

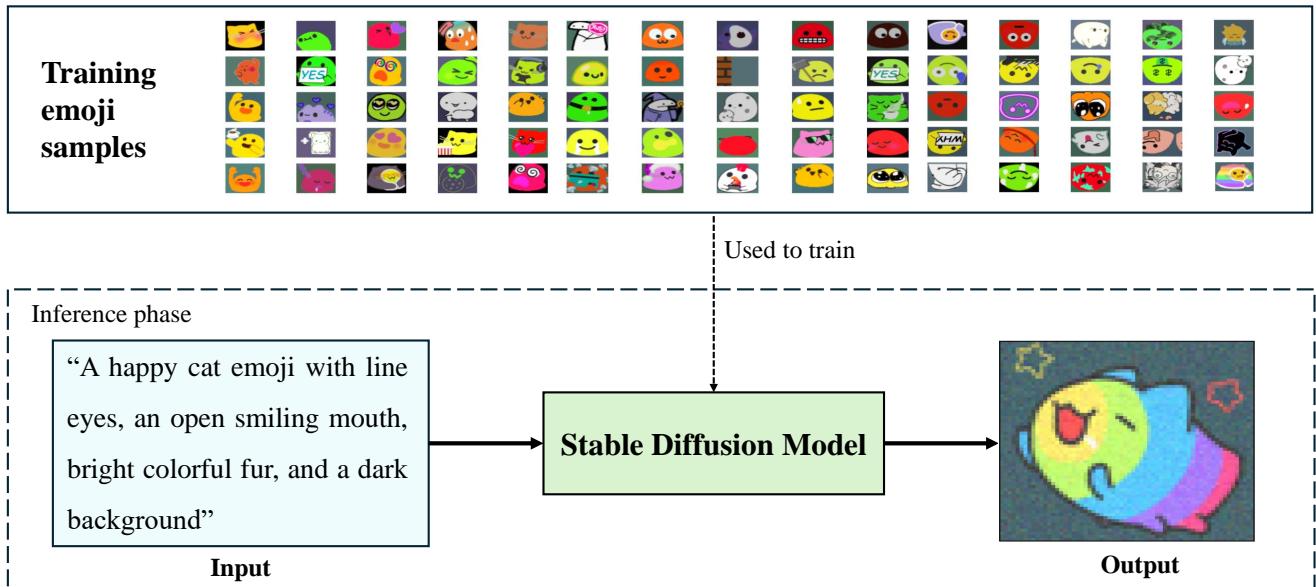
AI VIET NAM – AI COURSE 2024

# Project: Emoji Image Generation

Dinh-Thang Duong, Yen-Linh Vu, Anh-Khoi Nguyen, Quang-Vinh Dinh

## I. Giới thiệu

**Emoji Generation** (Tạm dịch: Tạo sinh ảnh biểu tượng cảm xúc) là một bài toán thuộc lĩnh vực Thị giác máy tính (Computer Vision) và Mô hình tạo sinh (Generative Models), tập trung vào việc xây dựng một hệ thống có khả năng tạo ra các emoji mới dựa trên mô tả văn bản hoặc các đặc trưng hình ảnh đầu vào. Trên thế giới, nhiều nghiên cứu và ứng dụng đã khai thác mô hình học sâu để tạo ra nội dung hình ảnh mới, chẳng hạn như [DALL-E](#) của OpenAI, [Imagen](#) của Google, hay [Stable Diffusion](#). Các mô hình này có thể tạo ra hình ảnh từ mô tả văn bản với độ chân thực và sáng tạo cao. Trong bài toán này, chúng ta sẽ áp dụng **Stable Diffusion Model**, một mô hình khuếch tán mạnh mẽ, để sinh ra các emoji theo yêu cầu. Mục tiêu của bài toán không chỉ dừng lại ở việc sao chép các emoji có sẵn, mà còn hướng đến khả năng sáng tạo, giúp tạo ra các emoji hoàn toàn mới nhưng vẫn giữ được phong cách nhất quán với một bộ emoji gốc.



Hình 1: Minh họa về bài toán sinh emoji bằng Stable Diffusion Model.

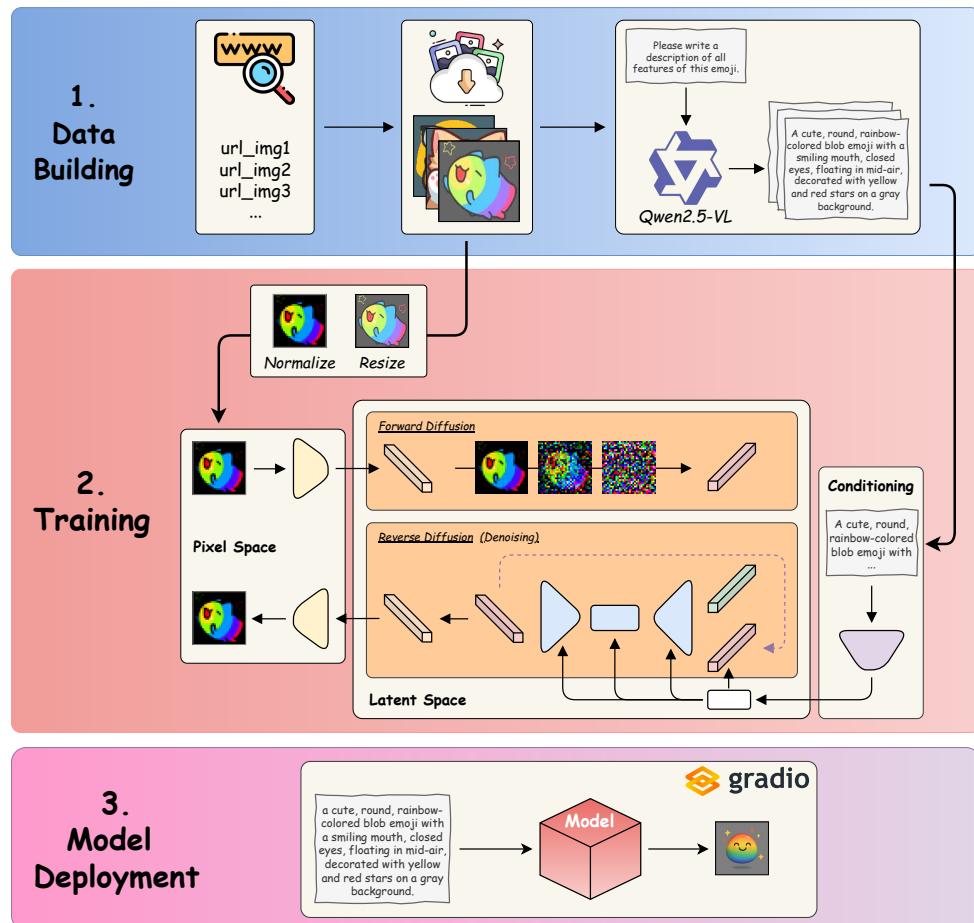
## I.1. Mô tả bài toán

Bài toán được mô tả như sau:

- **Input:** Một prompt mô tả emoji cần tạo (hình dáng, màu sắc, biểu cảm, nền). Ví dụ: "Một emoji chú mèo dễ thương, có màu cầu vồng, miệng cười, mắt nhắm, đang lơ lửng giữa không trung, nền tối màu".
- **Output:** Hình ảnh emoji mới được tạo theo prompt, mang phong cách từ tập dữ liệu emoji đã được huấn luyện.

## I.2. Project pipeline

Dựa trên các mô tả nội dung trên, ta có một pipeline tổng quát cho toàn bộ project được mô tả trong sơ đồ sau:



Hình 2: Pipeline tổng quát của bài toán sinh Emoji bằng Stable Diffusion Model.

Toàn bộ quy trình sinh ảnh emoji từ mô tả văn bản được tổ chức thành ba giai đoạn chính: **Data Building**, **Training**, và **Model Deployment**, như minh họa trong Hình 2.

## 1. Data Building (Xây dựng bộ dữ liệu)

Giai đoạn đầu tiên là xây dựng bộ dữ liệu để huấn luyện mô hình. Hệ thống tự động thu thập hình ảnh emoji từ Internet thông qua các đường dẫn URL. Sau khi tải về, ảnh được chuẩn hóa (normalize) và thay đổi kích thước (resize) về định dạng cố định. Đồng thời, mỗi ảnh được chú thích bằng văn bản mô tả chi tiết, chẳng hạn: "*A happy cat emoji with line eyes, an open smiling mouth, bright colorful fur, and a dark background*" - "Một emoji chú mèo dễ thương, có màu cầu vồng, miệng cười, mắt nhí nhố, đang lơ lửng giữa không trung, được trang trí bằng các ngôi sao màu vàng và đỏ trên nền xám.". Ta sử dụng mô hình ngôn ngữ đa phương thức **Qwen2.5-VL** để tự động hóa quá trình tạo nhãn mô tả sử dụng nhằm sinh ra caption từ ảnh, đóng vai trò là điều kiện đầu vào (conditioning prompt) trong quá trình huấn luyện.

## 2. Training (Huấn luyện mô hình)

Mô hình huấn luyện chính là **Stable Diffusion** — một mô hình khuếch tán (diffusion model) hiện đại trong lĩnh vực sinh ảnh. Ảnh sau khi chuẩn hóa được mã hóa vào không gian điểm ảnh (pixel space), sau đó đưa vào pipeline khuếch tán. Trong quá trình khuếch tán (forward diffusion), ảnh gốc được thêm nhiễu qua nhiều bước để chuyển vào không gian tiềm ẩn (latent space). Tiếp đó, giai đoạn khuếch tán ngược (reverse diffusion) học cách tái tạo lại ảnh từ nhiễu, với sự hỗ trợ từ thông tin điều kiện (prompt) đã sinh.

Trong mã nguồn, quá trình huấn luyện được xây dựng dựa trên thư viện **diffusers** của Hugging Face. Thư viện này cung cấp sẵn các thành phần mô hình cần thiết để thực hiện quy trình khuếch tán, được tổ chức thành một pipeline gọi là **StableDiffusionPipeline**.

**StableDiffusionPipeline** bao gồm các thành phần chính như bộ mã hóa văn bản (text encoder, ví dụ: CLIP hoặc T5), bộ tạo nhiễu (noise scheduler, chẳng hạn: DDIMScheduler), và mô hình U-Net để dự đoán nhiễu trong quá trình huấn luyện. Quá trình huấn luyện bao gồm các bước: lấy mẫu nhiễu, tính toán hàm mất mát và cập nhật trọng số của mô hình qua nhiều vòng lặp.

Trong quá trình huấn luyện, mô hình sử dụng hàm mất mát dạng **MSE** giữa nhiễu thực tế và nhiễu dự đoán. Việc lựa chọn này xuất phát từ thực tế là mô hình sử dụng VAE đã được huấn luyện sẵn (pretrained) và cố định trọng số, do đó không tính thêm thành phần KL Divergence như trong công thức gốc của mô hình Latent Diffusion. Đồng thời, với độ phân giải ảnh thấp (32x32), hàm MSE đơn thuần vẫn cho kết quả học hiệu quả và giúp giảm chi phí tính toán. Phía dưới đây, chúng ta sẽ đi vào tìm hiểu công thức hàm mất mát đầy đủ của mô hình Stable Diffusion theo đúng lý thuyết gốc sau.

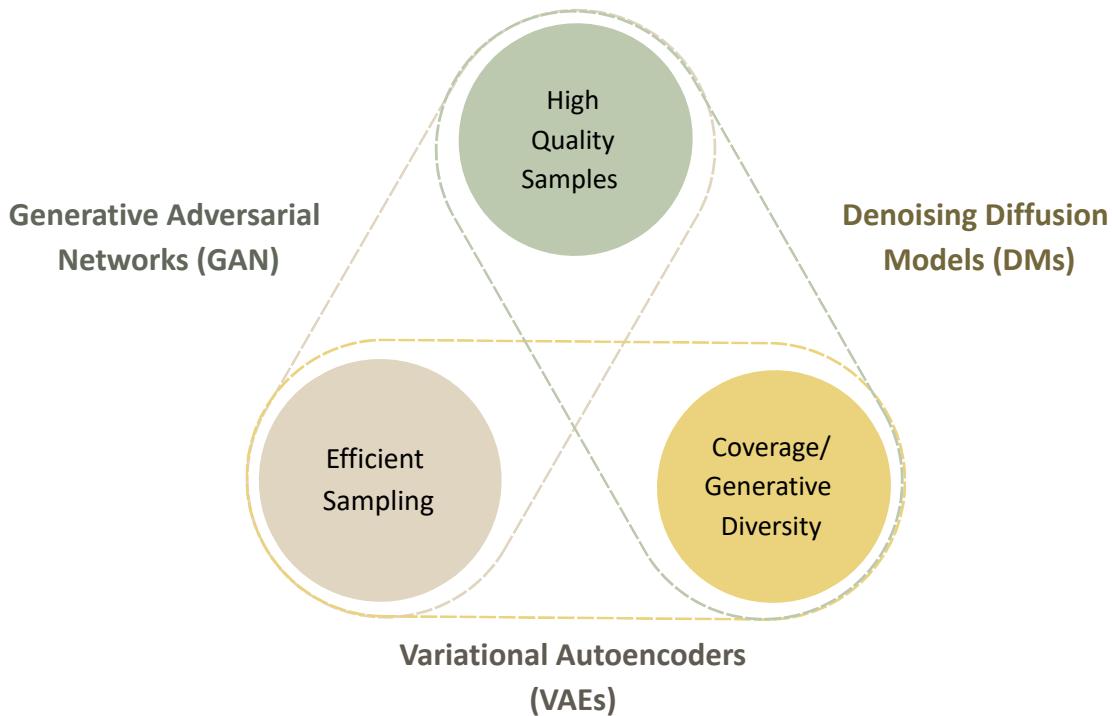
## 3. Model Deployment (Triển khai mô hình)

Sau khi huấn luyện xong, mô hình được chuẩn bị sẵn sàng để triển khai dưới dạng một ứng dụng đơn giản, có thể tương tác qua giao diện người dùng được xây dựng bằng thư viện **Gradio**. Người dùng chỉ cần nhập vào một mô tả emoji mong muốn, hệ thống sẽ tự động sinh ra hình ảnh emoji mới phù hợp với nội dung mô tả đó. Giao diện Gradio giúp quá trình sử dụng trở nên dễ dàng và trực quan, đồng thời hỗ trợ đánh giá chất lượng ảnh đầu ra một cách nhanh chóng.

## I.3. Tổng quan về Stable Diffusion Model

### I.3.1. Motivation

Trước khi đi vào chi tiết mô hình Stable Diffusion, hãy cùng nhìn lại lý do vì sao nó ra đời, dù trước đó đã có nhiều mô hình tạo sinh mạnh mẽ như GAN, VAE hay Diffusion Models.

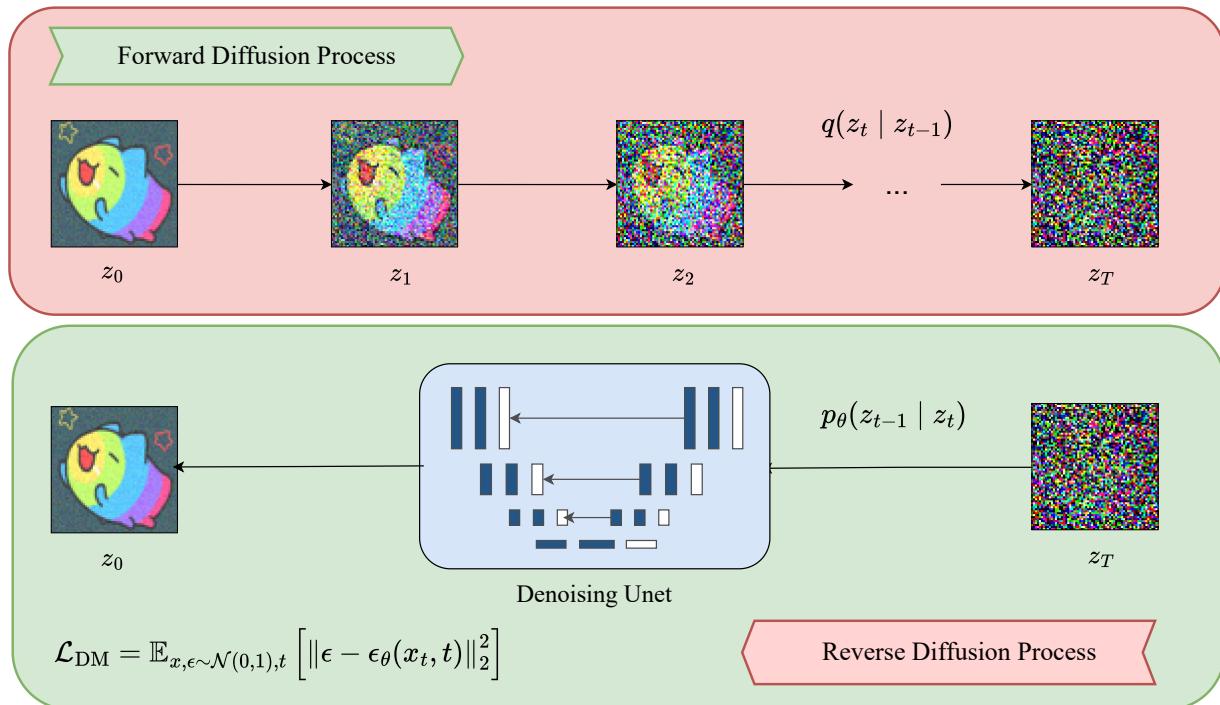


Hình 3: So sánh tổng quan các dòng mô hình sinh ảnh.

- **GANs** tạo ảnh sắc nét nhưng khó huấn luyện, dễ gặp lỗi mode collapse (mô hình chỉ sinh ra một số kiểu ảnh nhất định).
- **VAEs** ổn định hơn nhưng thường cho ảnh mờ, kém chân thực do giới hạn trong giả định phân phối tiềm ẩn.
- **Diffusion Models** nổi bật vì cho ảnh chất lượng cao, ổn định hơn GAN và mô hình hóa phân phối dữ liệu tốt hơn VAEs. Tuy nhiên, tốc độ sinh ảnh rất chậm.

Để hiểu nguyên nhân, ta cần nhắc lại cơ chế hoạt động của DMs (minh họa trong Hình 4), DMs tạo ảnh thông qua hai giai đoạn chính:

- **Forward Process (quá trình khuếch tán):** dần thêm nhiễu vào ảnh gốc  $z_0$  để thu được ảnh nhiễu hoàn toàn  $z_T$ .
- **Reverse Process (quá trình khử nhiễu):** sử dụng Denoising U-Net để tái tạo ảnh gốc từ  $z_T$  qua chuỗi bước  $z_T \rightarrow z_{T-1} \rightarrow \dots \rightarrow z_0$ .



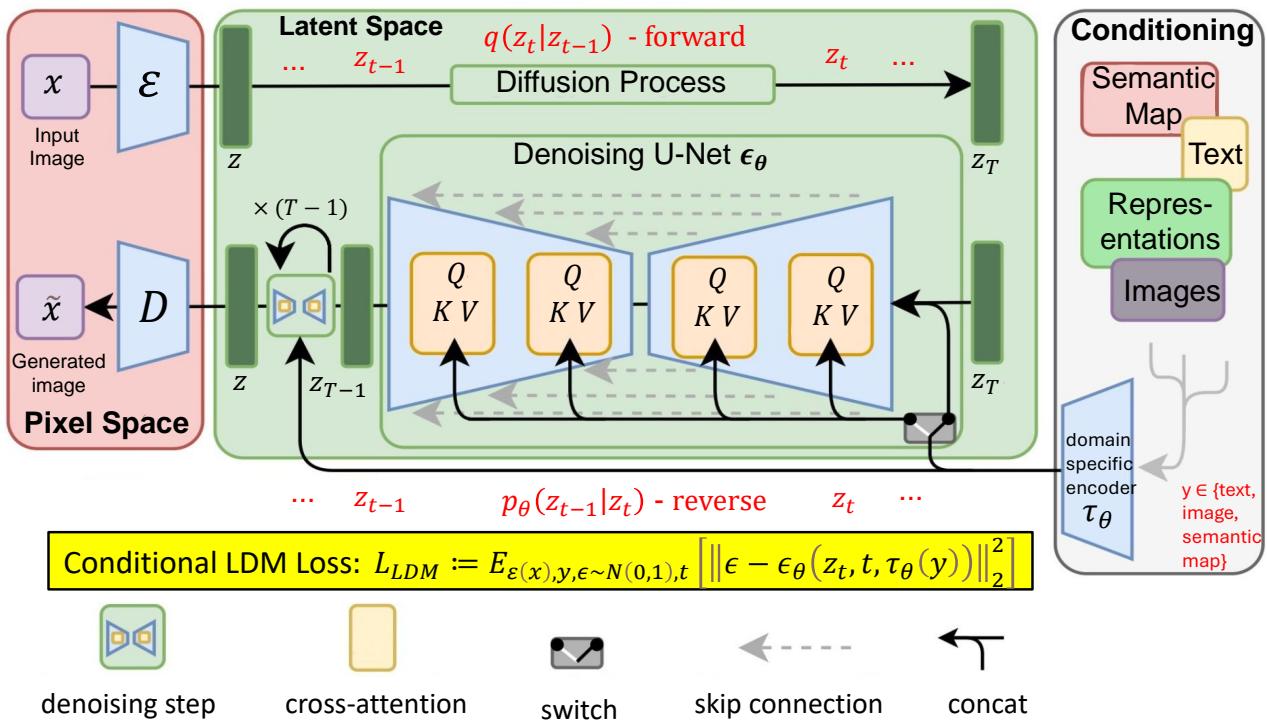
Hình 4: Cấu trúc tổng quát và luồng xử lý của Diffusion Model truyền thống.

Ta thấy được vấn đề nằm ở việc toàn bộ quá trình khuếch tán và khử nhiễu diễn ra trên ảnh kích thước lớn, nên mỗi bước sampling đều rất tốn tài nguyên. Hơn nữa, cần hàng trăm bước như vậy để khôi phục ảnh đầu ra. Đây là nguyên nhân chính khiến DMs trở nên chậm và khó ứng dụng thực tế, đặc biệt khi cần sinh ảnh theo thời gian thực.

Vì vậy, dù tạo ra ảnh có chất lượng tốt, DMs truyền thống vẫn gặp giới hạn nghiêm trọng về tốc độ xử lý và khả năng mở rộng. Chính hạn chế này đã thúc đẩy sự ra đời của **Stable Diffusion**.

### I.3.2. Stable Diffusion

Stable Diffusion thuộc nhóm **Latent Diffusion Models (LDMs)**, được giới thiệu bởi (Rombach et al., 2022). Ý tưởng chính của Stable Diffusion là, thay vì khuếch tán trực tiếp ảnh gốc  $x \in \mathbb{R}^{H \times W \times 3}$ , mô hình trước tiên sẽ mã hóa ảnh sang một vector đặc trưng  $z \in \mathbb{R}^d$  bằng encoder của VAE, sau đó thực hiện quá trình thêm nhiễu và khử nhiễu trong không gian này.



Hình 5: Kiến trúc của Stable Diffusion Model.

Kiến trúc trong Hình 5 cho ta thấy quá trình hoạt động của Stable Diffusion Model gồm ba bước chính:

1. **Encoding (mã hóa):** ảnh đầu vào  $x$  được encoder của VAE mã hóa thành vector tiềm ẩn  $z$ .
2. **Latent diffusion (khuếch tán trong không gian tiềm ẩn):** mô hình thêm nhiễu vào  $z$  để thu được  $z_t$ , sau đó dùng **U-Net** để dự đoán nhiễu  $\epsilon$  nhằm tái tạo lại vector gốc.
3. **Decoding (giải mã):** vector sạch  $z$  sau khi khử nhiễu được đưa qua decoder để tái tạo ảnh đầu ra.

Trong quá trình khử nhiễu, mô hình có thể nhận thêm **conditioning information (thông tin điều kiện)** như văn bản mô tả  $y$ , semantic map hoặc ảnh. Văn bản sẽ được mã hóa bởi một **text encoder** thành vector  $\tau_\theta(y)$ , rồi đưa vào U-Net thông qua **cross-attention** để ảnh sinh ra bám sát nội dung mô tả. Quá trình học được dán dắt bởi hàm mất mát có điều kiện:

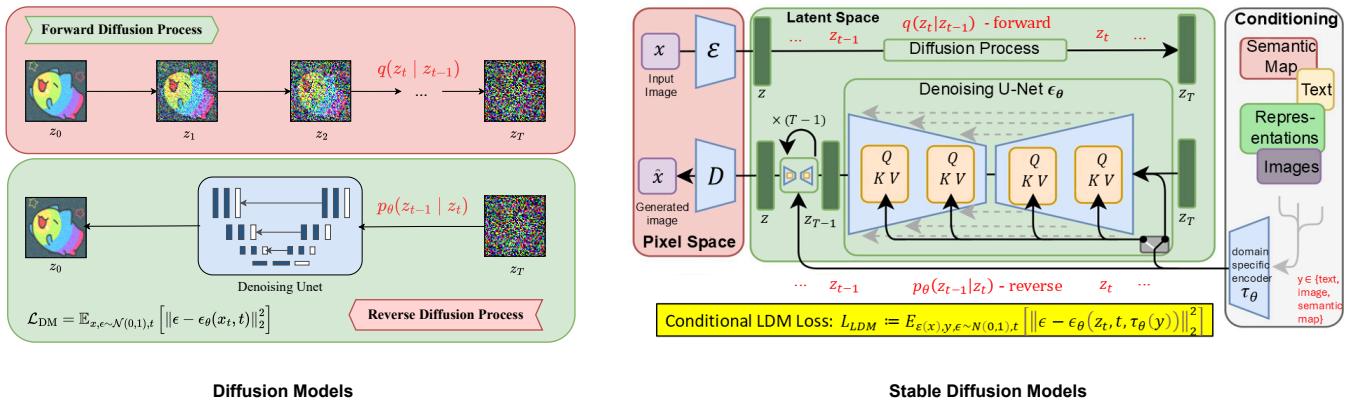
$$\mathcal{L}_{LDM} := \mathbb{E}_{\epsilon(x), y, \epsilon \sim \mathcal{N}(0, 1), t} [\|\epsilon - \epsilon_\theta(z_t, t, \tau_\theta(y))\|_2^2], \quad \text{trong đó:}$$

- $\epsilon \sim \mathcal{N}(0, 1)$ : nhiễu ngẫu nhiên được thêm vào.
- $z_t$ : vector latent sau khi bị thêm nhiễu tại bước  $t$ .
- $\epsilon_\theta$ : nhiễu được mô hình dự đoán.

Nhờ thực hiện toàn bộ quá trình khuếch tán trong **latent space**, kích thước đầu vào của U-Net giảm mạnh (thường 8–16 lần), từ đó tăng tốc huấn luyện và suy diễn (inference), giảm tiêu tốn tài nguyên mà vẫn giữ được chất lượng ảnh đầu ra ở mức cao.

### I.3.3. So sánh kiến trúc: Diffusion Models vs. Stable Diffusion Models

Điểm khác biệt cốt lõi giữa Diffusion Models (DMs) truyền thống và Stable Diffusion nằm ở không gian xử lý. Trong khi DMs thực hiện khuếch tán trực tiếp trên ảnh gốc có kích thước lớn  $x \in \mathbb{R}^{H \times W \times 3}$ , Stable Diffusion lựa chọn khuếch tán trong không gian tiềm ẩn (latent space)  $z \in \mathbb{R}^d$ , vốn đã được nén lại nhờ encoder của VAE. Điều này giúp giảm đáng kể chi phí tính toán, đồng thời tăng tốc độ suy diễn.



Hình 6: So sánh kiến trúc giữa Diffusion Model truyền thống và Stable Diffusion.

Bên cạnh đó, Stable Diffusion tích hợp thêm thông tin điều kiện (conditioning) như văn bản mô tả thông qua một text encoder huấn luyện sẵn (thường là CLIP). Vector hóa của prompt  $y$ , ký hiệu  $\tau_\theta(y)$ , được đưa vào mạng U-Net thông qua cơ chế cross-attention, cho phép mô hình kiểm soát nội dung ảnh sinh ra một cách hiệu quả.

## II. Cài đặt chương trình

Trong phần này, chúng ta sẽ tìm hiểu về quá trình xây dựng toàn bộ chương trình Emoji Image Generation sử dụng mô hình Stable Diffusion (SD). Trong đó, bao gồm hai phần nội dung lớn là thu thập bộ dữ liệu để huấn luyện mô hình và xây dựng mô hình Stable Diffusion.

### II.1. Thu thập bộ dữ liệu

Trong project này, chúng ta giả định chưa có sẵn bộ dữ liệu, vì vậy cần thực hiện thu thập dữ liệu. Dựa trên nội dung project, các file ảnh emoji sẽ được thu thập để huấn luyện mô hình SD. Nguồn dữ liệu từ internet là dễ tiếp cận nhất, và trong phạm vi project này, thư viện Selenium sẽ được sử dụng để thu thập dữ liệu từ trang web lưu trữ emoji, cụ thể là <https://discords.com/emoji-list>.



Hình 7: Trang web chứa các ảnh emoji có thể sử dụng trên ứng dụng Discord.

**Lưu ý:** Các bạn có thể tải trực tiếp bộ dữ liệu đã được thu thập sẵn ở phần IV. và bỏ qua phần này.

### II.1.1. Cài đặt thư viện Selenium

Chúng ta sẽ thực hiện thu thập dữ liệu trên Google Colab. Để sử dụng Selenium trên môi trường này, chúng ta cần thực thi đoạn code sau:

```

1 %%shell
2 # Ubuntu no longer distributes chromium-browser outside of snap
3 #
4 # Proposed solution: https://askubuntu.com/questions/1204571/how-to-install-
5 # chromium-without-snap
6
7 # Add debian buster
8 cat > /etc/apt/sources.list.d/debian.list << "EOF"
9 deb [arch=amd64 signed-by=/usr/share/keyrings/debian-buster.gpg] http://deb.
10 deb [arch=amd64 signed-by=/usr/share/keyrings/debian-buster-updates.gpg] http
11 deb [arch=amd64 signed-by=/usr/share/keyrings/debian-security-buster.gpg] http
12 EOF
13
14 # Add keys
15 apt-key adv --keyserver keyserver.ubuntu.com --recv-keys DCC9E9BF77E11517
16 apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 648ACFD622F3D138
17 apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 112695A0E562B32A
18 apt-key export 77E11517 | gpg --dearmour -o /usr/share/keyrings/debian-buster.
19 apt-key export 22F3D138 | gpg --dearmour -o /usr/share/keyrings/debian-buster-
20 apt-key export E562B32A | gpg --dearmour -o /usr/share/keyrings/debian-
21 security-buster.gpg
22
23 # Prefer debian repo for chromium* packages only
24 # Note the double-blank lines between entries
25 cat > /etc/apt/preferences.d/chromium.pref << "EOF"
26 Package: *
27 Pin: release a=eoan
28 Pin-Priority: 500
29
30 Package: *
31 Pin: origin "deb.debian.org"
32 Pin-Priority: 300
33
34 Package: chromium*
35 Pin: origin "deb.debian.org"
36 Pin-Priority: 700
37 EOF
38
39 # Install chromium and chromium-driver
40 apt-get update
41 apt-get install chromium chromium-driver

```

```

43
44 # Install selenium
45 pip install selenium

```

### II.1.2. Import các thư viện cần thiết

```

1 import os
2 import requests
3 import time
4 import pandas as pd
5 import random
6 import hashlib
7 import urllib.parse
8 from io import BytesIO
9 from PIL import Image
10
11 from tqdm import tqdm
12 from selenium import webdriver
13 from selenium.webdriver.chrome.service import Service
14 from selenium.webdriver.common.by import By
15 from selenium.webdriver.support.ui import WebDriverWait
16 from selenium.webdriver.support import expected_conditions as EC

```

### II.1.3. Khởi tạo driver trình duyệt web

Bước đầu tiên với Selenium, ta cần khởi tạo một trình duyệt web để có thể sử dụng trong các đoạn code sau:

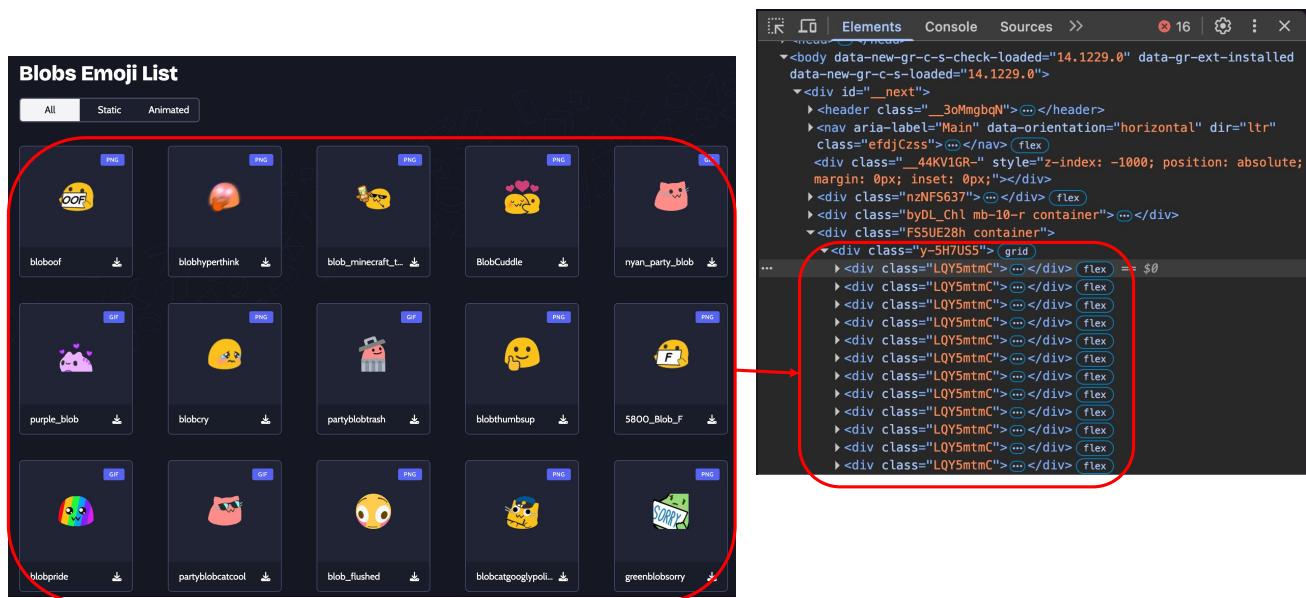
```

1 WEBDRIVER_DELAY_TIME_INT = 20
2 TIMEOUT_INT = 20
3 service = Service(executable_path=r"/usr/bin/chromedriver")
4 chrome_options = webdriver.ChromeOptions()
5 chrome_options.add_argument("--headless")
6 chrome_options.add_argument("--no-sandbox")
7 chrome_options.add_argument("--disable-dev-shm-usage")
8 chrome_options.add_argument("window-size=1920x1080")
9 chrome_options.headless = True
10 driver = webdriver.Chrome(service=service, options=chrome_options)
11 driver.implicitly_wait(TIMEOUT_INT)
12 wait = WebDriverWait(driver, WEBDRIVER_DELAY_TIME_INT)

```

### II.1.4. Xây dựng hàm trích xuất đường dẫn ảnh từ trang web

Xét danh sách các emoji thuộc thẻ (tag) "Blobs", ta có thể thấy danh sách các emoji được thể hiện trong giao diện người dùng như ở hình 8.



Hình 8: Danh sách các emojis có tag Blobs của một trang (page) được đổi chiếu giữa giao diện web và cấu trúc HTML.

Các ô ảnh này đều được thể hiện bằng một thẻ div trong file HTML của trang web với cùng một class. Vì vậy, khi có được đường dẫn của một trang danh sách emoji, ta hoàn toàn có thể sử dụng code để trích xuất toàn bộ các đường dẫn ảnh cũng như một số thông tin khác về ảnh. Theo đó, ta sẽ có đoạn code trích xuất toàn bộ đường dẫn của một trang page như sau:

```

1 def get_image_links_from_page(page_url, driver):
2     driver.get(page_url)
3     try:
4         container = wait.until(EC.presence_of_element_located(
5             (By.CSS_SELECTOR, "div.FS5UE28h.container"))
6     )
7     image_items = wait.until(EC.presence_of_all_elements_located(
8         (By.CSS_SELECTOR, "div.LQY5mtmC div.aLnnpRah.text-center"))
9     )
10
11     image_links = []
12     for img_elem in image_items:
13         img_div = img_elem.find_element(By.CSS_SELECTOR, "div.Mw1EAtrx img")
14
15         img_url = img_div.get_attribute("src")
16         img_title = img_div.get_attribute("title")
17         if img_url:
18             image_links.append((img_url, img_title))
19
20     return image_links
21 except Exception as e:
22     print(f"Error while trying to extract images: {e}")
23     return []

```

### II.1.5. Xây dựng hàm kiểm tra ảnh trùng

Dối với trang này, có một điểm cần lưu ý là ta có thể sẽ bắt gặp các ảnh emoji trùng khi thực hiện thu thập dữ liệu. Vì vậy, để tránh tình trạng lưu trữ các ảnh trùng, ta xây dựng một hàm kiểm tra ảnh trùng dựa vào đường dẫn ảnh với thư viện **hashlib** như sau:

```

1 def hash_image_content(url):
2     try:
3         response = requests.get(url, stream=True)
4         if response.status_code == 200:
5             return hashlib.md5(response.content).hexdigest()
6         else:
7             print(f"Error downloading image from {url}; status: {response.
8                   status_code}")
8             return None
9     except requests.exceptions.RequestException as e:
10        print(f"Error with the image download for {url}: {e}")
11        return None

```

### II.1.6. Xây dựng hàm đổi định dạng ảnh

Khi quan sát đường dẫn ảnh trong trang web, ta có thể nhận thấy định dạng của các ảnh này thuộc đuôi .webp. Vì vậy, ta cần thực hiện chuyển đổi về định dạng ảnh quen thuộc là .jpg để có thể tương tác với ảnh dễ dàng hơn. Theo đó, ta triển khai hàm `convert_webp_to_jpg()` như sau:

```

1 def convert_webp_to_jpg(webp_data):
2     try:
3         img = Image.open(BytesIO(webp_data))
4         if img.format == 'WEBP':
5             if img.mode == 'RGBA':
6                 img = img.convert('RGB')
7                 buffer = BytesIO()
8                 img.save(buffer, format="JPEG")
9                 return buffer.getvalue()
10            else:
11                return webp_data
12    except Exception as e:
13        print(f"Error converting WebP to JPG: {e}")
14        return webp_data

```

### II.1.7. Xây dựng hàm tải ảnh

Để tải được ảnh từ một đường dẫn chứa ảnh, ta sẽ xây dựng một hàm tải ảnh sử dụng thư viện `requests` như sau:

```

1 def download_image(img_url, img_name, folder_path):
2     try:
3         response = requests.get(img_url, stream=True)
4         if response.status_code == 200:
5             img_path = os.path.join(folder_path, f"{img_name}")
6             img_data = convert_webp_to_jpg(response.content)
7             with open(img_path, "wb") as f:

```

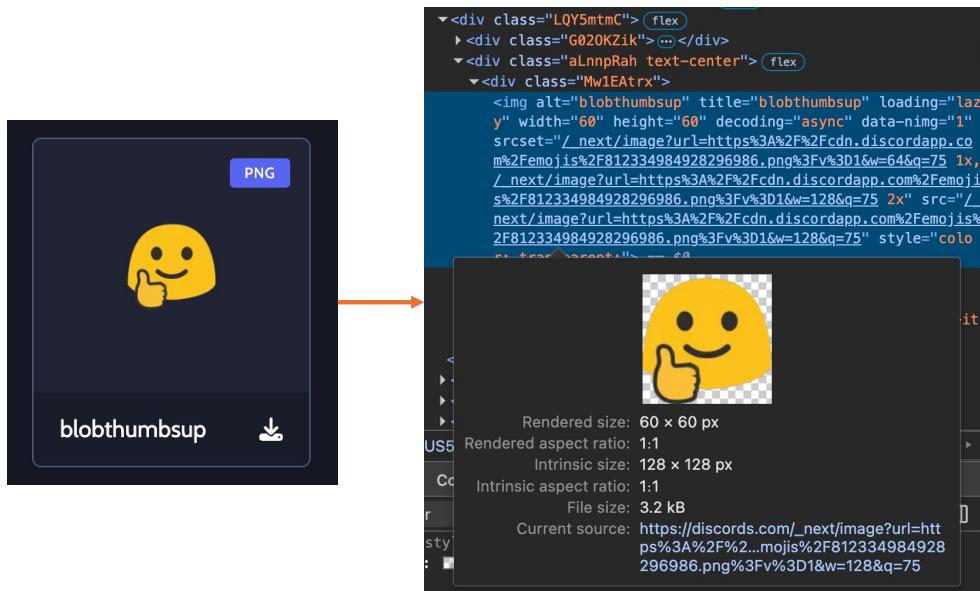
```

8         f.write(img_data)
9     else:
10        print(f"Error downloading image from {img_url}; status: {response.
11                                         status_code}")
12    except requests.exceptions.RequestException as e:
13        print(f"Error with the image download for {img_url}: {e}")

```

### II.1.8. Xây dựng hàm thu thập thông tin của một ảnh

Tổng hợp tất cả các hàm đang có từ phía trên, ta xây dựng một hàm xử lý thông tin của một đường dẫn ảnh. Theo đó bao gồm việc kiểm tra ảnh trùng, thực hiện tải ảnh và lưu các thông tin ảnh dưới dạng dictionary, thuận tiện cho việc lưu thành file .csv ở các đoạn code tiếp theo. Như vậy, ta có code triển khai như sau:



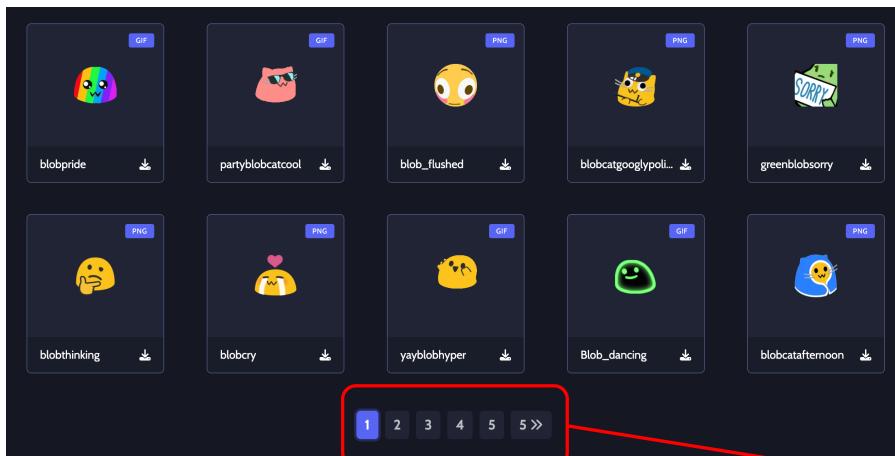
Hình 9: Một số thông tin của một ảnh emoji được thể hiện qua thẻ img trong file HTML.

```

1 def process_image_page(image_url, img_title, folder_path, idx, tag,
2                         seen_hashes):
3     img_hash = hash_image_content(image_url)
4     if img_hash and img_hash not in seen_hashes:
5         seen_hashes.add(img_hash)
6         new_file_name = f"{tag}_{idx:07d}.jpg"
7         download_image(image_url, new_file_name, folder_path)
8         metadata = {
9             "file_name": new_file_name,
10            "image_url": image_url,
11            "image_title": img_title,
12            "tag": tag
13        }
14        return metadata
15    else:
16        return None

```

## II.1.9. Xây dựng hàm duyệt qua từng trang web



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

Hình 10: Tận dụng tham số query bên trong đường dẫn trang chứa các emojis để có thể tự động duyệt qua các trang tiếp theo.

```

1 def loop_over_pages(base_url, tags, total_pages, driver, folder_path):
2     os.makedirs(folder_path, exist_ok=True)
3     all_metadata = []
4     seen_hashes = set()
5
6     for tag in tags:
7         all_images = []
8
9         for page in tqdm(range(1, total_pages + 1), desc=f"Extracting Images
10                           for {tag}", unit="page"):
11             page_url = f"{base_url}/emoji-list/tag/{tag}?page={page}"
12             images = get_image_links_from_page(page_url, driver)
13             all_images.extend(images)
14
15             time.sleep(1)
16
17             metadata_list = []
18             for idx, (img_url, img_title) in enumerate(all_images, start=1):
19                 metadata = process_image_page(img_url, img_title, folder_path, idx
20                                               , tag, seen_hashes)
21
22                 if metadata:
23                     metadata_list.append(metadata)
24
25             all_metadata.extend(metadata_list)
26
27     return all_metadata

```

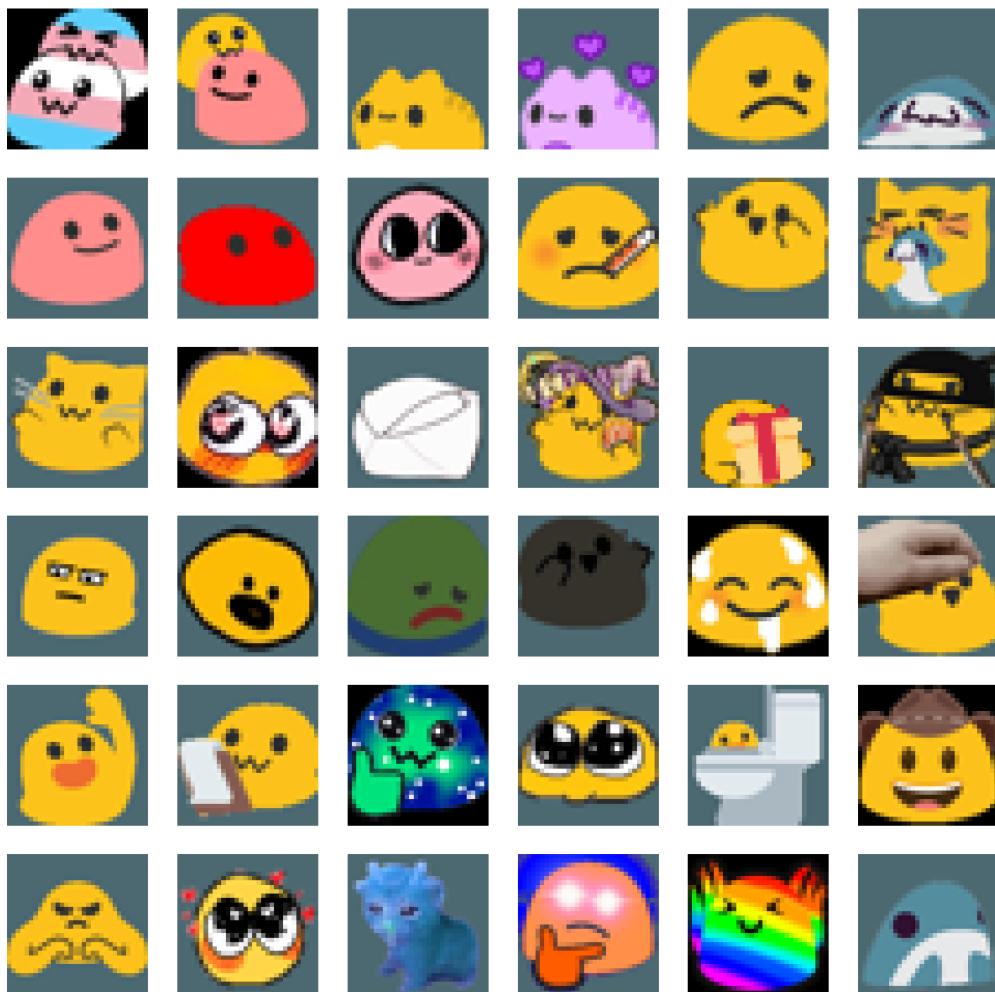
Với hàm trên, ta cho phép xử lý một danh sách các tag emoji khác nhau, song trong project này chúng ta sẽ chỉ tập trung vào tag **Blobs**. Với mỗi tag sẽ được thông qua hai vòng lặp, một dùng để trích xuất toàn bộ các thông tin ảnh và hai là thực hiện xử lý thông tin đó dựa trên các hàm

đã xây dựng ở những bước trước. Khi kết thúc, ta trả về một list chứa metadata của các ảnh emoji.

### II.1.10. Xây dựng hàm lưu thông tin metadata

Với list các metadata trả về từ hàm `loop_over_pages()`, ta thực hiện lưu lại thành file .csv thông qua hàm sau:

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



Hình 11: Trực quan hóa một số hình ảnh mà chúng ta đã thu thập được.

### II.1.11. Thực hiện thu thập dữ liệu

Cuối cùng, với toàn bộ các hàm trên, ta tổng hợp lại để tiến hành thực hiện thu thập dữ liệu ảnh emoji với code triển khai sau đây:

```
1 os.makedirs("crawled_data", exist_ok=True)
```

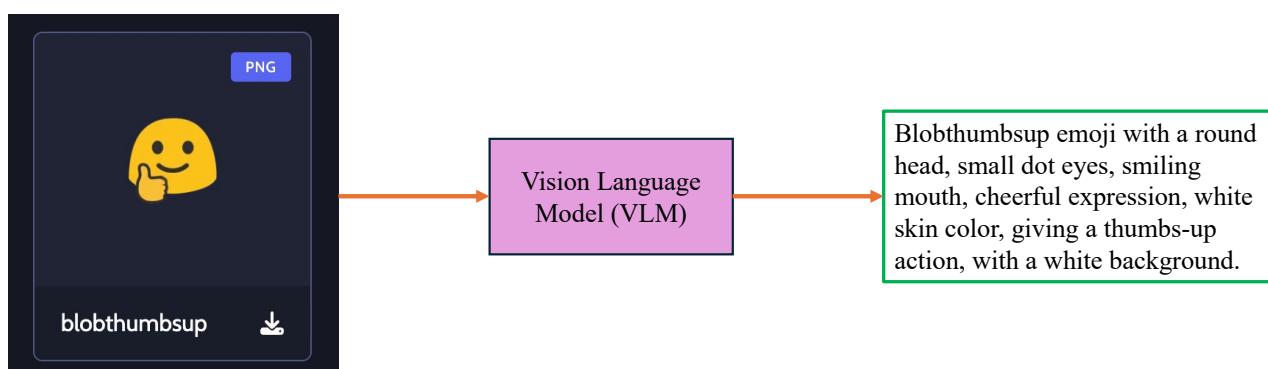
```

2 folder_path = os.path.join("crawled_data", "images")
3 metadata_file = os.path.join("crawled_data", "metadata.csv")
4
5 base_url = "https://discords.com"
6 tags = ["Blobs"]
7 total_pages = 1000
8
9 metadata_list = loop_over_pages(base_url, tags, total_pages, driver,
10                                 folder_path)
11 save_metadata(metadata_list, metadata_file)
12
13 print("Start downloading images...")
14 with tqdm(total=len(metadata_list), desc="Downloading Images", unit="image")
15     as pbar:
16         for metadata in metadata_list:
17             img_url = metadata['image_url']
18             file_name = metadata['file_name']
19             download_image(img_url, file_name, folder_path)
20             pbar.update(1)
21
22 print("Download images completed.")
23 total_crawled_images = len(os.listdir(folder_path))
24 print(f"Total crawled images: {total_crawled_images}.")
25 driver.quit()

```

### II.1.12. Tạo mô tả cho ảnh emoji

Vì mô hình Stable Diffusion (SD) với input là một câu prompt mô tả ảnh muốn tạo sinh, trong khi dữ liệu ảnh hiện tại chưa có thông tin mô tả chi tiết. Một cách để chúng ta có thể giải quyết vấn đề này nhanh chóng đó là tận dụng một mô hình ngôn ngữ lớn để giúp chúng ta tạo nhanh các mô tả thông qua prompting.



Hình 12: Sử dụng mô hình ngôn ngữ thị giác lớn (VLM) để tạo mô tả cho các ảnh emoji.

Song, trong nội dung file mô tả của project, chúng ta sẽ không đề cập chi tiết về phần sử dụng mô hình ngôn ngữ thị giác lớn. Thay vào đó, bộ dữ liệu emoji với mô tả đính kèm sẽ được cung cấp sẵn.

## II.2. Xây dựng mô hình Stable Diffusion

Trong phần này, ta sẽ thực hiện triển khai mô hình Stable Diffusion sử dụng thư viện PyTorch và Diffusers của HuggingFace để huấn luyện mô hình trên bộ dữ liệu emoji mà chúng ta đã chuẩn bị. Các bước thực hiện như sau:

### II.2.1. Import các thư viện cần thiết

```

1 import os
2 import math
3 import torch
4 import random
5 import numpy as np
6 import pandas as pd
7 import matplotlib.pyplot as plt
8
9 from torch import nn
10 from PIL import Image
11 from tqdm import tqdm
12 from diffusers import AutoencoderKL
13 from torch.nn import functional as F
14 from torchvision import transforms
15 from torch.utils.data import Dataset, DataLoader
16 from transformers import CLIPTokenizer, CLIPTextModel
17 import torch.optim.lr_scheduler as lr_scheduler
18 from torch.amp import GradScaler, autocast

```

### II.2.2. Cố định tham số ngẫu nhiên

Nhằm mục đích có thể tái tạo lại kết quả đã đạt được trong mỗi lần chạy lại chương trình, ta thực hiện cố định trạng thái ngẫu nhiên cho toàn bộ các hàm, module có liên quan đến các phép ngẫu nhiên như sau:

```

1 def set_seed(seed=42):
2     random.seed(seed)
3     np.random.seed(seed)
4     torch.manual_seed(seed)
5     torch.cuda.manual_seed(seed)
6     torch.cuda.manual_seed_all(seed)
7     torch.backends.cudnn.deterministic = True
8     torch.backends.cudnn.benchmark = False
9     os.environ["PYTHONHASHSEED"] = str(seed)
10    print(f"Seed set to {seed}")
11
12 set_seed()
13 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

```

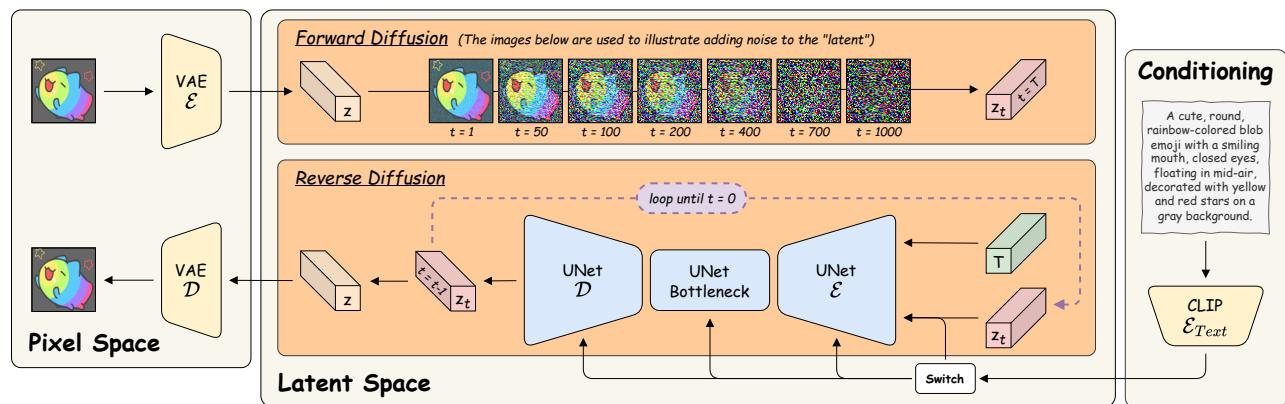
### II.2.3. Tải bộ dữ liệu

Chúng ta thực hiện tải bộ dữ liệu đã thu thập được ở phần trước vào notebook mới này thông qua lệnh sau:

```

1 # https://drive.google.com/file/d/15Z_F4Dwgb3NLqEGnVMUEJqyxXgW7Gx-h/view?usp=sharing
2 !gdown 15Z_F4Dwgb3NLqEGnVMUEJqyxXgW7Gx-h
3 !unzip blobs_crawled_data.zip

```



Hình 13: Minh họa tổng quát kiến trúc mạng Stable Diffusion cho bài toán tạo sinh hình emoji.

#### II.2.4. Xây dựng các class Attention

Trong Stable Diffusion có bao gồm kiến trúc U-Net. Kiến trúc này có tận dụng các phép Attention nhằm cải thiện khả xử lý mối quan hệ không gian trong ảnh và vấn đề phụ thuộc dài hạn trong quá trình tạo sinh ảnh. Vì vậy, ở bước này, ta cần định nghĩa các lớp Attention bao gồm SelfAttention và CrossAttention như sau:

```

1 class SelfAttention(nn.Module):
2     def __init__(self, num_attn_heads, hidden_dim, in_proj_bias=True,
3                  out_proj_bias=True):
4         super().__init__()
5         self.num_heads = num_attn_heads
6         self.head_size = hidden_dim // num_attn_heads
7
7         self.qkv_proj = nn.Linear(hidden_dim, 3 * hidden_dim, bias=
8                         in_proj_bias)
9         self.output_proj = nn.Linear(hidden_dim, hidden_dim, bias=
10                         out_proj_bias)
11
12     def forward(self, features, use_causal_mask=False):
13         b, s, d = features.shape
14
15         qkv_combined = self.qkv_proj(features)
16         q_mat, k_mat, v_mat = torch.chunk(qkv_combined, 3, dim=-1)
17
18         q_mat = q_mat.view(b, s, self.num_heads, self.head_size).permute(0, 2,
19                           1, 3)
20         k_mat = k_mat.view(b, s, self.num_heads, self.head_size).permute(0, 2,
21                           1, 3)
22         v_mat = v_mat.view(b, s, self.num_heads, self.head_size).permute(0, 2,
23                           1, 3)

```

```

20     qk = torch.matmul(q_mat, k_mat.transpose(-2, -1))
21     sqrt_qk = qk / math.sqrt(self.head_size)
22
23     if use_causal_mask:
24         causal_mask = torch.triu(torch.ones_like(sqrt_qk, dtype=torch.bool),
25                                  diagonal=1)
25         sqrt_qk = sqrt_qk.masked_fill(causal_mask, -torch.inf)
26
27     attn_weights = torch.softmax(sqrt_qk, dim=-1)
28     attn_values = torch.matmul(attn_weights, v_mat)
29
30     attn_values = attn_values.permute(0, 2, 1, 3).contiguous()
31     attn_values = attn_values.view(b, s, d)
32
33     final_output = self.output_proj(attn_values)
34     return final_output

```

```

1 class CrossAttention(nn.Module):
2     def __init__(self, num_attn_heads, query_dim, context_dim, in_proj_bias=True,
3                  out_proj_bias=True):
4         super().__init__()
5         self.num_heads = num_attn_heads
6         self.head_size = query_dim // num_attn_heads
7
7         self.query_map = nn.Linear(query_dim, query_dim, bias=in_proj_bias)
8         self.key_map = nn.Linear(context_dim, query_dim, bias=in_proj_bias)
9         self.value_map = nn.Linear(context_dim, query_dim, bias=in_proj_bias)
10
11         self.output_map = nn.Linear(query_dim, query_dim, bias=out_proj_bias)
12
13     def forward(self, query_input, context_input):
14         b_q, s_q, d_q = query_input.shape
15         _, s_kv, _ = context_input.shape
16
17         q_mat = self.query_map(query_input)
18         k_mat = self.key_map(context_input)
19         v_mat = self.value_map(context_input)
20
21         q_mat = q_mat.view(b_q, s_q, self.num_heads, self.head_size).permute(0
22                               , 2, 1, 3)
22         k_mat = k_mat.view(b_q, s_kv, self.num_heads, self.head_size).permute(
23                               0, 2, 1, 3)
23         v_mat = v_mat.view(b_q, s_kv, self.num_heads, self.head_size).permute(
24                               0, 2, 1, 3)
24
25         qk = torch.matmul(q_mat, k_mat.transpose(-2, -1))
26         sqrt_qk = qk / math.sqrt(self.head_size)
27         attn_weights = torch.softmax(sqrt_qk, dim=-1)
28
29         attn_values = torch.matmul(attn_weights, v_mat)
30         attn_values = attn_values.permute(0, 2, 1, 3).contiguous()
31         attn_values = attn_values.view(b_q, s_q, d_q)
32
33         final_output = self.output_map(attn_values)
34         return final_output

```

## II.2.5. Khai báo class DDPM

Một thành phần không thể thiếu đối với cài đặt của Stable Diffusion đó là DDPM (Denoising Diffusion Probabilistic Models). Để triển khai thành phần này, ta thực hiện như sau:

```
1 class DDPMScheduler:
2     def __init__(self,
3                  random_generator,
4                  train_timesteps=1000,
5                  diffusion_beta_start=0.00085,
6                  diffusion_beta_end=0.012
7                  ):
8
9
10    self.betas = torch.linspace(
11        diffusion_beta_start ** 0.5, diffusion_beta_end ** 0.5,
12        train_timesteps,
13        dtype=torch.float32) ** 2
14    self.alphas = 1.0 - self.betas
15    self.alphas_cumulative_product = torch.cumprod(self.alphas, dim=0)
16    self.one_val = torch.tensor(1.0)
17    self.prng_generator = random_generator
18    self.total_train_timesteps = train_timesteps
19    self.schedule_timesteps = torch.from_numpy(np.arange(0,
20                                                       train_timesteps)[::-1].copy())
21
22    def set_steps(self, num_sampling_steps=50):
23        self.num_sampling_steps = num_sampling_steps
24        step_scaling_factor = self.total_train_timesteps // self.
25                                         num_sampling_steps
26        timesteps_for_sampling = (
27            np.arange(0, num_sampling_steps) * step_scaling_factor
28        ).round()[::-1].copy().astype(np.int64)
29        self.schedule_timesteps = torch.from_numpy(timesteps_for_sampling)
30
31    def _get_prior_timestep(self, current_timestep):
32        previous_t = current_timestep - self.total_train_timesteps // self.
33                                         num_sampling_steps
34        return previous_t
35
36    def _calculate_variance(self, timestep):
37        prev_t = self._get_prior_timestep(timestep)
38        alpha_cumprod_t = self.alphas_cumulative_product[timestep]
39        alpha_cumprod_t_prev = self.alphas_cumulative_product[prev_t] if
40                                prev_t >= 0 else self.one_val
41        beta_t_current = 1 - alpha_cumprod_t / alpha_cumprod_t_prev
42        variance_value = (1 - alpha_cumprod_t_prev) / (1 - alpha_cumprod_t) *
43                           beta_t_current
44        variance_value = torch.clamp(variance_value, min=1e-20)
45        return variance_value
46
47    def adjust_strength(self, strength_level=1):
48        initial_step_index = self.num_sampling_steps - int(self.
49                                         num_sampling_steps * strength_level)
50        self.schedule_timesteps = self.schedule_timesteps[initial_step_index:]
51        self.start_sampling_step = initial_step_index # Lu li bc
```

```

        b   t      u
45
46     def step(self, current_t, current_latents, model_prediction):
47         t = current_t
48         prev_t = self._get_prior_timestep(t)
49
50         alpha_cumprod_t = self.alphas_cumulative_product[t]
51         alpha_cumprod_t_prev = self.alphas_cumulative_product[prev_t] if
52                         prev_t >= 0 else self.one_val
53         beta_cumprod_t = 1 - alpha_cumprod_t
54         beta_cumprod_t_prev = 1 - alpha_cumprod_t_prev
55         alpha_t_current = alpha_cumprod_t / alpha_cumprod_t_prev
56         beta_t_current = 1 - alpha_t_current
57
57         predicted_original = (current_latents - beta_cumprod_t ** 0.5 *
58                               model_prediction) / alpha_cumprod_t **
59                               0.5
60
61         original_coeff = (alpha_cumprod_t_prev ** 0.5 * beta_t_current) /
62                           beta_cumprod_t
63         current_coeff = alpha_t_current ** 0.5 * beta_cumprod_t_prev /
64                           beta_cumprod_t
65
62         predicted_prior_mean = original_coeff * predicted_original +
63                           current_coeff * current_latents
64
64         variance_term = 0
65         if t > 0:
66             target_device = model_prediction.device
67             noise_component = torch.randn(
68                 model_prediction.shape,
69                 generator=self.prng_generator,
70                 device=target_device,
71                 dtype=model_prediction.dtype
72             )
73             variance_term = (self._calculate_variance(t) ** 0.5) *
74                           noise_component
75
75         predicted_prior_sample = predicted_prior_mean + variance_term
76         return predicted_prior_sample
77
78     def add_noise(self, initial_samples, noise_timesteps):
79         alphas_cumprod = self.alphas_cumulative_product.to(
80             device=initial_samples.device,
81             dtype=initial_samples.dtype
82         )
83         noise_timesteps = noise_timesteps.to(initial_samples.device)
84         sqrt_alpha_cumprod = alphas_cumprod=noise_timesteps] ** 0.5
85         sqrt_alpha_cumprod = sqrt_alpha_cumprod.view(
86             sqrt_alpha_cumprod.shape[0], *[1] * (initial_samples.ndim - 1))
87         )
88         sqrt_one_minus_alpha_cumprod = (1 - alphas_cumprod=noise_timesteps])
89                         ** 0.5
90         sqrt_one_minus_alpha_cumprod = sqrt_one_minus_alpha_cumprod.view(
91             sqrt_one_minus_alpha_cumprod.shape[0], *[1] * (initial_samples.

```

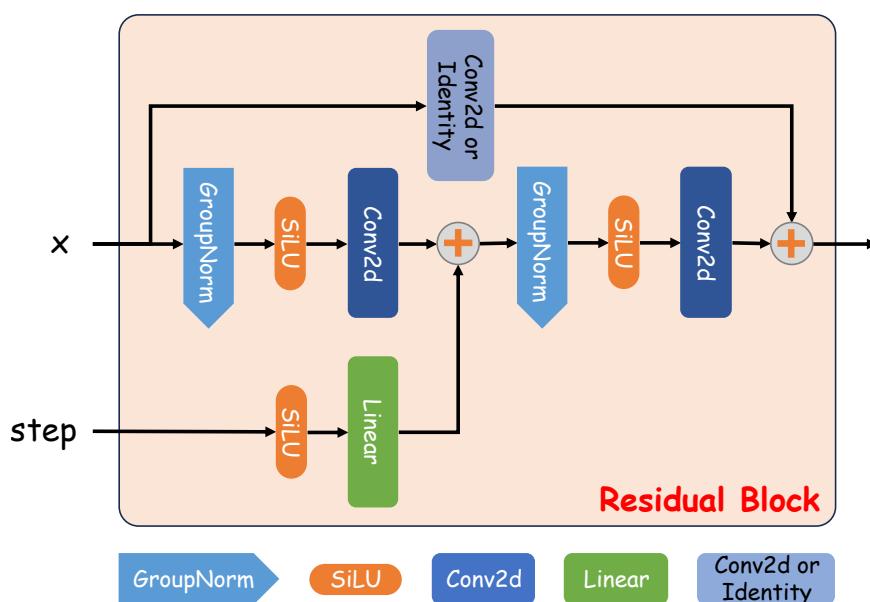
```

91         ndim - 1))
92     )
93     random_noise = torch.randn(
94         initial_samples.shape, generator=self.prng_generator,
95         device=initial_samples.device, dtype=initial_samples.dtype
96     )
97     noisy_result = sqrt_alpha_cumprod * initial_samples +
98                     sqrt_one_minus_alpha_cumprod *
99                     random_noise
100
101    return noisy_result, random_noise

```

## II.2.6. Khai báo kiến trúc U-Net

Với các class đã khai báo bên trên, ta tiến hành định nghĩa một số các thành phần liên quan đến kiến trúc U-Net được sử dụng trong Stable Diffusion như sau:



Hình 14: Minh họa kiến trúc của khối Residual trong Unet.

```

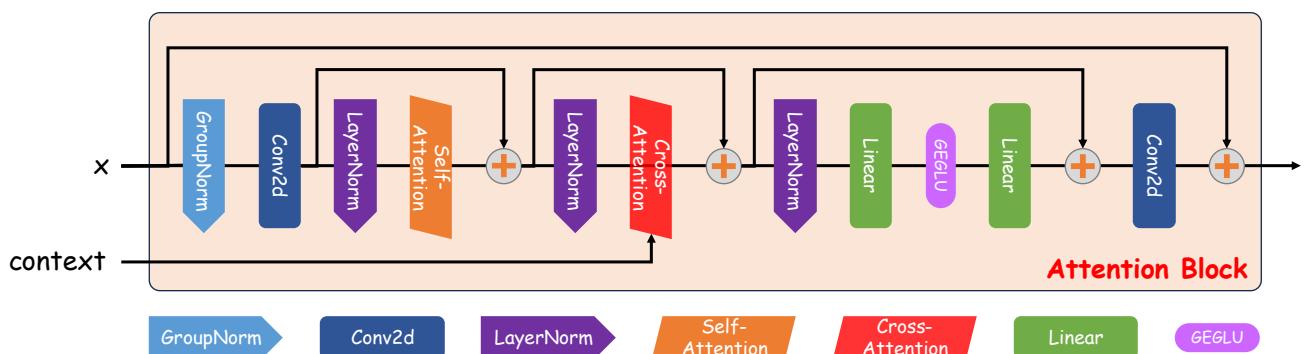
1  class UNET_ResidualBlock(nn.Module):
2      def __init__(self, in_channels, out_channels, time_dim=1280):
3          super().__init__()
4          self.gn_feature = nn.GroupNorm(32, in_channels)
5          self.conv_feature = nn.Conv2d(in_channels, out_channels, kernel_size=3
6                                      , padding=1)
7          self.time_embedding_proj = nn.Linear(time_dim, out_channels)
8
8          self.gn_merged = nn.GroupNorm(32, out_channels)
9          self.conv_merged = nn.Conv2d(out_channels, out_channels, kernel_size=3
10                                     , padding=1)
11
11         if in_channels == out_channels:
12             self.residual_connection = nn.Identity()
13         else:

```

```

14         self.residual_connection = nn.Conv2d(in_channels, out_channels,
15                                         kernel_size=1, padding=0)
16
17     def forward(self, input_feature, time_emb):
18         residual = input_feature
19
20         h = self.gn_feature(input_feature)
21         h = F.silu(h)
22         h = self.conv_feature(h)
23
24         time_emb_processed = F.silu(time_emb)
25         time_emb_projected = self.time_embedding_proj(time_emb_processed)
26         time_emb_projected = time_emb_projected.unsqueeze(-1).unsqueeze(-1)
27
28         merged_feature = h + time_emb_projected
29         merged_feature = self.gn_merged(merged_feature)
30         merged_feature = F.silu(merged_feature)
31         merged_feature = self.conv_merged(merged_feature)
32
33         output = merged_feature + self.residual_connection(residual)
34         return output

```



Hình 15: Minh họa kiến trúc của khối Attention trong Unet.

```

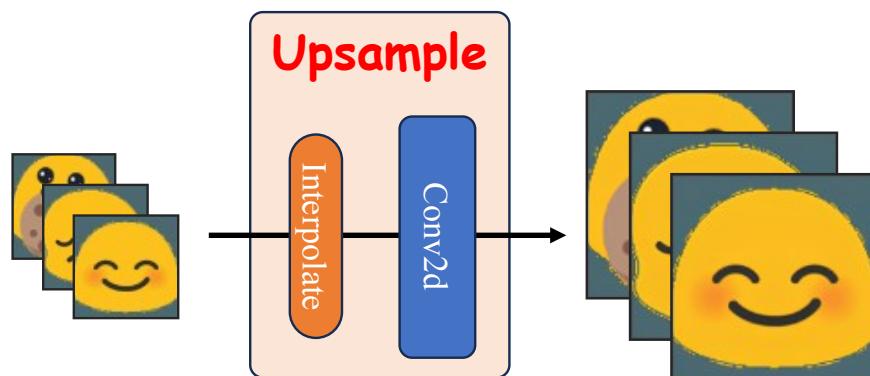
1 class UNET_AttentionBlock(nn.Module):
2     def __init__(self, num_heads, head_dim, context_dim=512):
3         super().__init__()
4         embed_dim = num_heads * head_dim
5
6         self.gn_in = nn.GroupNorm(32, embed_dim, eps=1e-6)
7         self.proj_in = nn.Conv2d(embed_dim, embed_dim, kernel_size=1, padding=
8             0)
9
9         self.ln_1 = nn.LayerNorm(embed_dim)
10        self.attn_1 = SelfAttention(num_heads, embed_dim, in_proj_bias=False)
11        self.ln_2 = nn.LayerNorm(embed_dim)
12        self.attn_2 = CrossAttention(num_heads, embed_dim, context_dim,
13                                     in_proj_bias=False)
14        self.ln_3 = nn.LayerNorm(embed_dim)
15
15        self.ffn_geglu = nn.Linear(embed_dim, 4 * embed_dim * 2)

```

```

16     self.ffn_out = nn.Linear(4 * embed_dim, embed_dim)
17     self.proj_out = nn.Conv2d(embed_dim, embed_dim, kernel_size=1, padding
18                             =0)
19
20     def forward(self, input_tensor, context_tensor):
21         skip_connection = input_tensor
22
23         B, C, H, W = input_tensor.shape
24         HW = H * W
25
26         h = self.gn_in(input_tensor)
27         h = self.proj_in(h)
28         h = h.view(B, C, HW).transpose(-1, -2)
29
30         attn1_skip = h
31         h = self.ln_1(h)
32         h = self.attn_1(h)
33         h = h + attn1_skip
34
35         attn2_skip = h
36         h = self.ln_2(h)
37         h = self.attn_2(h, context_tensor)
38         h = h + attn2_skip
39
40         ffn_skip = h
41         h = self.ln_3(h)
42         intermediate, gate = self.ffn_gelu(h).chunk(2, dim=-1)
43         h = intermediate * F.gelu(gate)
44         h = self.ffn_out(h)
45         h = h + ffn_skip
46
47         h = h.transpose(-1, -2).view(B, C, H, W)
48         output = self.proj_out(h) + skip_connection
49         return output

```



Hình 16: Minh họa kiến trúc của khối Upsample.

```

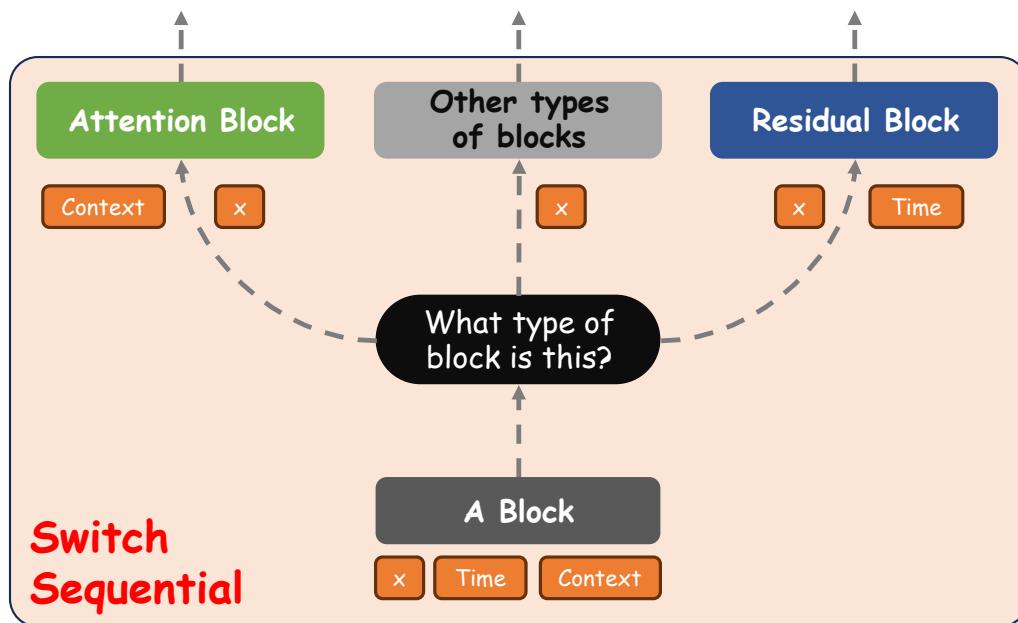
1 class Upsample(nn.Module):
2     def __init__(self, num_channels):
3         super().__init__()

```

```

4         self.conv = nn.Conv2d(num_channels, num_channels, kernel_size=3,
5                               padding=1)
6
7     def forward(self, feature_map):
8         x = F.interpolate(feature_map, scale_factor=2, mode='nearest')
9         x = self.conv(x)
10        return x

```



Hình 17: Minh họa kiến trúc của khối Switch Sequential trong Unet.

```

1 class SwitchSequential(nn.Sequential):
2     def forward(self, x, guidance_context, time_embedding):
3         for module_instance in self:
4             if isinstance(module_instance, UNET_AttentionBlock):
5                 x = module_instance(x, guidance_context)
6             elif isinstance(module_instance, UNET_ResidualBlock):
7                 x = module_instance(x, time_embedding)
8             else:
9                 x = module_instance(x)
10            return x
11
12
13 class TimeEmbedding(nn.Module):
14     def __init__(self, n_embd):
15         super().__init__()
16         self.proj1 = nn.Linear(n_embd, 4 * n_embd)
17         self.proj2 = nn.Linear(4 * n_embd, 4 * n_embd)
18
19     def forward(self, x):
20         x = self.proj1(x)
21         x = F.silu(x)
22         x = self.proj2(x)

```

23 | return x

## II.2.7. Xây dựng hàm mã hóa thông tin thời gian

Để đưa thông tin thời gian (timestep) và ngữ cảnh điều kiện vào quá trình Denoising, ta sẽ triển khai các hàm sau với mục tiêu mã hóa thông tin timestep thành vector:

```

1 def embed_a_timestep(timestep, embedding_dim=320):
2     half_dim = embedding_dim // 2
3     freqs = torch.exp(-math.log(10000) *
4                         torch.arange(start=0, end=half_dim, dtype=torch.float32) /
5                         half_dim)
6     x = torch.tensor([timestep], dtype=torch.float32)[:, None] * freqs[None]
7     return torch.cat([torch.cos(x), torch.sin(x)], dim=-1)
8
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)

```

## II.2.8. Khai báo Diffusion model

```

1 class Diffusion(nn.Module):
2     def __init__(self, h_dim=128, n_head=4):
3         super().__init__()
4         self.time_embedding = TimeEmbedding(320)
5         self.unet = UNET(h_dim, n_head)
6         self.unet_output = UNETOutputLayer(h_dim, 4)
7
8         @torch.autocast(
9             device_type='cuda', dtype=torch.float16,
10             enabled=True, cache_enabled=True
11         )
12     def forward(self, latent, context, time):
13         time = self.time_embedding(time)
14         output = self.unet(latent, context, time)
15         output = self.unet_output(output)
16         return output

```

## II.2.9. Khai báo mô hình CLIP

Để mã hóa thông tin mô tả của một ảnh thành dạng vector, ta sử dụng mô hình CLIP. Code cài đặt như sau:

```

1 class CLIPTextEncoder(nn.Module):
2     def __init__(self):
3         super().__init__()
4         CLIP_id = "openai/clip-vit-base-patch32"

```

```

5     self.tokenizer = CLIPTokenizer.from_pretrained(CLIP_id)
6     self.text_encoder = CLIPTextModel.from_pretrained(CLIP_id)
7     self.device = "cuda" if torch.cuda.is_available() else "cpu"
8
9     for param in self.text_encoder.parameters():
10        param.requires_grad = False
11
12    self.text_encoder.eval()
13    self.text_encoder.to(self.device)
14
15  def forward(self, prompts):
16      inputs = self.tokenizer(
17          prompts,
18          padding="max_length",
19          truncation=True,
20          max_length=self.text_encoder.config.max_position_embeddings,
21          return_tensors="pt"
22      )
23      input_ids = inputs.input_ids.to(self.device)
24      attention_mask = inputs.attention_mask.to(self.device)
25
26      with torch.no_grad():
27          text_encoder_output = self.text_encoder(
28              input_ids=input_ids,
29              attention_mask=attention_mask
30          )
31      last_hidden_states = text_encoder_output.last_hidden_state
32
33  return last_hidden_states

```

### II.2.10. Khai báo mô hình pre-trained VAE

Để mã hóa ảnh đầu vào về dạng không gian tiệm ẩn và và giải mã không gian tiệm ẩn được tạo từ quá trình ngược của Diffusion thành ảnh, ta sẽ sử dụng mô hình Variational Autoencoder (VAE). Tại đây, ta tận dụng một pre-trained model có sẵn từ HuggingFace như sau:

```

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

```

### II.2.11. Khai báo hàm trực quan hóa ảnh và scale giá trị ảnh

```

1 def show_images(images, title="", titles=[]):
2     plt.figure(figsize=(8, 8))
3     for i in range(min(25, len(images))):
4         plt.subplot(5, 5, i+1)
5         img = images[i].permute(1, 2, 0).cpu().numpy()
6         plt.imshow(img)
7         if titles:
8             plt.title(titles[i])
9             plt.axis("off")
10            plt.suptitle(title)

```

```

11     plt.tight_layout()
12     plt.show()
13
14
15 def rescale(value, in_range, out_range, clamp=False):
16     in_min, in_max = in_range
17     out_min, out_max = out_range
18
19     in_span = in_max - in_min
20     out_span = out_max - out_min
21
22     scaled_value = (value - in_min) / (in_span + 1e-8)
23     rescaled_value = out_min + (scaled_value * out_span)
24
25     if clamp:
26         rescaled_value = torch.clamp(
27             rescaled_value,
28             out_min, out_max
29         )
30
31     return rescaled_value

```

### II.2.12. Khai báo class PyTorch dataset

Ta xây dựng class PyTorch dataset cho bộ ảnh-mô tả về emoji đã thu thập được như sau:

```

1 WIDTH, HEIGHT = 32, 32
2 batch_size = 32
3
4 class EmojiDataset(Dataset):
5     def __init__(self, csv_files, image_folder, transform=None):
6         self.dataframe = pd.concat([pd.read_csv(csv_file) for csv_file in
7                                     csv_files])
8         self.images_folder = image_folder
9         self.dataframe["image_path"] = self.dataframe["file_name"].str.replace
10            ("\\\", "/")
11         self.image_paths = self.dataframe["image_path"].tolist()
12         self.titles = self.dataframe["prompt"].tolist()
13         self.transform = transform
14
15     def __len__(self):
16         return len(self.dataframe)
17
18     def __getitem__(self, idx):
19         image_path = self.images_folder + "/" + self.image_paths[idx]
20         title = self.titles[idx]
21         title = title.replace(' ', '').replace(';', '')
22         image = Image.open(image_path).convert('RGB')
23
24         if self.transform:
25             image = self.transform(image)
26
27         return image, title

```

Với class trên, ta khởi tạo DataLoader dùng để huấn luyện mô hình như sau (các bạn lưu ý thay đổi đường dẫn thư mục dataset cho phù hợp với máy của các bạn):

```
1 transform = transforms.Compose([
2     transforms.Resize(
3         (WIDTH, HEIGHT),
4         interpolation=transforms.InterpolationMode.BICUBIC
5     ),
6     transforms.ToTensor(),
7     transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
8 ])
9
10 csv_files = ['/content/blobs_crawled_data/metadata.csv']
11 image_folder = '/content/blobs_crawled_data/images'
12
13 train_dataset = EmojiDataset(
14     csv_files=csv_files,
15     image_folder=image_folder,
16     transform=transform
17 )
18 train_dataloader = DataLoader(
19     train_dataset,
20     batch_size=batch_size,
21     shuffle=True,
22     num_workers=2,
23     pin_memory=True,
24     persistent_workers=True
25 )
```

### II.2.13. Thực hiện huấn luyện

Trước khi thực hiện huấn luyện, ta đóng gói phần cài đặt huấn luyện mô hình vào thành một hàm nhằm thuận tiện trong việc sử dụng code:

```
1 def train(diffusion, vae, text_encoder, scheduler,
2         optimizer, lr_scheduler, scaler,
3         criterion, dataloader, num_epochs, device="cuda"):
4     losses = []
5     for epoch in range(num_epochs):
6         diffusion.train()
7
8         epoch_loss = 0.0
9         progress_bar = tqdm(dataloader, desc=f"Epoch {epoch+1}/{num_epochs}",
10                           leave=False)
11
12         for batch_idx, (images, titles) in enumerate(progress_bar):
13             images = images.to(device)
14             image_titles = [f"A photo of {title}" for title in titles]
15             image_titles = [title if random.random() < 0.5 else "" for title
16                             in image_titles]
17
18             with torch.no_grad():
19                 latents = vae.encode(images).latent_dist.sample() * 0.18215
20
21             timesteps = torch.randint(
22                 0, diffusion.num_timesteps, (latents.shape[0],),
23                 device=latents.device)
```

```

20         0, scheduler.total_train_timesteps,
21         (latents.shape[0],), device=device
22     )
23
24     noisy_latents, noise = scheduler.add_noise(latents, timesteps)
25     time_embeddings = embed_timesteps(timesteps).to(device)
26     text_embeddings = text_encoder(image_titles)
27
28     noise_pred = diffusion(noisy_latents, text_embeddings,
29                             time_embeddings)
30
31     with autocast(device_type="cuda", dtype=torch.float16,
32                    enabled=True, cache_enabled=True):
33         loss = criterion(noise_pred, noise)
34
35     optimizer.zero_grad()
36     scaler.scale(loss).backward()
37     scaler.step(optimizer)
38     scaler.update()
39
40     batch_loss = loss.item()
41     epoch_loss += batch_loss
42
43     progress_bar.set_postfix(loss=f"{batch_loss:.5f}",
44                               lr=f"{optimizer.param_groups[0]['lr']:.6f}")
45
46     lr_scheduler.step()
47     avg_epoch_loss = epoch_loss / len(dataloader)
48     if (epoch + 1) % 10 == 0 or epoch == 0:
49         print(f"Epoch [{epoch+1}/{num_epochs}] - Avg Loss: {avg_epoch_loss:.5f}")
50     losses.append(avg_epoch_loss)
51
52     print("Training finished!")
53     return losses

```

Sau đó, ta khai báo các tham số, các mô hình thành phần và các hàm tham gia vào quá trình huấn luyện và thực hiện lời gọi hàm `train()` để tiến hành việc huấn luyện mô hình Stable Diffusion:

```

1 EPOCHS = 300
2
3 h_dim = 384
4 n_head = 8
5
6 vae = vae.to(device)
7 diffusion = Diffusion(h_dim, n_head).to(device)
8 clip = CLIPTextEncoder().to(device)
9
10 random_generator = torch.Generator(device="cuda")
11 noise_scheduler = DDPMscheduler(random_generator)
12
13 optimizer = torch.optim.AdamW(diffusion.parameters(), lr=1e-4)
14 criterion = torch.nn.MSELoss()
15

```

```

16 lrate_scheduler = lr_scheduler.CosineAnnealingLR(
17     optimizer, T_max=EPOCHS, eta_min=1e-5
18 )
19 scaler = GradScaler()
20
21 def count_parameters(model):
22     return sum(p.numel() for p in model.parameters())
23
24 vae_params = count_parameters(vae)
25 diffusion_params = count_parameters(diffusion)
26 clip_params = count_parameters(clip)
27
28 print(f"VAE parameters: {vae_params}")
29 print(f"Diffusion parameters: {diffusion_params}")
30 print(f"CLIP parameters: {clip_params}")
31 print(f"Total parameters: {vae_params + diffusion_params + clip_params}")
32
33 losses = train(diffusion, vae, clip, noise_scheduler,
34                 optimizer, lrate_scheduler, scaler,
35                 criterion, train_dataloader, EPOCHS, device=device)

```

## II.2.14. Lưu trữ mô hình

Sau khi huấn luyện xong, việc lưu trữ mô hình cần được thực hiện để có thể tái sử dụng ở bất cứ đâu. Theo đó, đoạn code sau cần được thực thi:

```

1 torch.save(diffusion.state_dict(), "Emoji_SD.pth")
2 !zip -r Emoji_SD.zip Emoji_SD.pth

```

## II.3. Triển khai mô hình

### II.3.1. Xây dựng hàm tạo sinh ảnh

Với mô hình đã huấn luyện được ở phần trước, ta hoàn toàn có thể tạo một hàm thực hiện inference mô hình với một câu prompt bất kì nhằm lấy kết quả tạo sinh hình của mô hình như sau:

```

1 LATENTS_WIDTH = WIDTH // 8
2 LATENTS_HEIGHT = HEIGHT // 8
3
4 def generate_image(
5     prompt,
6     diffusion,
7     vae,
8     text_encoder,
9     scheduler,
10    num_inference_steps=100,
11    seed=None,
12    device="cuda" if torch.cuda.is_available() else "cpu",
13 ):
14     rng_generator = torch.Generator(device=device)
15     if seed is None:

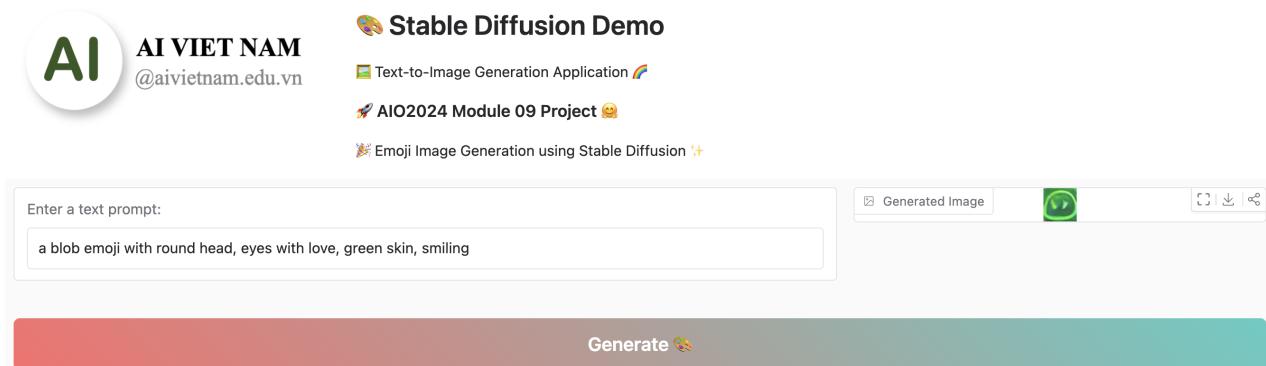
```

```

16     rng_generator.seed()
17 else:
18     rng_generator.manual_seed(seed)
19
20 prompts = [prompt]
21 text_embeddings = text_encoder(prompts).to(device)
22
23 scheduler.set_steps(num_inference_steps)
24
25 latent_shape = (1, 4, LATENTS_HEIGHT, LATENTS_WIDTH)
26
27 noisy_latents = torch.randn(
28     latent_shape, generator=rng_generator, device=device)
29 timesteps = scheduler.schedule_timesteps
30
31 for t in tqdm(timesteps):
32     latent_model_input = noisy_latents
33
34     time_embedding = embed_a_timestep(t).to(device)
35
36     with torch.no_grad():
37         noise_pred = diffusion(
38             latent_model_input,
39             text_embeddings,
40             time_embedding
41         )
42     noisy_latents = scheduler.step(
43         t,
44         noisy_latents,
45         noise_pred
46     )
47
48 final_latents = noisy_latents / 0.18215
49
50 with torch.no_grad():
51     decoded_image_tensor = vae.decode(final_latents).sample
52
53 image_output = rescale(decoded_image_tensor, (-1, 1), (0, 255), clamp=True)
54 image_output = image_output.permute(0, 2, 3, 1).to("cpu", torch.uint8).numpy()
55 return image_output[0]

```

Với hàm trên, ta hoàn toàn có thể ứng dụng vào một ứng dụng web demo về tạo sinh hình ảnh đơn giản để triển khai thành một chương trình hoàn chỉnh như ảnh sau (các bạn có thể trải nghiệm thử web demo tại phần IV.).



Hình 18: Hình ảnh trang web demo cho ứng dụng sinh ảnh biểu tượng sử dụng Stable Diffusion.

### III. Câu hỏi trắc nghiệm:

1. Điểm yếu lớn nhất khiến Diffusion Models truyền thống khó ứng dụng trong thực tế là gì?
  - (a) Chất lượng ảnh thấp.
  - (b) Tốn kém chi phí lưu trữ.
  - (c) Quá trình huấn luyện không ổn định.
  - (d) Tốc độ sinh ảnh chậm.
2. Stable Diffusion thực hiện quá trình khuếch tán trong không gian nào để tăng hiệu quả tính toán?
  - (a) Không gian ảnh RGB.
  - (b) Không gian pixel grayscale.
  - (c) Không gian tiềm ẩn (latent space).
  - (d) Không gian vector hóa của prompt.
3. Trong hàm mất mát của Stable Diffusion, mô hình học để dự đoán:
  - (a) Ảnh gốc  $x$ .
  - (b) Vector tiềm ẩn  $z$ .
  - (c) Điều kiện văn bản  $y$ .
  - (d) Nhiêu  $\epsilon$ .
4. Mô hình nào sau đây được dùng để mã hóa prompt văn bản trong Stable Diffusion?
  - (a) BERT.
  - (b) GPT-2.
  - (c) CLIP Text Encoder.
  - (d) ResNet.
5. Cross-attention trong UNet có vai trò:
  - (a) Tăng khả năng mã hóa không gian ảnh.
  - (b) Truyền đặc trưng từ prompt văn bản vào quá trình sinh ảnh.
  - (c) Thay thế self-attention để tăng tốc độ.
  - (d) Giảm độ lệch thông tin giữa các lớp.
6. Lợi ích chính của việc sử dụng kiến trúc UNet trong quá trình khử nhiễu là gì?
  - (a) Kết hợp đặc trưng ở nhiều cấp độ khác nhau.
  - (b) Giảm số tham số mô hình.
  - (c) Tự động hóa quá trình huấn luyện.
  - (d) Truyền tải prompt qua latent vector.

7. Text embedding  $\tau_\theta(y)$  đóng vai trò gì trong quá trình sinh ảnh?

- (a) Làm nhiễu điểu kiện để tăng đa dạng.
- (b) Sinh ảnh trực tiếp không qua khử nhiễu.
- (c) Điều kiện hoá quá trình sinh ảnh theo prompt.
- (d) Loại bỏ thông tin nhiễu trong ảnh gốc.

8. Đoạn mã sau thực hiện chức năng gì trong pipeline Stable Diffusion?

```

1 text_input = tokenizer(prompt, padding="max_length", max_length=tokenizer
                      .model_max_length, return_tensors=
                      "pt")
2 text_embeddings = text_encoder(text_input.input_ids.to(device))[0]

```

- (a) Sinh ảnh đầu ra từ latent vector.
- (b) Mã hóa văn bản thành vector đặc trưng.
- (c) Tạo nhiễu khởi tạo.
- (d) Tính toán tần số huấn luyện.

9. Trong đoạn mã dưới, vai trò của biến `latents` là gì?

```

1 latents = torch.randn(
2     (batch_size, unet.in_channels, height // 8, width // 8),
3     device=device
4 )

```

- (a) Biểu diễn ảnh RGB đầu ra.
- (b) Nhiễu khởi tạo cho quá trình sinh ảnh.
- (c) Prompt đã mã hóa.
- (d) Kết quả đầu ra cuối cùng.

10. Tại sao diffusion trong latent space giúp giảm chi phí tính toán?

- (a) Vì số bước sampling bị giảm.
- (b) Vì latent space là không gian nén, dữ liệu nhỏ hơn nhiều.
- (c) Vì VAE loại bỏ nhu cầu khử nhiễu.
- (d) Vì không cần huấn luyện lại từ đầu.

## IV. Phụ lục

1. **Datasets:** Các file dataset được đề cập trong bài có thể được tải tại [đây](#).
2. **Hint:** Các file code gợi ý có thể được tải tại [đây](#).
3. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
4. **Demo:** Web demo và mã nguồn của ứng dụng có thể được truy cập tại [đây](#).

**5. Rubric:**

Mục	Kiến Thức	Đánh Giá
I.	<ul style="list-style-type: none"> <li>- Kiến thức về crawl data.</li> <li>- Kiến thức về thư viện Selenium để crawl dữ liệu.</li> </ul>	<ul style="list-style-type: none"> <li>- Nắm được cách sử dụng thư viện Selenium để lấy dữ liệu từ một trang web.</li> </ul>
II.	<ul style="list-style-type: none"> <li>- Các kiến thức cơ bản về bài toán Image Generation.</li> <li>- Các kiến thức cơ bản về mô hình Stable Diffusion.</li> <li>- Sử dụng thư viện PyTorch, Diffusers để triển khai mô hình Stable Diffusion và thực hiện huấn luyện trên bộ dữ liệu đã thu thập được.</li> </ul>	<ul style="list-style-type: none"> <li>- Hiểu được các khái niệm cơ bản về bài toán Image Generation.</li> <li>- Hiểu được các lý thuyết cơ bản trong mô hình Stable Diffusion.</li> <li>- Có khả năng sử dụng thư viện PyTorch và Diffusers để triển khai một mô hình Stable Diffusion nhằm thực hiện huấn luyện trên dữ liệu đã thu thập.</li> </ul>

- *Hết* -