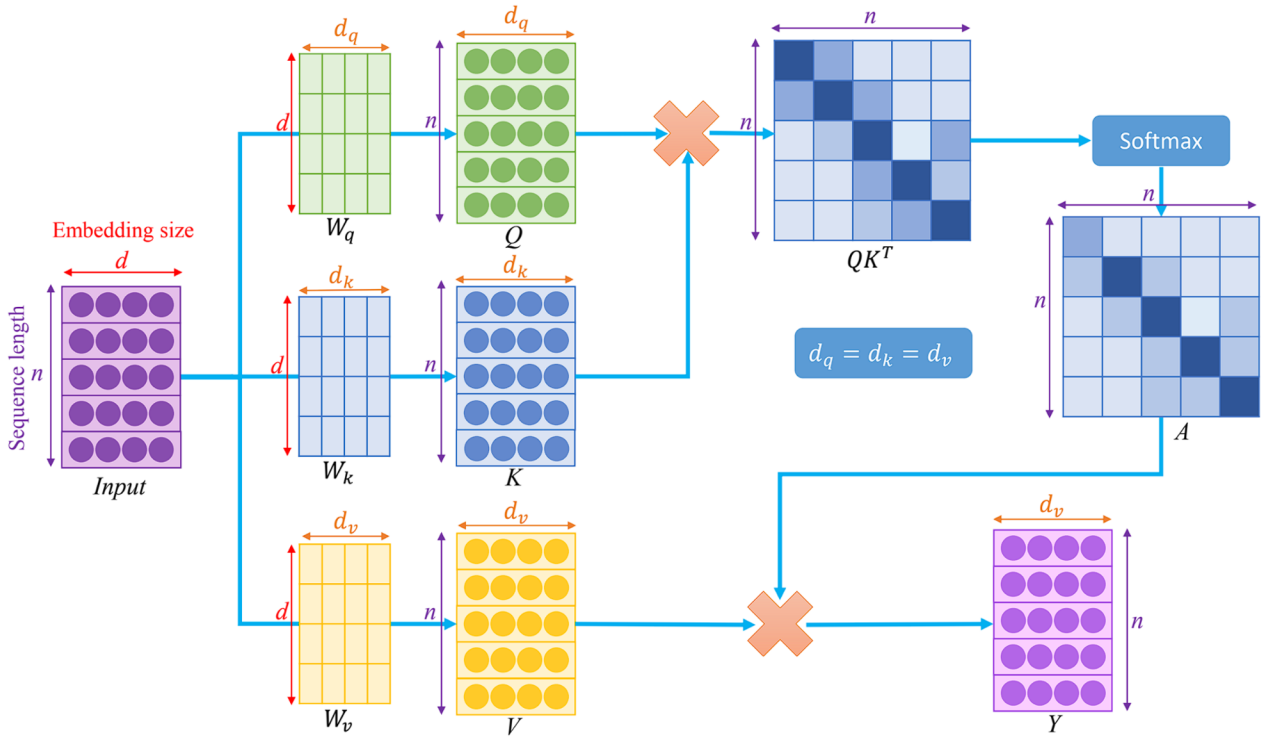


$$\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon_t$$

```python
def add_noise(self, initial_samples, noise_timesteps):
    alphas_cumprod = self.alphas_cumulative_product.to(
        device=initial_samples.device,
        dtype=initial_samples.dtype
    )
    noise_timesteps = noise_timesteps.to(initial_samples.device)
    sqrt_alpha_cumprod = alphas_cumprod[noise_timesteps] ** 0.5
    sqrt_alpha_cumprod = sqrt_alpha_cumprod.view(
        sqrt_alpha_cumprod.shape[0], *([1] * (initial_samples.ndim - 1))
    )
    sqrt_one_minus_alpha_cumprod = (1 - alphas_cumprod[noise_timesteps]) ** 0.5
    sqrt_one_minus_alpha_cumprod = sqrt_one_minus_alpha_cumprod.view(
        sqrt_one_minus_alpha_cumprod.shape[0], *([1] * (initial_samples.ndim - 1))
    )
    random_noise = torch.randn(
        initial_samples.shape, generator=self.prng_generator,
        device=initial_samples.device, dtype=initial_samples.dtype
    )
    a = sqrt_alpha_cumprod * initial_samples
    b = sqrt_one_minus_alpha_cumprod * random_noise
    noisy_result = a + b
    return noisy_result, random_noise
```

Training SD from scratch

❖ Self-Attention Review

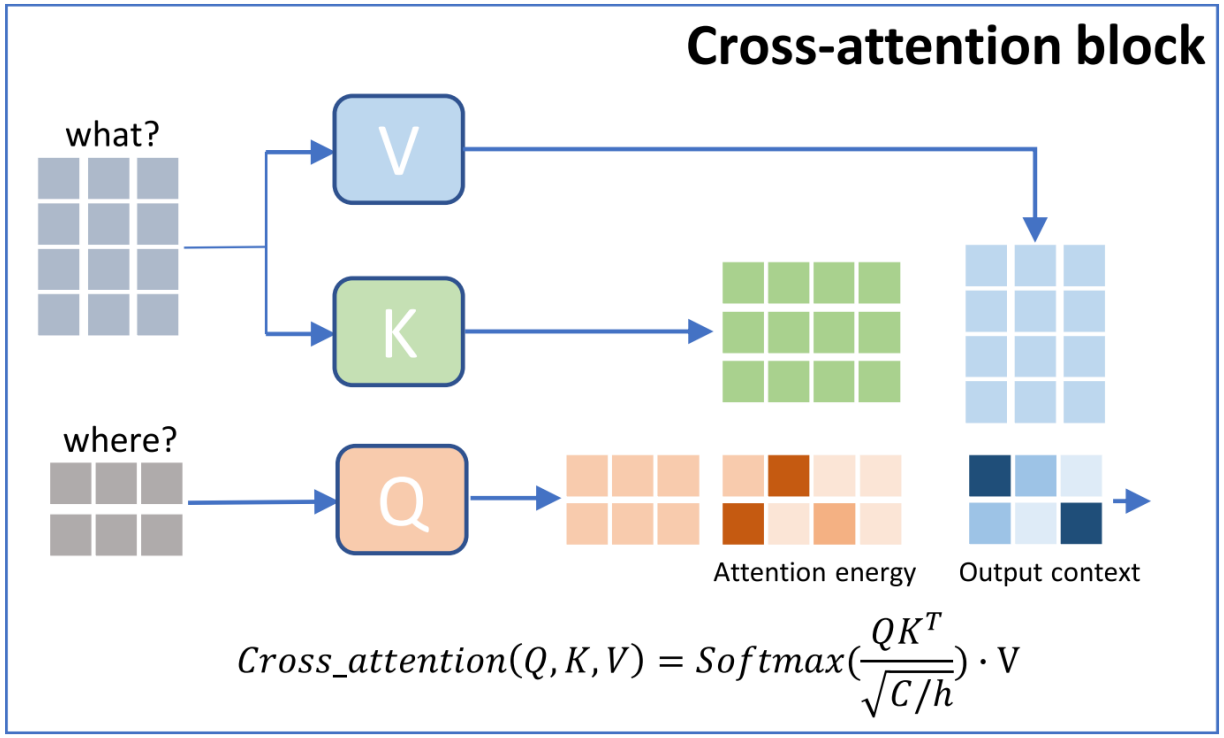


$$Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V$$

```python
class SelfAttention(nn.Module):
    def __init__(self, num_attn_heads, hidden_dim,
                 in_proj_bias=True, out_proj_bias=True):
        super().__init__()
        self.num_heads = num_attn_heads
        self.head_size = hidden_dim // num_attn_heads

        self.qkv_proj = nn.Linear(hidden_dim, 3 * hidden_dim, bias=in_proj_bias)
        self.output_proj = nn.Linear(hidden_dim, hidden_dim, bias=out_proj_bias)

    def forward(self, features, use_causal_mask=False):
        b, s, d = features.shape

        qkv_combined = self.qkv_proj(features)
        q_mat, k_mat, v_mat = torch.chunk(qkv_combined, 3, dim=-1)

        q_mat = q_mat.view(b, s, self.num_heads, self.head_size).permute(0, 2, 1, 3)
        k_mat = k_mat.view(b, s, self.num_heads, self.head_size).permute(0, 2, 1, 3)
        v_mat = v_mat.view(b, s, self.num_heads, self.head_size).permute(0, 2, 1, 3)

        qk = torch.matmul(q_mat, k_mat.transpose(-2, -1))
        sqrt_qk = qk / math.sqrt(self.head_size)

        if use_causal_mask:
            causal_mask = torch.triu(
                torch.ones_like(sqrt_qk, dtype=torch.bool), diagonal=1)
            sqrt_qk = sqrt_qk.masked_fill(causal_mask, -torch.inf)

        attn_weights = torch.softmax(sqrt_qk, dim=-1)
        attn_values = torch.matmul(attn_weights, v_mat)

        attn_values = attn_values.permute(0, 2, 1, 3).contiguous()
        attn_values = attn_values.view(b, s, d)

        final_output = self.output_proj(attn_values)
        return final_output
```

Training SD from scratch

❖ Cross-Attention Review

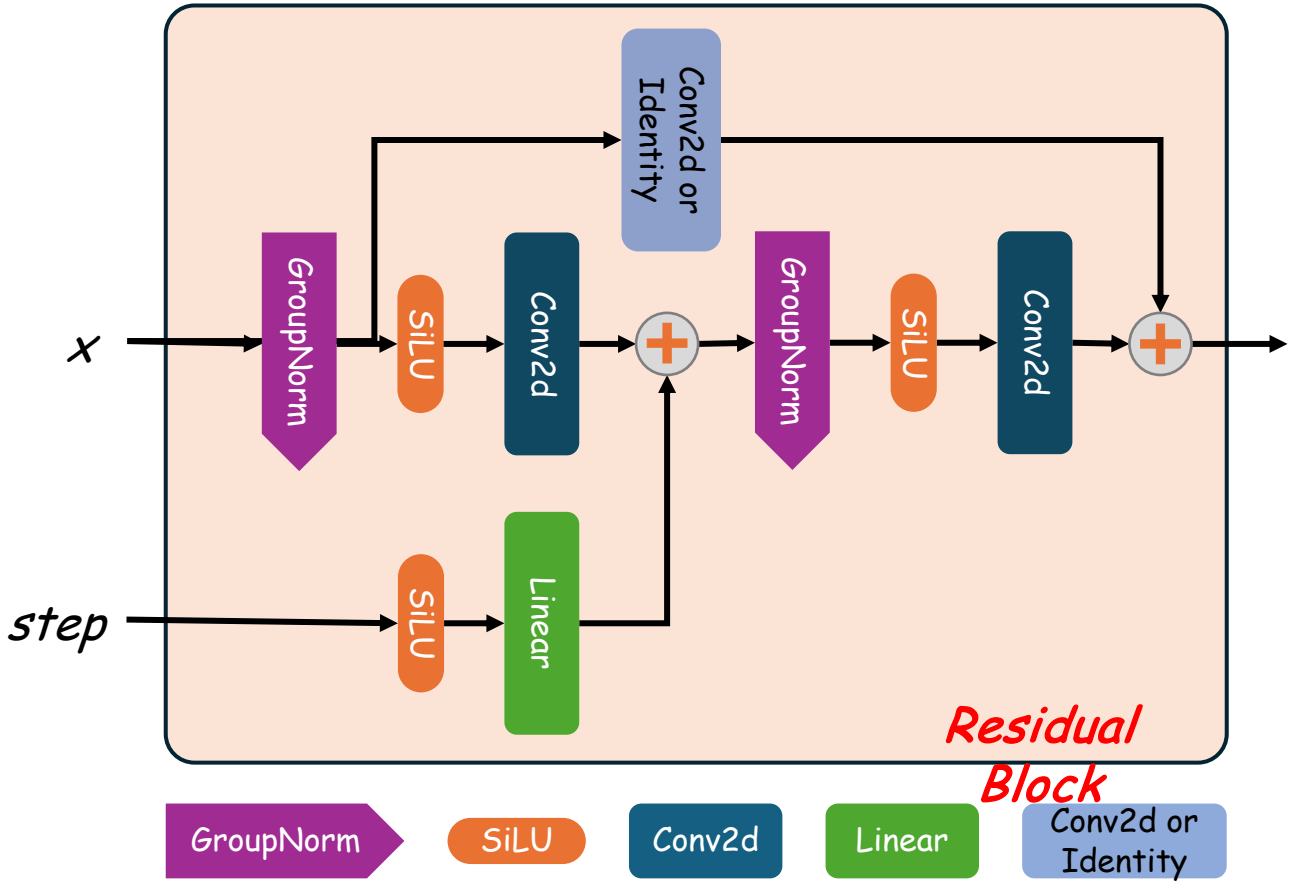



```python
class CrossAttention(nn.Module):
    def __init__(self, num_attn_heads, query_dim, context_dim,
                 in_proj_bias=True, out_proj_bias=True):
        super().__init__()
        self.num_heads = num_attn_heads
        self.head_size = query_dim // num_attn_heads

        self.query_map = nn.Linear(query_dim, query_dim, bias=in_proj_bias)
        self.key_map = nn.Linear(context_dim, query_dim, bias=in_proj_bias)
        self.value_map = nn.Linear(context_dim, query_dim, bias=in_proj_bias)

        self.output_map = nn.Linear(query_dim, query_dim, bias=out_proj_bias)

    def forward(self, query_input, context_input):
        b_q, s_q, d_q = query_input.shape
        _, s_kv, _ = context_input.shape

        q_mat = self.query_map(query_input)
        k_mat = self.key_map(context_input)
        v_mat = self.value_map(context_input)

        q_mat = q_mat.view(b_q, s_q, self.num_heads, self.head_size).permute(0, 2, 1, 3)
        k_mat = k_mat.view(b_q, s_kv, self.num_heads, self.head_size).permute(0, 2, 1, 3)
        v_mat = v_mat.view(b_q, s_kv, self.num_heads, self.head_size).permute(0, 2, 1, 3)

        qk = torch.matmul(q_mat, k_mat.transpose(-2, -1))
        sqrt_qk = qk / math.sqrt(self.head_size)
        attn_weights = torch.softmax(sqrt_qk, dim=-1)

        attn_values = torch.matmul(attn_weights, v_mat)
        attn_values = attn_values.permute(0, 2, 1, 3).contiguous()
        attn_values = attn_values.view(b_q, s_q, d_q)

        final_output = self.output_map(attn_values)
        return final_output
```

https://academic.oup.com/bib/article/25/1/bbad484/7510975

Training SD from scratch

❖ Unet – Residual Block



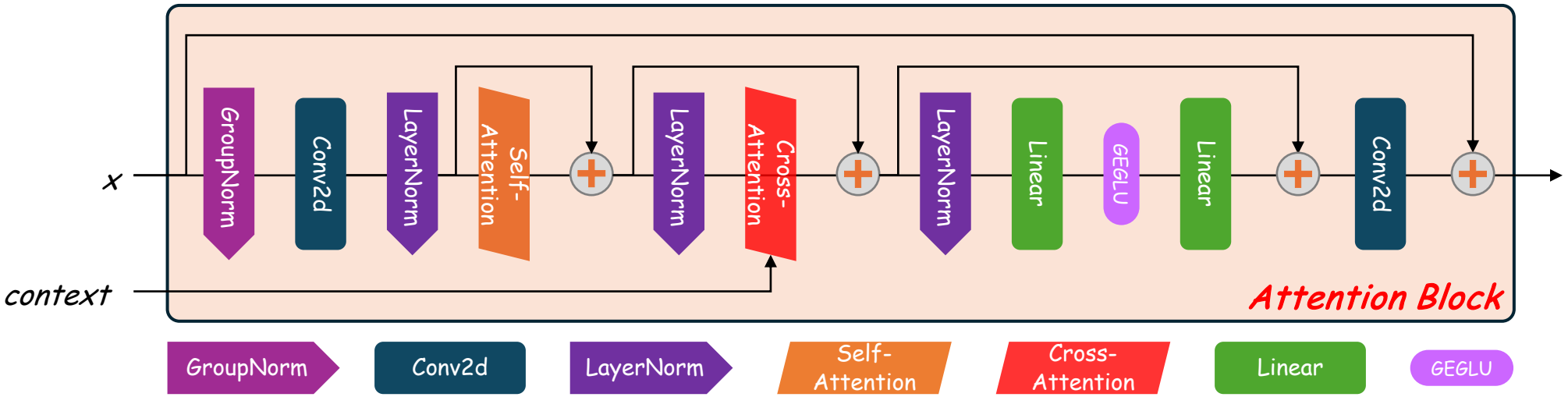

```python
class UNET_ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, time_dim=1280):
        super().__init__()
        self.gn_feature = nn.GroupNorm(32, in_channels)
        self.conv_feature = nn.Conv2d(
            in_channels, out_channels, kernel_size=3, padding=1)
        self.time_embedding_proj = nn.Linear(time_dim, out_channels)

        self.gn_merged = nn.GroupNorm(32, out_channels)
        self.conv_merged = nn.Conv2d(
            out_channels, out_channels, kernel_size=3, padding=1)

        if in_channels == out_channels:
            self.residual_connection = nn.Identity()
        else:
            self.residual_connection = nn.Conv2d(
                in_channels, out_channels, kernel_size=1, padding=0)

    def forward(self, input_feature, time_emb):
        residual = input_feature

        h = self.gn_feature(input_feature)
        h = F.silu(h)
        h = self.conv_feature(h)

        time_emb_processed = F.silu(time_emb)
        time_emb_projected = self.time_embedding_proj(time_emb_processed)
        time_emb_projected = time_emb_projected.unsqueeze(-1).unsqueeze(-1)

        merged_feature = h + time_emb_projected
        merged_feature = self.gn_merged(merged_feature)
        merged_feature = F.silu(merged_feature)
        merged_feature = self.conv_merged(merged_feature)

        output = merged_feature + self.residual_connection(residual)
        return output
```

Training SD from scratch

❖ Unet – Attention Block

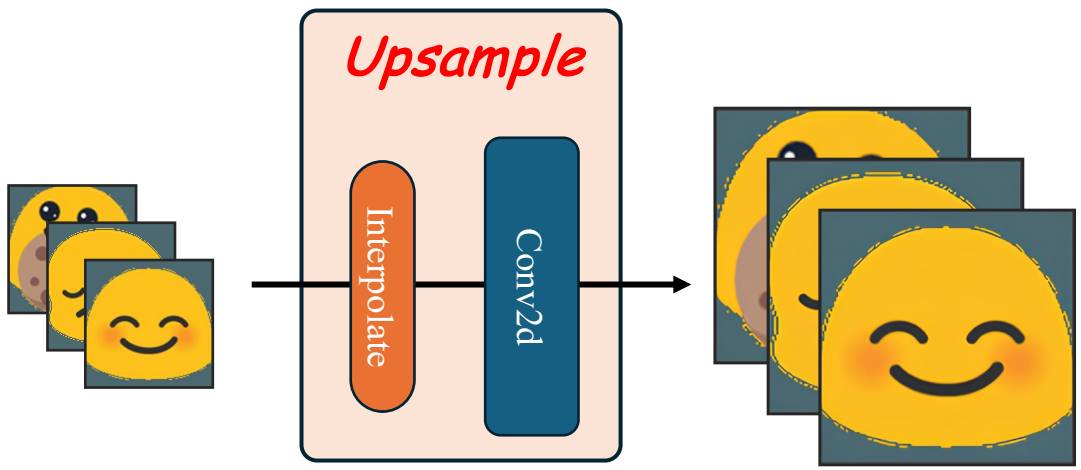
```python
class UNET_AttentionBlock(nn.Module):
    def __init__(self, num_heads, head_dim, context_dim=512):
        super().__init__()
        embed_dim = num_heads * head_dim

        self.gn_in = nn.GroupNorm(32, embed_dim, eps=1e-6)
        self.proj_in = nn.Conv2d(embed_dim, embed_dim, kernel_size=1, padding=0)

        self.ln_1 = nn.LayerNorm(embed_dim)
        self.attn_1 = SelfAttention(num_heads, embed_dim, in_proj_bias=False)
        self.ln_2 = nn.LayerNorm(embed_dim)
        self.attn_2 = CrossAttention(
            num_heads, embed_dim, context_dim, in_proj_bias=False)
        self.ln_3 = nn.LayerNorm(embed_dim)

        self.ffn_geglu = nn.Linear(embed_dim, 4 * embed_dim * 2)
        self.ffn_out = nn.Linear(4 * embed_dim, embed_dim)
        self.proj_out = nn.Conv2d(embed_dim, embed_dim, kernel_size=1, padding=0)
```

```python
def forward(self, input_tensor, context_tensor):
    skip_connection = input_tensor
    B, C, H, W = input_tensor.shape
    HW = H * W

    h = self.proj_in(self.gn_in(input_tensor))
    h = h.view(B, C, HW).transpose(-1, -2)

    attn1_skip = h
    h = self.attn_1(self.ln_1(h)) + attn1_skip

    attn2_skip = h
    h = self.attn_2(self.ln_2(h), context_tensor) + attn2_skip

    ffn_skip = h
    intermediate, gate = self.ffn_geglu(self.ln_3(h)).chunk(2, dim=-1)
    h = self.ffn_out(intermediate * F.gelu(gate)) + ffn_skip

    h = h.transpose(-1, -2).view(B, C, H, W)
    output = self.proj_out(h) + skip_connection
    return output
```



Training SD from scratch

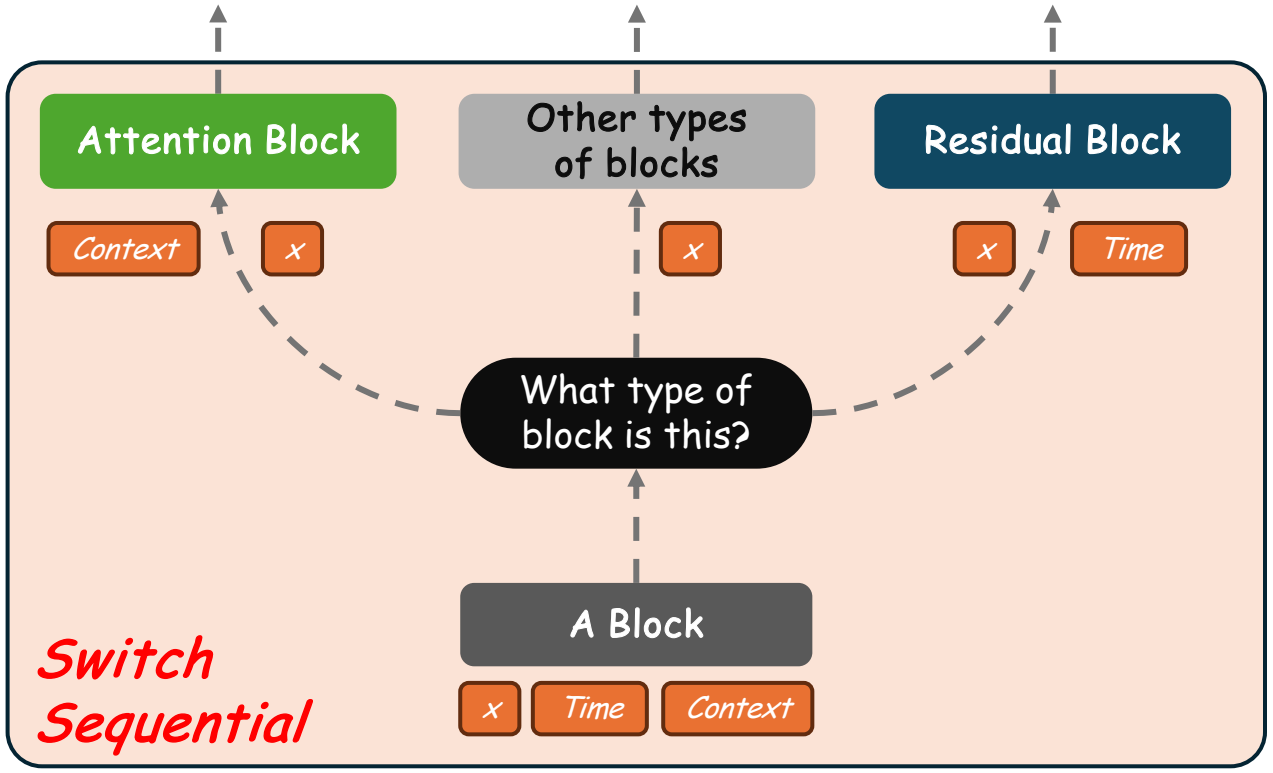
❖ Unet – Upsample Block




```python
class Upsample(nn.Module):
    def __init__(self, num_channels):
        super().__init__()
        self.conv = nn.Conv2d(
            num_channels, num_channels,
            kernel_size=3, padding=1)

    def forward(self, feature_map):
        x = F.interpolate(feature_map,
                          scale_factor=2,
                          mode='nearest')
        x = self.conv(x)
        return x
```

Training SD from scratch

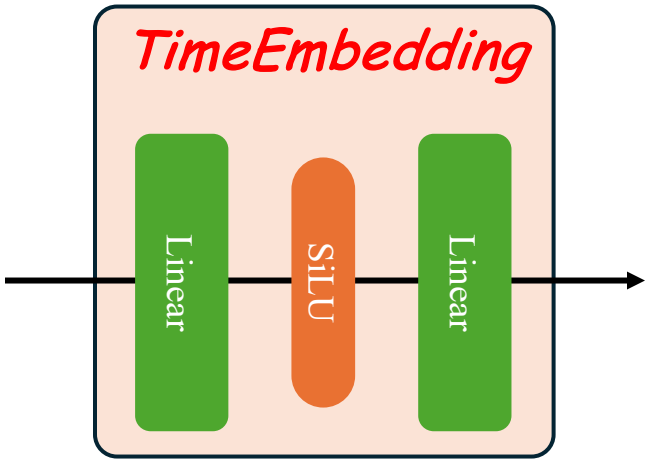
❖ Unet – Switch Sequential Block




```python
class SwitchSequential(nn.Sequential):
  def forward(self, x, guidance_context, time_embedding):
    for module_instance in self:
      if isinstance(module_instance, UNET_AttentionBlock):
        x = module_instance(x, guidance_context)
      elif isinstance(module_instance, UNET_ResidualBlock):
        x = module_instance(x, time_embedding)
      else:
        x = module_instance(x)
    return x
```

Training SD from scratch

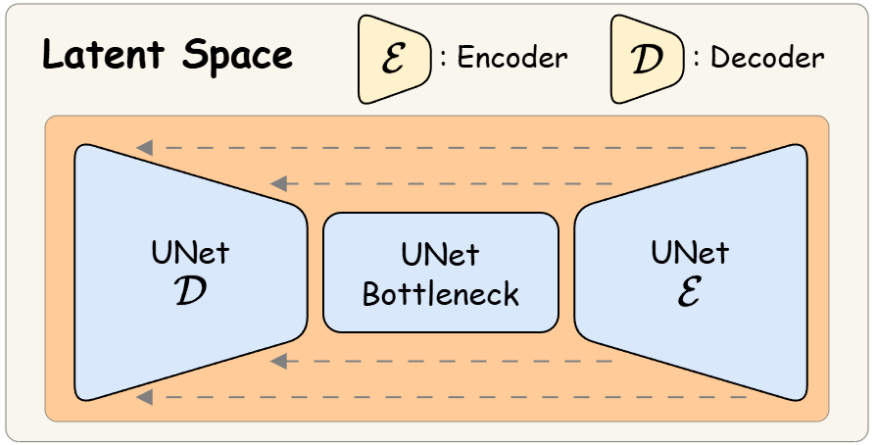
❖ Unet – TimeEmbedding Block




```python
class TimeEmbedding(nn.Module):
    def __init__(self, n_embd):
        super().__init__()
        self.proj1 = nn.Linear(n_embd, 4 * n_embd)
        self.proj2 = nn.Linear(4 * n_embd, 4 * n_embd)

    def forward(self, x):
        x = self.proj1(x)
        x = F.silu(x)
        x = self.proj2(x)
        return x
```

Training SD from scratch

❖ Unet

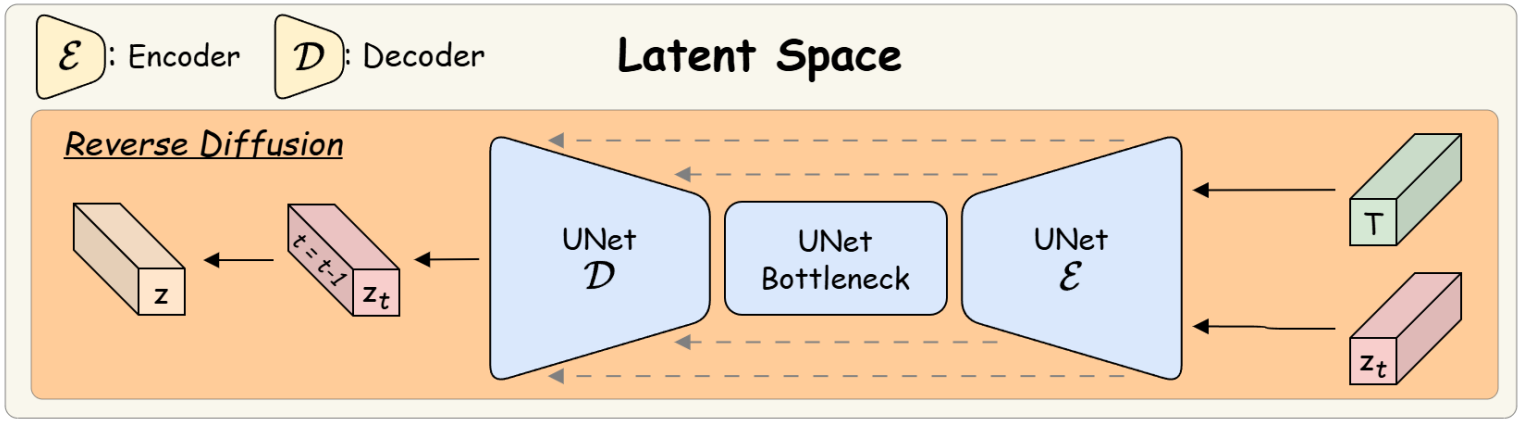



```python
class UNET(nn.Module):
    def __init__(self, h_dim, n_head):
        super().__init__()

        self.down_blocks = nn.ModuleList([
            SwitchSequential(nn.Conv2d(4, h_dim,
                                       kernel_size=3, padding=1)),
            SwitchSequential(
                UNET_ResidualBlock(h_dim, h_dim),
                UNET_AttentionBlock(n_head, (h_dim)//n_head)),
            SwitchSequential(
                UNET_ResidualBlock(h_dim, h_dim),
                UNET_AttentionBlock(n_head, (h_dim)//n_head)),
            SwitchSequential(nn.Conv2d(h_dim, h_dim,
                                       kernel_size=3, stride=2, padding=1)),
            SwitchSequential(UNET_ResidualBlock(h_dim, 2*h_dim)),
            SwitchSequential(UNET_ResidualBlock(2*h_dim, 2*h_dim)),
        ])
```

```python
        self.mid_block = SwitchSequential(
            UNET_ResidualBlock(2*h_dim, 2*h_dim),
            UNET_AttentionBlock(n_head, (2*h_dim)//n_head),
            UNET_ResidualBlock(2*h_dim, 2*h_dim),
        )

        self.up_blocks = nn.ModuleList([
            SwitchSequential(UNET_ResidualBlock(2*h_dim + 2*h_dim, 2*h_dim)),
            SwitchSequential(UNET_ResidualBlock(2*h_dim + 2*h_dim, 2*h_dim)),
            SwitchSequential(
                UNET_ResidualBlock(2*h_dim + h_dim, 2*h_dim),
                Upsample(2*h_dim)),
            SwitchSequential(
                UNET_ResidualBlock(2*h_dim + h_dim, h_dim),
                UNET_AttentionBlock(n_head, h_dim//n_head)),
            SwitchSequential(
                UNET_ResidualBlock(2*h_dim, h_dim),
                UNET_AttentionBlock(n_head, h_dim//n_head)),
            SwitchSequential(
                UNET_ResidualBlock(2*h_dim, h_dim),
                UNET_AttentionBlock(n_head, h_dim//n_head)),
        ])
```

Training SD from scratch

❖ Unet




```python
def forward(self, latent_input, context_embedding, time_embedding):
    down_block_residuals = []
    current_feature_map = latent_input

    for block in self.down_blocks:
        current_feature_map = block(
            current_feature_map, context_embedding, time_embedding)
        down_block_residuals.append(current_feature_map)

    current_feature_map = self.mid_block(
        current_feature_map, context_embedding, time_embedding)

    for block in self.up_blocks:
        residual = down_block_residuals.pop()
        current_feature_map = torch.cat(
            (current_feature_map, residual), dim=1)
        current_feature_map = block(
            current_feature_map, context_embedding, time_embedding)
    return current_feature_map
```

```python
class UNETOutputLayer(nn.Module):
    def __init__(self, input_channels, output_channels):
        super().__init__()
        self.final_groupnorm = nn.GroupNorm(32, input_channels)
        self.final_conv = nn.Conv2d(input_channels, output_channels,
                                    kernel_size=3, padding=1)

    def forward(self, feature_map):
        norm_map = self.final_groupnorm(feature_map)
        activated_map = F.silu(norm_map)
        output_map = self.final_conv(activated_map)

        return output_map
```

Training SD from scratch

❖ Timestep Handling

fixed

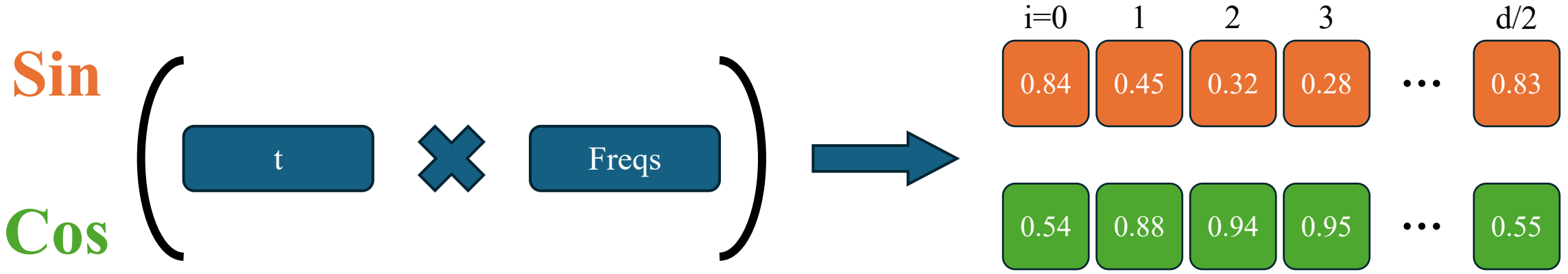
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

$$\text{Freq} = \frac{1}{10000^{2i/d}} = e^{-log(10000^{2i/d})} = e^{-\frac{2i}{d}log(10000)}, \text{where } i \in [0, \frac{d}{2}]$$


```python
def embed_timesteps(timesteps, embedding_dim=320):
    half_dim = embedding_dim // 2
    freqs = torch.exp(-math.log(10000) *
                      torch.arange(half_dim, dtype=torch.float32) /
                      half_dim).to(device=timesteps.device)
    args = timesteps[:, None].float() * freqs[None, :]
    return torch.cat([torch.cos(args), torch.sin(args)], dim=-1)
```

Training SD from scratch

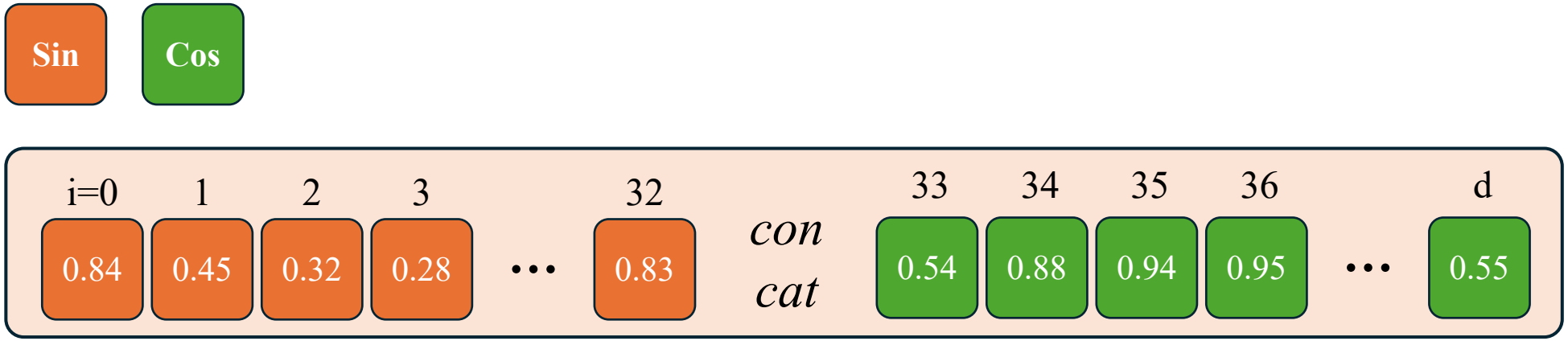
❖ Timestep Handling




```
 9 def embed_timesteps(timesteps, embedding_dim=320):
10     half_dim = embedding_dim // 2
11     freqs = torch.exp(-math.log(10000) *
12                       torch.arange(half_dim, dtype=torch.float32) /
13                       half_dim).to(device=timesteps.device)
14     args = timesteps[:, None].float() * freqs[None, :]
15     return torch.cat([torch.cos(args), torch.sin(args)], dim=-1)
```

Training SD from scratch

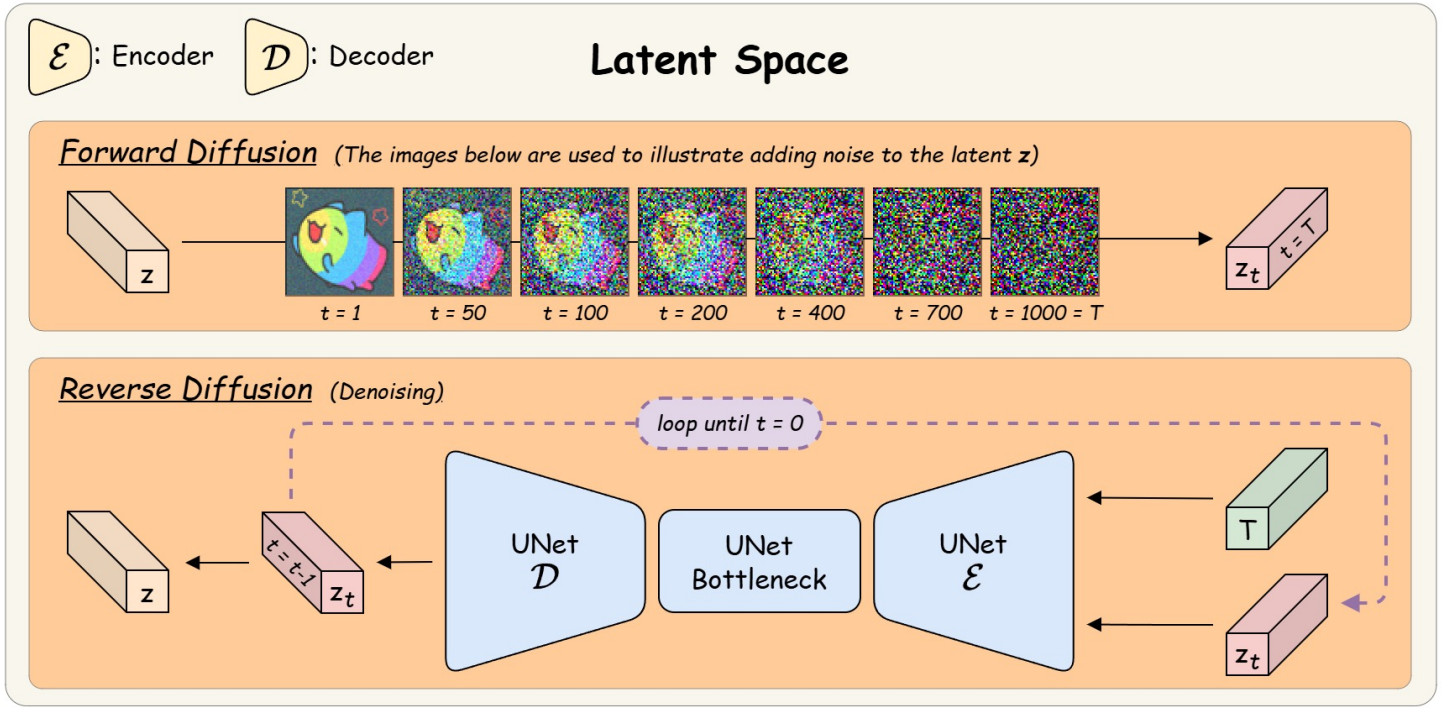
❖ Timestep Handling




```python
def embed_timesteps(timesteps, embedding_dim=320):
    half_dim = embedding_dim // 2
    freqs = torch.exp(-math.log(10000) *
                      torch.arange(half_dim, dtype=torch.float32) /
                      half_dim).to(device=timesteps.device)
    args = timesteps[:, None].float() * freqs[None, :]
    return torch.cat([torch.cos(args), torch.sin(args)], dim=-1)
```

Training SD from scratch

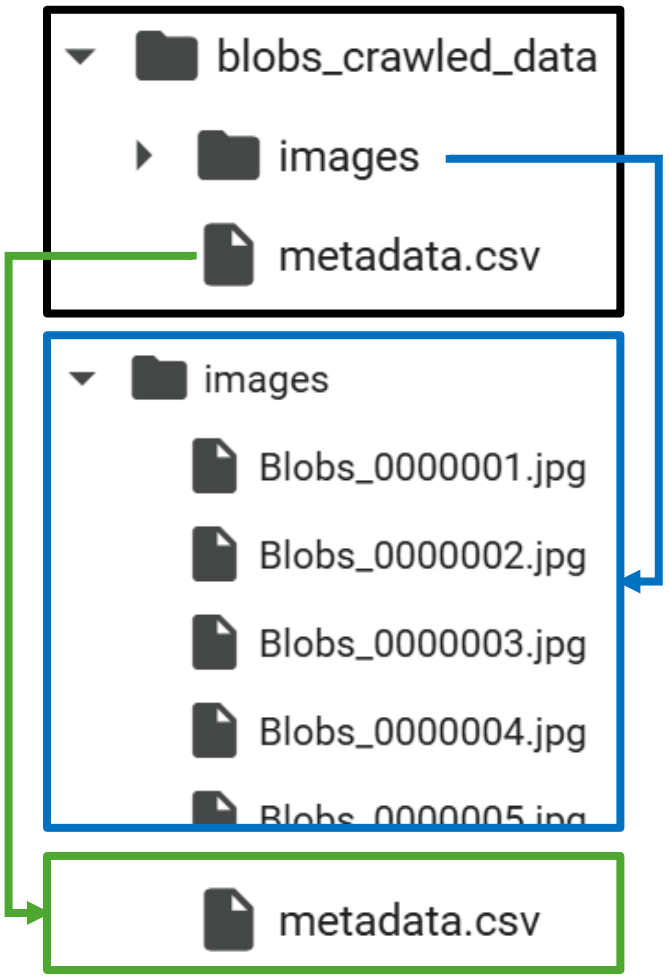
❖ Diffusion Process




```python
class Diffusion(nn.Module):
    def __init__(self, h_dim=128, n_head=4):
        super().__init__()
        self.time_embedding = TimeEmbedding(320)
        self.unet = UNET(h_dim, n_head)
        self.unet_output = UNETOutputLayer(h_dim, 4)

    @torch.autocast(
        device_type='cuda', dtype=torch.float16,
        enabled=True, cache_enabled=True
    )
    def forward(self, latent, context, time):
        time = self.time_embedding(time)
        output = self.unet(latent, context, time)
        output = self.unet_output(output)
        return output
```

Training SD from scratch

❖ Dataset Preparation




```python
transform = transforms.Compose([
    transforms.Resize(
        (WIDTH, HEIGHT),
        interpolation=transforms.InterpolationMode.BICUBIC
    ),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])

csv_files = ['/content/blobs_crawled_data/metadata.csv']
image_folder = '/content/blobs_crawled_data/images'

train_dataset = EmojiDataset(
    csv_files=csv_files,
    image_folder=image_folder,
    transform=transform
)
train_dataloader = DataLoader(
    train_dataset,
    batch_size=batch_size,
    shuffle=True,
    num_workers=2,
    pin_memory=True,
    persistent_workers=True
)
```

AI VIET NAM
@aivietnam.edu.vn

Training SD from scratch

file_name	image_url	image_title	tag	prompt
4621 unique values	4621 unique values	3344 categories	1 category	4087 categories
Blobs_0000001.jpg	https://discords.com/_next/image?url=https%3A%2F%2Fcdn.discordap	bloboof	Blobs	bloboof emoji with a square head, small eyes, open mouth, surprised ex
Blobs_0000002.jpg	https://discords.com/_next/image?url=https%3A%2F%2Fcdn.discordap	blobhyperthink	Blobs	blobhyperthink emoji with a round head, large circular eyes, wide open r
Blobs_0000003.jpg	https://discords.com/_next/image?url=https%3A%2F%2Fcdn.discordap	blob_minecraft_totem_of_undying	Blobs	blob_minecraft_totem_of_undying emoji with a blob-like head, large eye
Blobs_0000004.jpg	https://discords.com/_next/image?url=https%3A%2F%2Fcdn.discordap	BlobCuddle	Blobs	BlobCuddle emoji with a rounded head, closed eyes, small mouth, conte
Blobs_0000005.jpg	https://discords.com/_next/image?url=https%3A%2F%2Fcdn.discordap	nyan_party_blob	Blobs	Nyan Party Blob emoji with a rounded head, small black eyes, squiggly r

```python
WIDTH, HEIGHT = 64, 64
batch_size = 512

class EmojiDataset(Dataset):
    def __init__(self, csv_files, image_folder, transform=None):
        self.dataframe = pd.concat([pd.read_csv(csv_file) for csv_file in csv_files])
        self.images_folder = image_folder
        self.dataframe["image_path"] = self.dataframe["file_name"].str.replace("\\", "/")
        self.image_paths = self.dataframe["image_path"].tolist()
        self.titles = self.dataframe["prompt"].tolist()
        self.transform = transform
```

```python
    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, idx):
        image_path = self.images_folder + "/" + self.image_paths[idx]
        title = self.titles[idx]
        title = title.replace('"', "").replace("'", "")
        image = Image.open(image_path).convert('RGB')

        if self.transform:
            image = self.transform(image)

        return image, title
```

❖ Dataset Preparation

Training SD from scratch

❖ Model Training

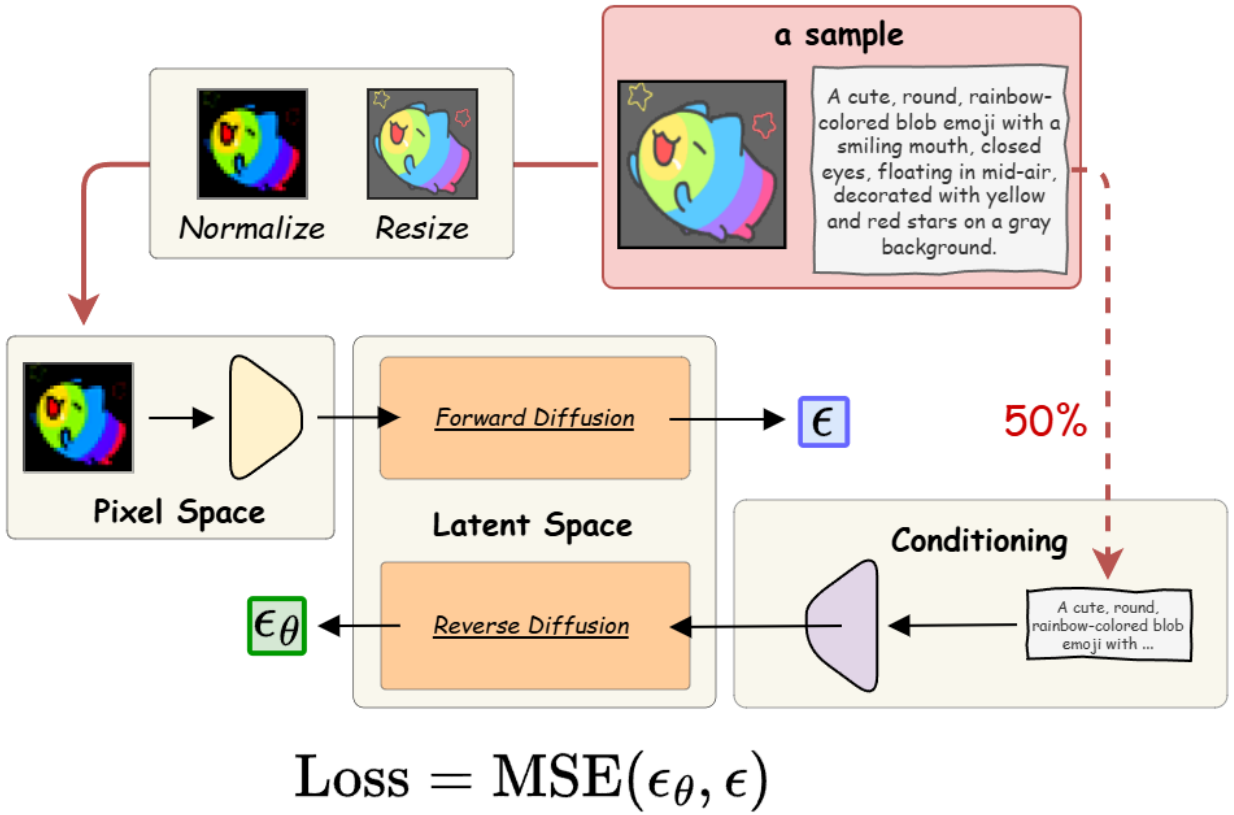
```python
def train(diffusion, vae, text_encoder, scheduler,
          optimizer, lr_scheduler, scaler,
          criterion, dataloader, num_epochs, device="cuda"):
    losses = []
    for epoch in range(num_epochs):
        diffusion.train()
        epoch_loss = 0.0
        progress_bar = tqdm(dataloader, desc=f"Epoch {epoch+1}/{num_epochs}", leave=False)
        for batch_idx, (images, titles) in enumerate(progress_bar):
            images = images.to(device)
            image_titles = [f"A photo of {title}" for title in titles]
            image_titles = [title if random.random() < 0.5 else "" for title in image_titles]

            with torch.no_grad():
                latents = vae.encode(images).latent_dist.sample() * 0.18215
            timesteps = torch.randint(
                0, scheduler.total_train_timesteps,
                (latents.shape[0],), device=device
            )
            noisy_latents, noise = scheduler.add_noise(latents, timesteps)
            time_embeddings = embed_timesteps(timesteps).to(device)
            text_embeddings = text_encoder(image_titles)
            noise_pred = diffusion(noisy_latents, text_embeddings, time_embeddings)

            with autocast(device_type='cuda', dtype=torch.float16,
                          enabled=True, cache_enabled=True):
                loss = criterion(noise_pred, noise)
```

SKIP

```python
            optimizer.zero_grad()
            scaler.scale(loss).backward()
            scaler.step(optimizer)
            scaler.update()

            batch_loss = loss.item()
            epoch_loss += batch_loss

            progress_bar.set_postfix(loss=f"{batch_loss:.5f}",
                                     lr=f"{optimizer.param_groups[0]['lr']:.6f}")

        lr_scheduler.step()
        avg_epoch_loss = epoch_loss / len(dataloader)
        if (epoch + 1) % 10 == 0 or epoch == 0:
            print(f"Epoch [{epoch+1}/{num_epochs}] - Avg Loss: {avg_epoch_loss:.5f}")
        losses.append(avg_epoch_loss)

    return losses
```

Training SD from scratch

❖ Model Training



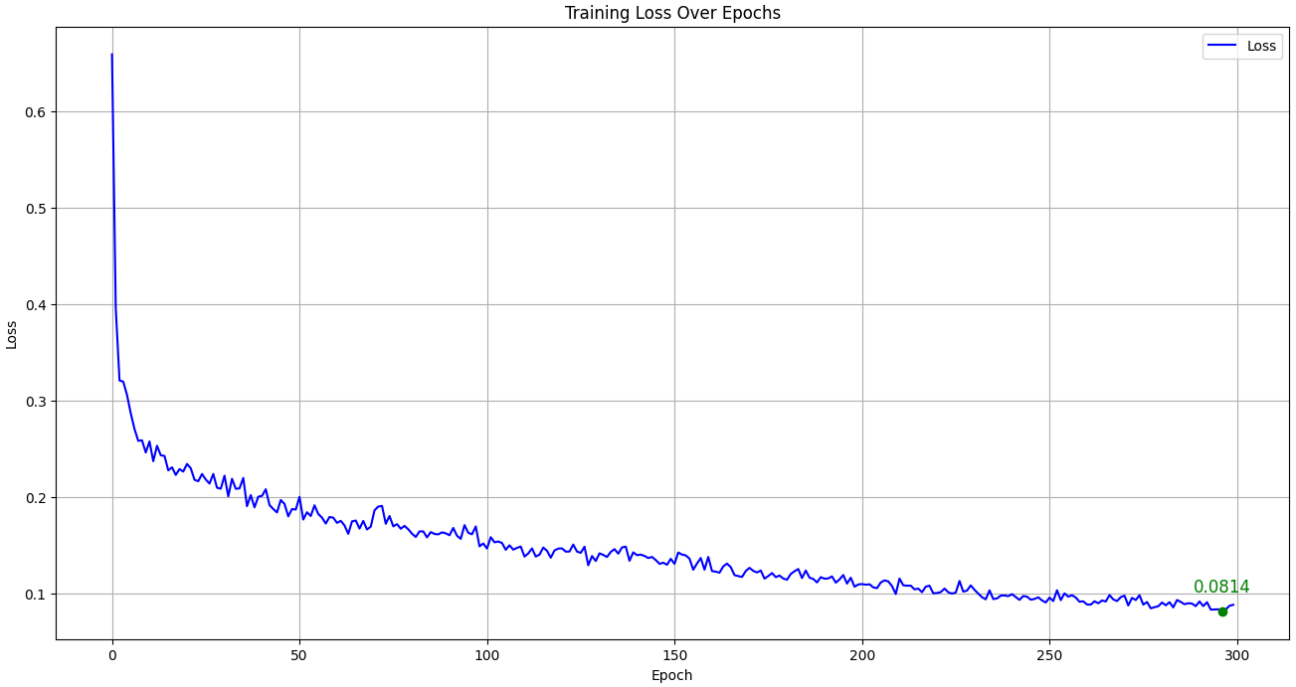
$$\text{Loss} = \text{MSE}(\epsilon_\theta, \epsilon)$$

```python
def train(diffusion, vae, text_encoder, scheduler,
          optimizer, lr_scheduler, scaler,
          criterion, dataloader, num_epochs, device="cuda"):
    losses = []
    for epoch in range(num_epochs):
        diffusion.train()
        epoch_loss = 0.0
        progress_bar = tqdm(dataloader, desc=f"Epoch {epoch+1}/{num_epochs}", leave=False)
        for batch_idx, (images, titles) in enumerate(progress_bar):
            images = images.to(device)
            image_titles = [f"A photo of {title}" for title in titles]
            image_titles = [title if random.random() < 0.5 else "" for title in image_titles]

            with torch.no_grad():
                latents = vae.encode(images).latent_dist.sample() * 0.18215
            timesteps = torch.randint(
                0, scheduler.total_train_timesteps,
                (latents.shape[0],), device=device
            )
            noisy_latents, noise = scheduler.add_noise(latents, timesteps)
            time_embeddings = embed_timesteps(timesteps).to(device)
            text_embeddings = text_encoder(image_titles)
            noise_pred = diffusion(noisy_latents, text_embeddings, time_embeddings)

            with autocast(device_type='cuda', dtype=torch.float16,
                          enabled=True, cache_enabled=True):
                loss = criterion(noise_pred, noise)
```

The rationale for the normalization with the value 0.18215 is discussed in https://arxiv.org/pdf/2112.10752 in Section 4.3.2, Appendix D.1 and G.

Training SD from scratch

❖ Model Training



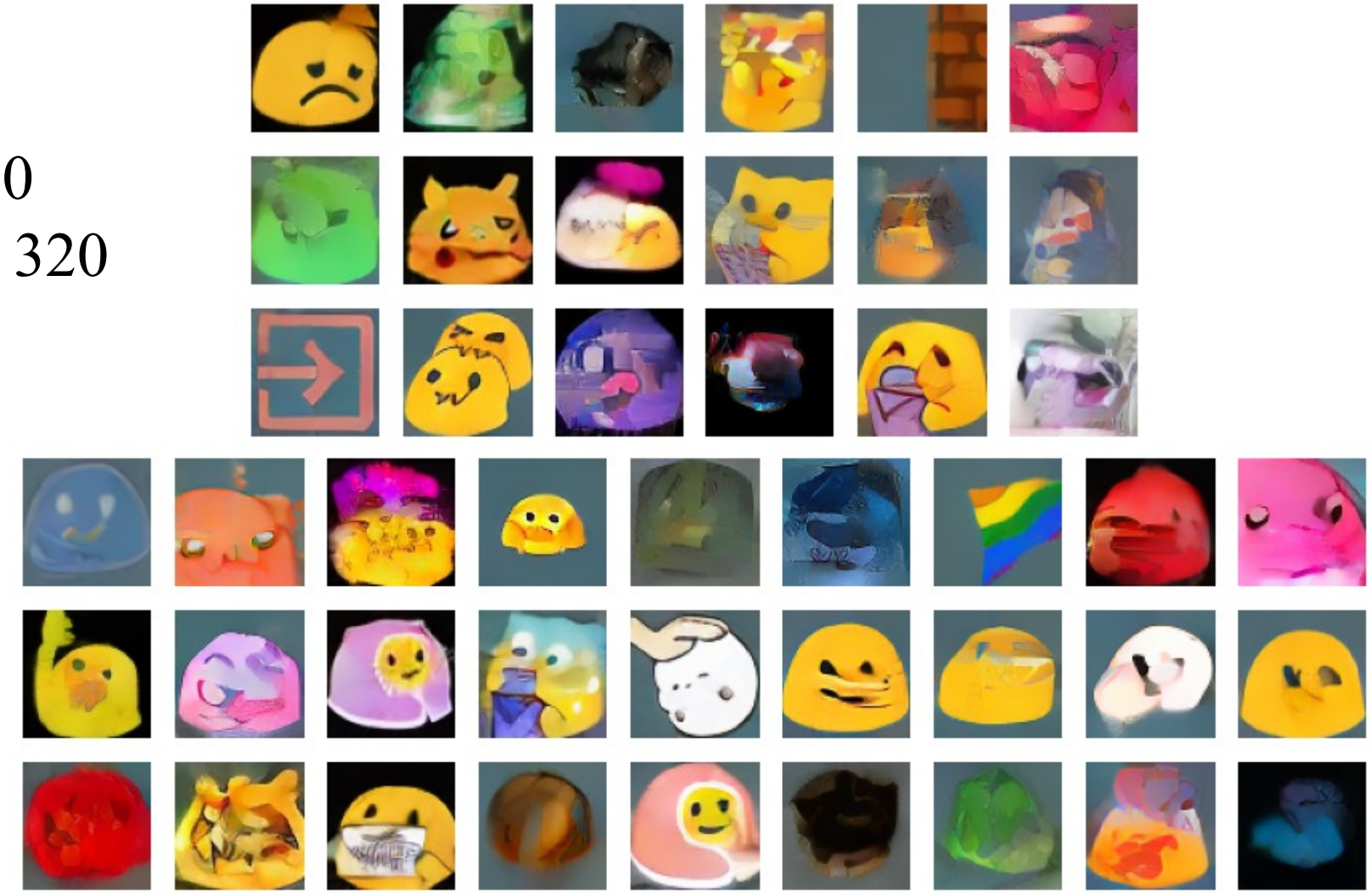

```
VAE parameters: 83653863
Diffusion parameters: 72993540
CLIP parameters: 63165952
Total parameters: 219813355
```

```python
EPOCHS = 300  # takes more than 3 hours to complete the training

h_dim = 256
n_head = 8

vae = vae.to(device)
diffusion = Diffusion(h_dim, n_head).to(device)
clip = CLIPTextEncoder().to(device)

random_generator = torch.Generator(device='cuda')
noise_scheduler = DDPMScheduler(random_generator)

optimizer = torch.optim.AdamW(diffusion.parameters(), lr=1e-4)
criterion = torch.nn.MSELoss()

lrate_scheduler = lr_scheduler.CosineAnnealingLR(
    optimizer, T_max=EPOCHS, eta_min=1e-5
)
scaler = GradScaler()

def count_parameters(model):
    return sum(p.numel() for p in model.parameters())

vae_params = count_parameters(vae)
diffusion_params = count_parameters(diffusion)
clip_params = count_parameters(clip)

print(f"VAE parameters: {vae_params}")
print(f"Diffusion parameters: {diffusion_params}")
print(f"CLIP parameters: {clip_params}")
print(f"Total parameters: {vae_params + diffusion_params + clip_params}")
```

Training SD from scratch

❖ Image Generation

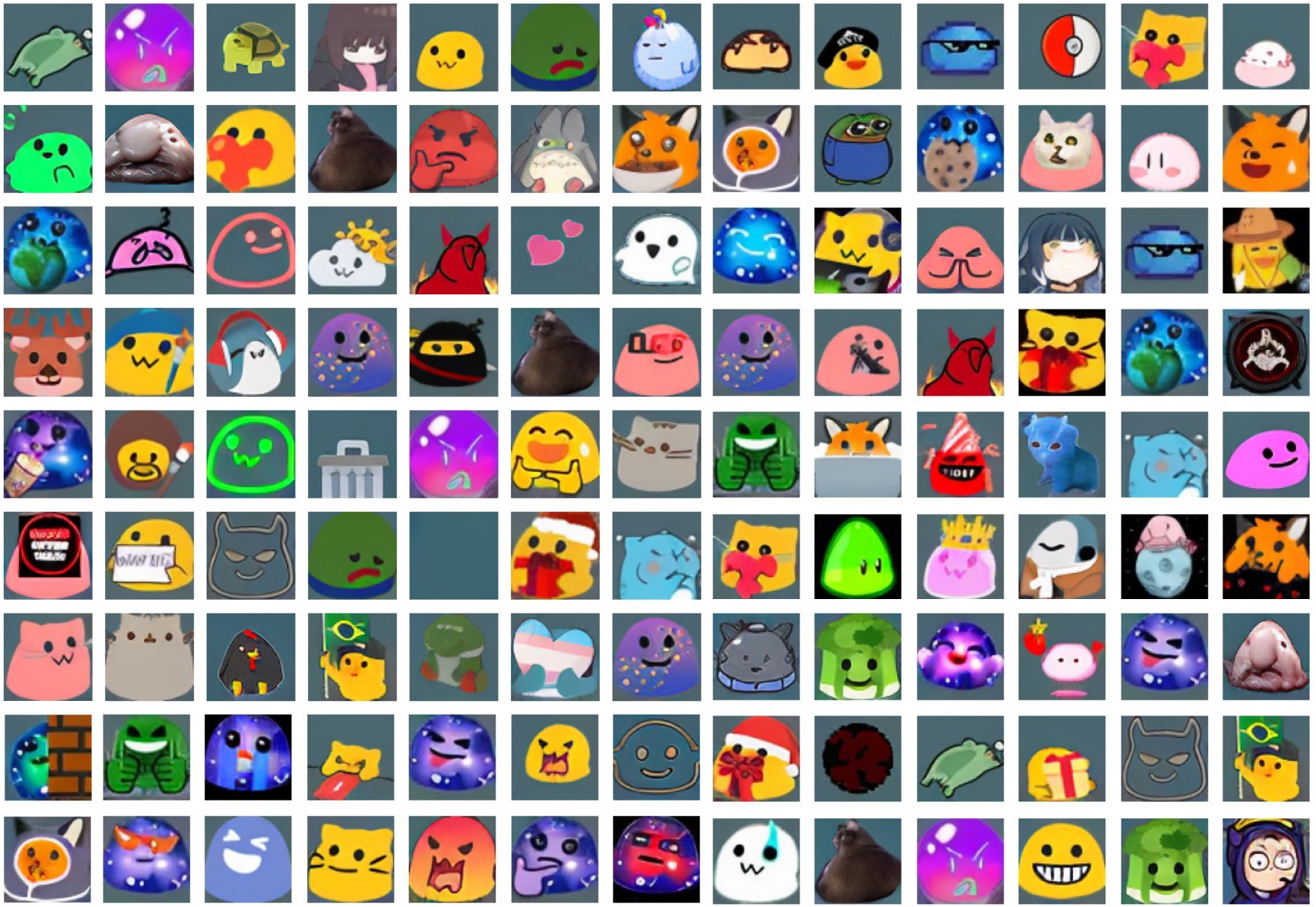
Epochs: 300
Batch size: 320



Training SD from scratch

❖ Image Generation

Epochs: 5000
Batch size: 640



QUIZ

Training SD from pre-trained models

Training SD from pre-trained models

❖ Introduction of Stable Diffusion

High-Resolution Image Synthesis with Latent Diffusion Models

Robin Rombach[1] * Andreas Blattmann[1] * Dominik Lorenz[1] Patrick Esser[R] Björn Ommer[1]
[1]Ludwig Maximilian University of Munich & IWR, Heidelberg University, Germany [R]Runway ML
https://github.com/CompVis/latent-diffusion

Abstract

By decomposing the image formation process into a sequential application of denoising autoencoders, diffusion models (DMs) achieve state-of-the-art synthesis results on image data and beyond. Additionally, their formulation allows for a guiding mechanism to control the image generation process without retraining. However, since these models typically operate directly in pixel space, optimization of powerful DMs often consumes hundreds of GPU days and inference is expensive due to sequential evaluations. To enable DM training on limited computational resources while retaining their quality and flexibility, we apply them in the latent space of powerful pretrained autoencoders. In contrast to previous work, training diffusion models on such a representation allows for the first time to reach a near-optimal point between complexity reduction and detail preservation, greatly boosting visual fidelity. By introducing cross-attention layers into the model architecture, we turn diffusion models into powerful and flexi-

Figure 1. Boosting the upper bound on achievable quality with less agressive downsampling. Since diffusion models offer excellent inductive biases for spatial data, we do not need the heavy spatial downsampling of related generative models in latent space, but can still greatly reduce the dimensionality of the data via suitable autoencoding models, see Sec. 3. Images are from the DIV2K [1] validation set, evaluated at 512^2 px. We denote the spatial downsampling factor by f. Reconstruction FIDs [29] and PSNR are calculated on ImageNet-val. [12]; see also Tab. 8.

Stable Diffusion

Stable Diffusion was made possible thanks to a collaboration with Stability AI and Runway and builds upon our previous work:

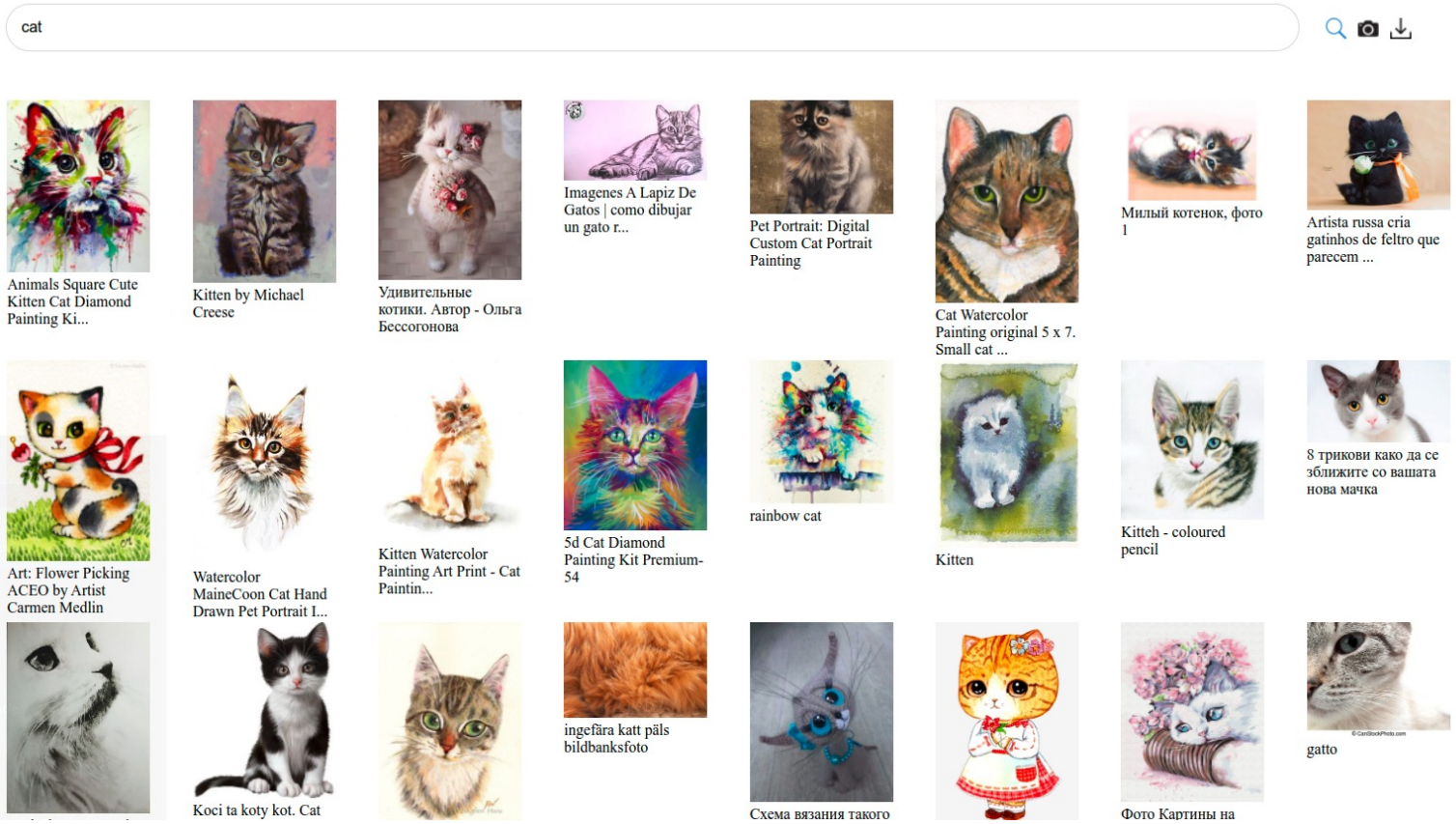
High-Resolution Image Synthesis with Latent Diffusion Models
Robin Rombach*, Andreas Blattmann*, Dominik Lorenz, Patrick Esser, Björn Ommer
CVPR '22 Oral | GitHub | arXiv | Project page

Stable Diffusion is a latent text-to-image diffusion model. Thanks to a generous compute donation from Stability AI and support from LAION, we were able to train a Latent Diffusion Model on 512x512 images from a subset of the LAION-5B database. Similar to Google's Imagen, this model uses a frozen CLIP ViT-L/14 text encoder to condition the model on text prompts. With its 860M UNet and 123M text encoder, the model is relatively lightweight and runs on a GPU with at least 10GB VRAM. See this section below and the model card.

https://ommer-lab.com/research/latent-diffusion-models/
https://github.com/CompVis/stable-diffusion

Training SD from pre-trained models

❖ Introduction of Stable Diffusion



Stable Diffusion:

- **Text Conditioning**: Incorporates a CLIP-based encoder for stronger text-image alignment.
- **Scalability**: Leverages large-scale datasets and extensive training to achieve high-quality image synthesis with fewer computational demands compared to prior latent diffusion variants.

LAION-5: a large, open dataset containing billions of image-text pairs for multi-modal AI research.

Training SD from pre-trained models

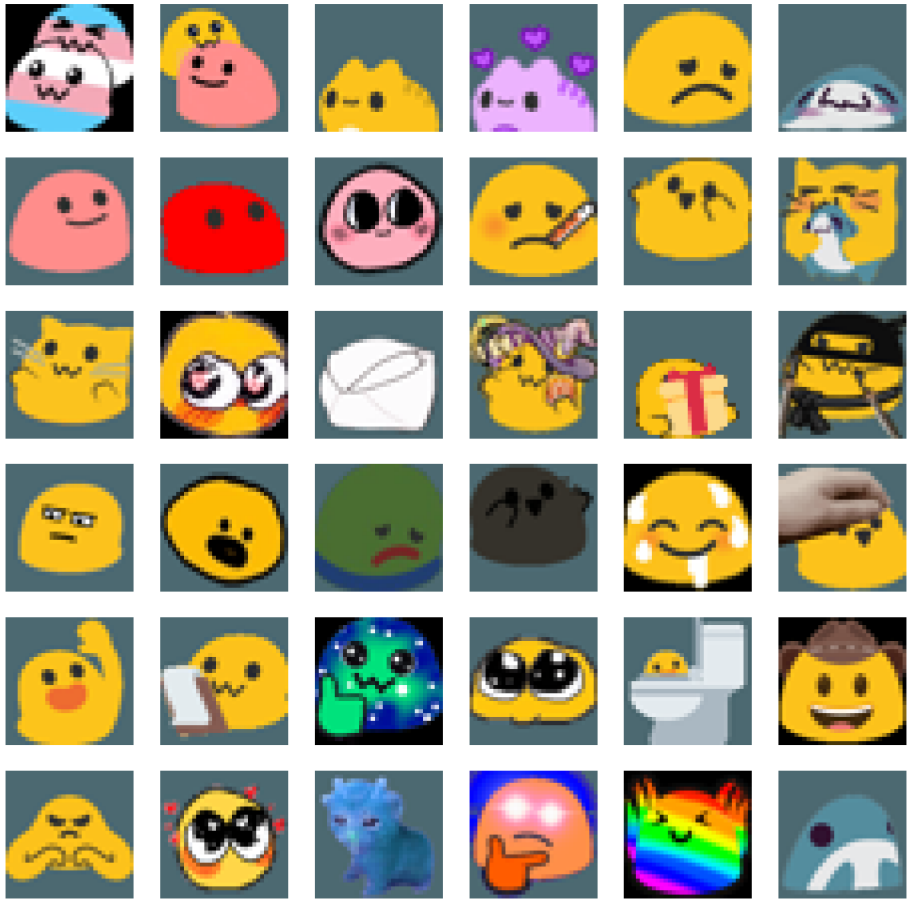
❖ Coding: Create PyTorch Datasets

```python
import os
import argparse
import math
import pandas as pd
import matplotlib.pyplot as plt
import torch
import torch.nn as nn

from torch.utils.data import Dataset, DataLoader
from torch.optim.lr_scheduler import LambdaLR
from torchvision import transforms
from diffusers import StableDiffusionPipeline,
from diffusers import UNet2DConditionModel
from diffusers import DDPMScheduler
from diffusers import AutoencoderKL
from transformers import CLIPTextModel, CLIPTokenizer, CLIPFeatureExtractor
from huggingface_hub import login, HfApi, create_repo
from PIL import Image
from tqdm.auto import tqdm
```

Training SD from pre-trained models

❖ Coding: Create PyTorch Datasets

```python
class EmojiDataset(Dataset):
    def __init__(self, metadata_df, image_dir, transform=None):
        self.metadata = metadata_df
        self.image_dir = image_dir
        self.transform = transform
        self.background_color = (255, 255, 255)

    def __len__(self):
        return len(self.metadata)

    def __getitem__(self, idx):
        img_path = os.path.join(self.image_dir, self.metadata.iloc[idx]['file_name'])
        img = Image.open(img_path)
        img.load()
        if img.mode == 'RGBA':
            background = Image.new("RGB", img.size, self.background_color)
            background.paste(img, mask=img.split()[3])
            img = background
        elif 'transparency' in img.info:
            img = img.convert("RGBA")
            background = Image.new("RGB", img.size, self.background_color)
            background.paste(img, mask=img.split()[3])
            img = background
        else:
            img = img.convert("RGB")
        prompt = self.metadata.iloc[idx]['prompt']
        if self.transform:
            img = self.transform(img)
        return img, prompt
```



Training SD from pre-trained models

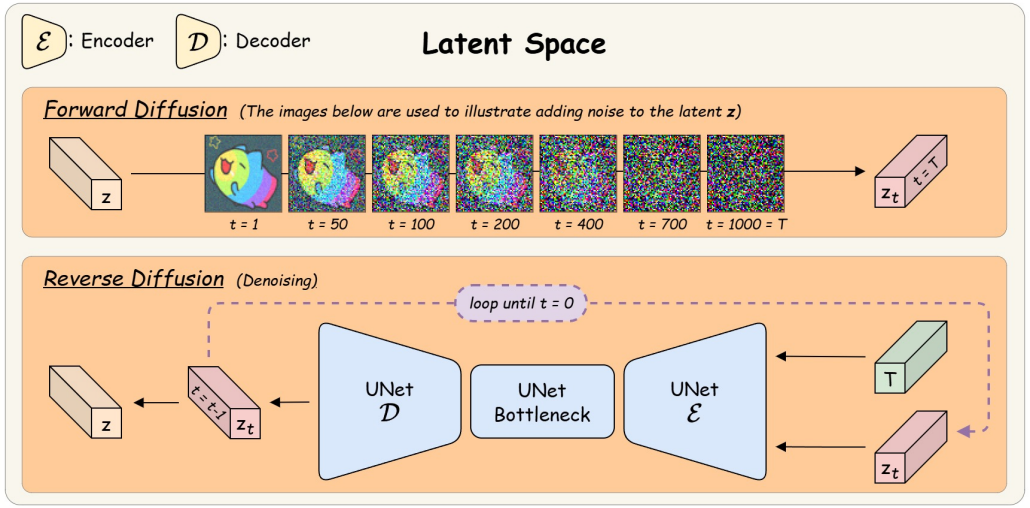
❖ Coding: Create PyTorch DataLoader

```python
def create_dataloaders(data_dir, image_size, batch_size, num_workers=2):
    metadata = pd.read_csv(os.path.join(data_dir, "metadata.csv"))
    torch.manual_seed(42)
    indices = torch.randperm(len(metadata)).tolist()
    train_size = int(0.9 * len(metadata))
    test_size = len(metadata) - train_size
    train_metadata = metadata.iloc[indices[:train_size]].reset_index(drop=True)
    test_metadata = metadata.iloc[indices[train_size:]].reset_index(drop=True)

    train_transform = transforms.Compose([
        transforms.Resize((image_size, image_size)),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.ColorJitter(brightness=0.2, contrast=0.2),
        transforms.ToTensor(),
        transforms.Normalize([0.5], [0.5])
    ])
    test_transform = transforms.Compose([
        transforms.Resize((image_size, image_size)),
        transforms.ToTensor(),
        transforms.Normalize([0.5], [0.5])
    ])
    image_dir = os.path.join(data_dir, "images")
    train_dataset = EmojiDataset(train_metadata, image_dir, train_transform)
    test_dataset = EmojiDataset(test_metadata, image_dir, test_transform)

    train_loader = DataLoader(
        train_dataset,
        batch_size=batch_size,
        shuffle=True,
        pin_memory=True,
        num_workers=num_workers
    )
    test_loader = DataLoader(
        test_dataset,
        batch_size=batch_size,
        shuffle=False,
        pin_memory=True,
        num_workers=num_workers
    )

    return train_loader, test_loader, test_dataset
```

Training SD from pre-trained models

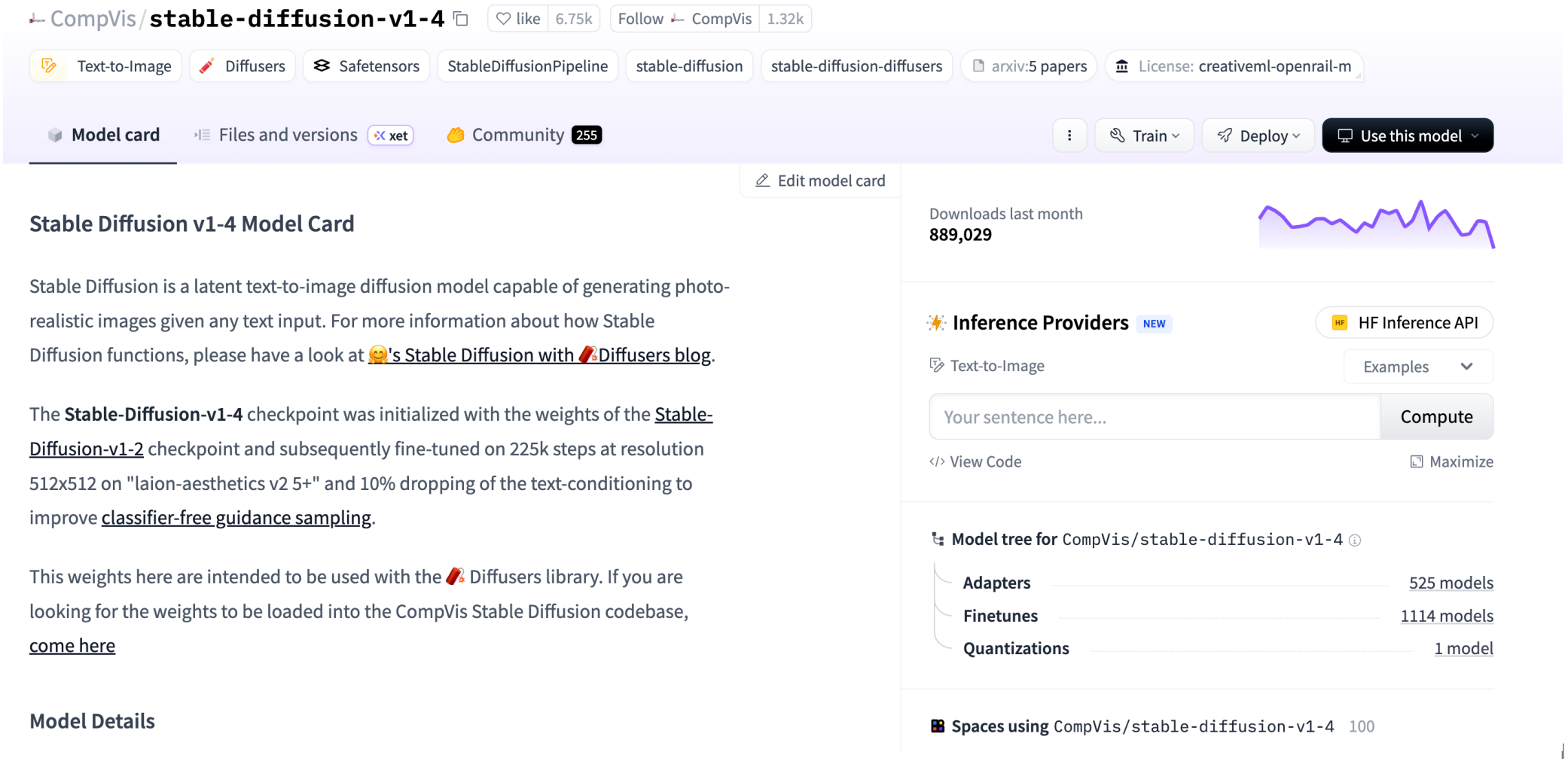
❖ Coding: Create model

```python
def freeze_unet_layers(unet):
    for name, param in unet.named_parameters():
        if "attn2" not in name:
            param.requires_grad = False

def build_sd_components(model_id):
    unet = UNet2DConditionModel.from_pretrained(model_id, subfolder="unet")
    text_encoder = CLIPTextModel.from_pretrained(model_id, subfolder="text_encoder")
    tokenizer = CLIPTokenizer.from_pretrained(model_id, subfolder="tokenizer")
    scheduler = DDPMScheduler.from_pretrained(model_id, subfolder="scheduler")
    vae = AutoencoderKL.from_pretrained(model_id, subfolder="vae")
    return unet, text_encoder, tokenizer, scheduler, vae

def generate_model_name(args, model_id):
    return (
        f"sd_bs{args.batch_size}_lr{args.learning_rate}_wd{args.weight_decay}"
        f"_img{args.image_size}_epochs{args.epochs}_{model_id}"
    )
```



Training SD from pre-trained models

❖ Coding: Create model



Training SD from pre-trained models

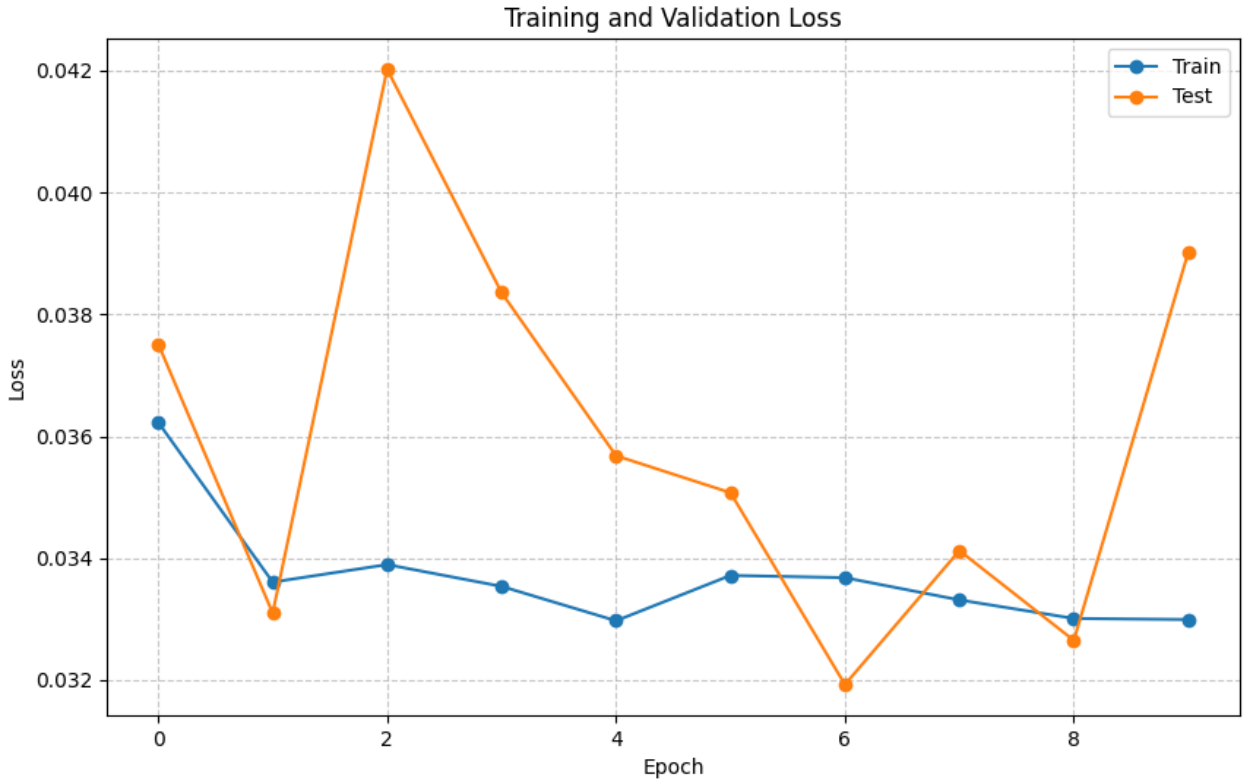
❖ Coding: Create model

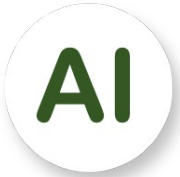
Model	Library	Details
stable-diffusion-v1-1	🤗 Diffusers	237k steps at resolution 256x256 on laion2B-en. 194k steps at resolution 512x512 on laion-high-resolution.
stable-diffusion-v1-2	🤗 Diffusers	v1-1 plus: 515k steps at 512x512 on "laion-improved-aesthetics".
stable-diffusion-v1-3	🤗 Diffusers	v1-2 plus: 195k steps at 512x512 on "laion-improved-aesthetics", with 10% dropping of text-conditioning.
stable-diffusion-v1-4	🤗 Diffusers	v1-2 plus: 225k steps at 512x512 on "laion-aesthetics v2 5+", with 10% dropping of text conditioning.
stable-diffusion-v-1-1-original	CompVis	237k steps at resolution 256x256 on laion2B-en. 194k steps at resolution 512x512 on laion-high-resolution.
stable-diffusion-v-1-2-original	CompVis	v1-1 plus: 515k steps at 512x512 on "laion-improved-aesthetics".
stable-diffusion-v-1-3-original	CompVis	v1-2 plus: 195k steps at 512x512 on "laion-improved-aesthetics", with 10% dropping of text-conditioning.
stable-diffusion-v-1-4-original	CompVis	v1-2 plus: 225k steps at 512x512 on "laion-aesthetics v2 5+", with 10% dropping of text conditioning.

https://huggingface.co/CompVis

Training SD from pre-trained models

❖ Results



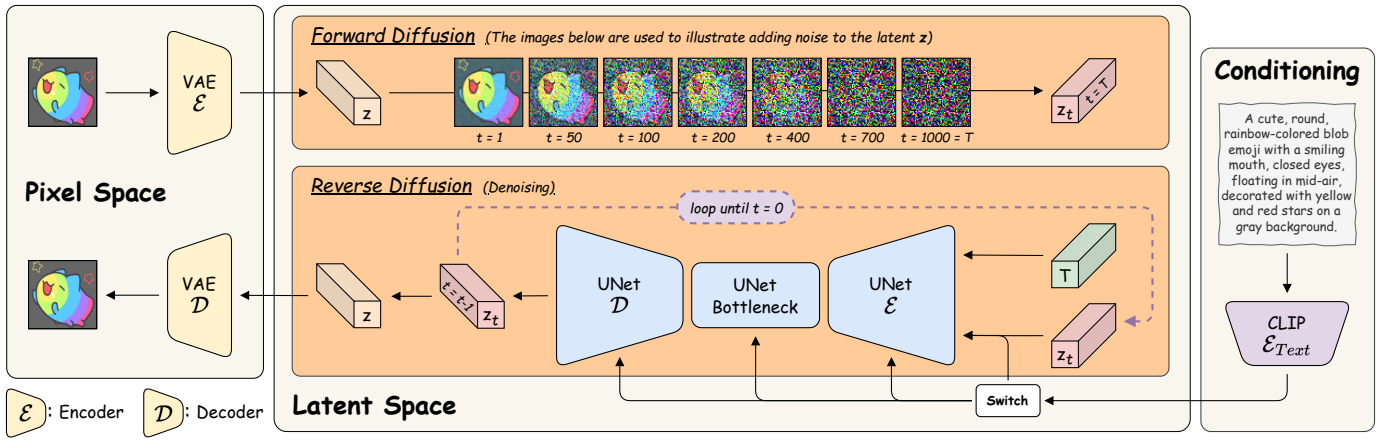


AI VIET NAM
@aivietnam.edu.vn

Summarization and Q&A

Summarization and Q&A

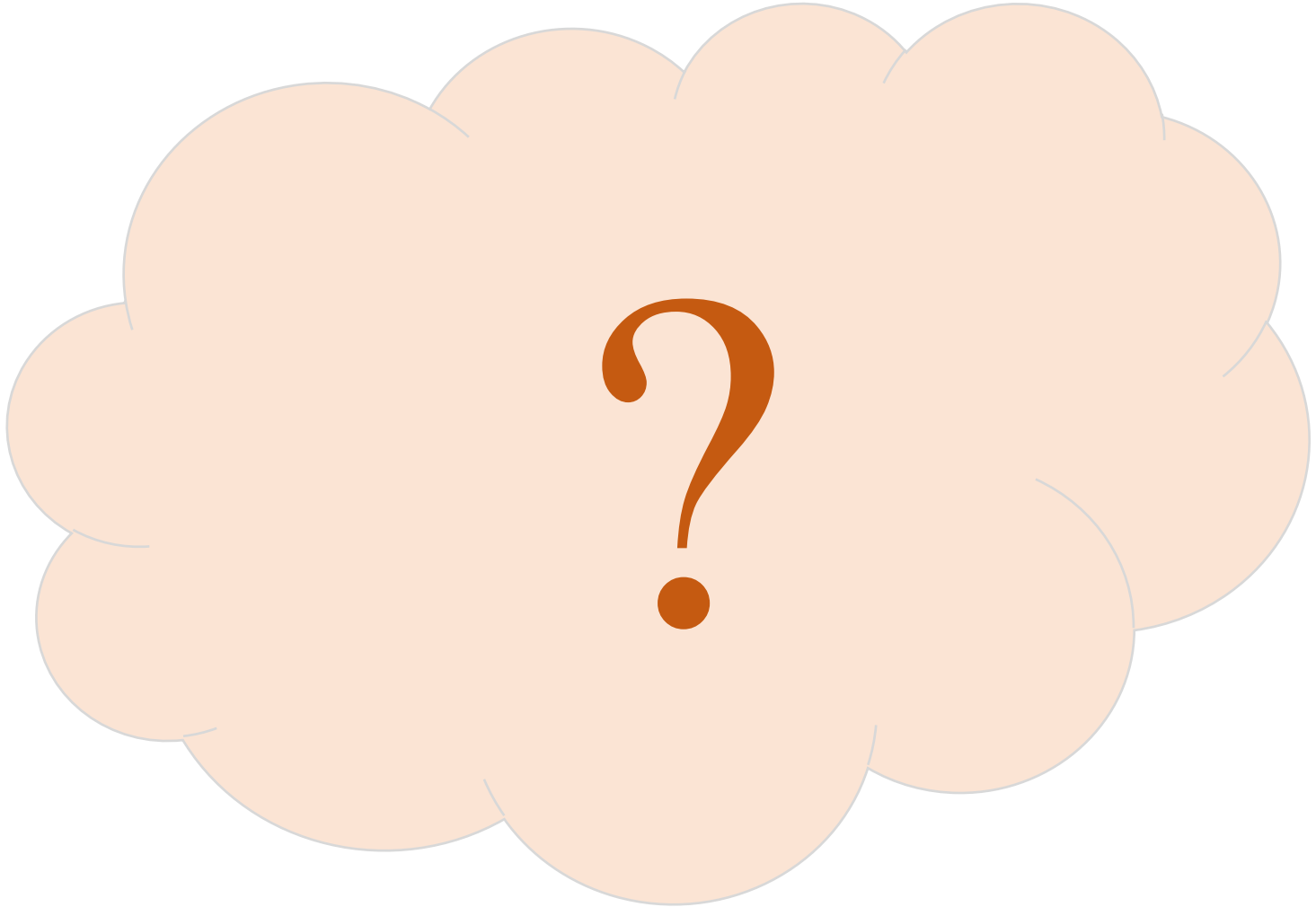
❖ Content



In this section, we have discussed about:

- The task of Image Generation (IG) and Stable Diffusion (SD).
- A Python program to crawl Discord Emojis on website.
- Implement and training:
  - A SD model from scratch using PyTorch and Diffusers.
  - A SD model from a pre-trained model using Diffusers.

Summarization and Q&A




Thank you!