A

Project Report

On

# Text To Image Using GANs

Submitted in partial fulfillment for award of the degree of

**Bachelor of Technology**

**In**

**Computer Science and Engineering**

By

**Ankush Kamboj (2161082)**

**Sajal Prajapati (2161293)**

**Siddhartha Patwal (2161320)**

**Ayush Gupta (2161107)**

**Under the Guidance of**

**Mr. Shashi Kumar Sharma**

**ASSISTANTPROFESSOR**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**


**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**GRAPHIC ERA HILL UNIVERSITY, BHIMTAL CAMPUS**

**SATTAL ROAD, P.O. BHOWALI,**

**DISTRICT- NAINITAL-263132**

**2024-2025**

# STUDENT'S DECLARATION

**Ankush Kamboj, Sajal Prajapati, Siddhartha Patwal and Ayush Gupta** declare the work, which is being presented in the project, entitled '**Text To Image Using GANs**' in partial fulfillment of the requirement for the award of the degree **Bachelor of Technology (B.Tech.)** in the session **2024-2025**, is an authentic record of my work carried out under the supervision of **Mr. Shashi Kumar Sharma.**

The matter embodied in this project has not been submitted by me for the award of any other degree.

**Date:**

**Ankush Kamboj (2161082)**

**Sajal Prajapati (2161293)**

**Siddhartha Patwal (2161320)**

**Ayush Gupta (2161107)**

# CERTIFICATE

The project report entitled **"Text To Image Using GANs"** being submitted by **Ankush Kamboj, Siddhartha Patwal, Sajal Prajapati, Ayush Gupta** of B.Tech.(CSE)to Graphic Era Hill University Bhimtal Campus for the award of bonafide work carried out by them. She has worked under my guidance and supervision and fulfilled the requirement for the submission of a report.

| | | |
|---|---|---|
| **Mr. Shashi Kumar Sharma** | **Mr. Ayush Kapri** | **Dr. Ankur Singh Bist** |
| **(Project Guide)** | **(Project Coordinators)** | **(Head, CSE)** |

# ACKNOWLEDGEMENT

We take immense pleasure in thanking the Honorable Director **'Prof. (Col.) Anil Nair (Retd.)',** GEHU Bhimtal Campus to permit me and carry out this project work with his excellent and optimistic supervision. This has all been possible due to his novel inspiration, able guidance, and useful suggestions that helped me to develop as a creative researcher and complete the research work, in time.

Words are inadequate in offering my thanks to GOD for providing me with everything that we need. We again want to extend thanks to our president **'Prof. (Dr.) Kamal Ghanshala'** for providing us with all infrastructure and facilities to work in need without which this work could not be possible.

Many thanks to **'Dr. Ankur Singh Bist'** (Head, Department of Computer Science and Engineering, GEHU Bhimtal Campus), our project co-ordinators **'Mr. Ayush Kapri'** (Assistant Professor, Department of Computer Science and Engineering, GEHUBhimtal Campus), our project guide **Mr. Shashi Kumar Sharma, Assistant Professor, CSE Dept.** and other faculties for their insightful comments, constructive suggestions, valuable advice, and time in reviewing this thesis. Finally, yet importantly, we would like to express my heartiest thanks to my beloved parents, for their moral support, affection, and blessings. We would also like to pay my sincere thanks to all my friends and well-wishers for their help and wishes for the successful completion of this research.

**Ankush Kamboj (2161082)**

**Sajal Prajapati (2161293)**

**Siddhartha Patwal (2161320)**

**Ayush Gupta (2161107)**

# ABSTRACT

**The field of artificial intelligence has witnessed remarkable advancements in recent years, especially in the domain of deep learning and generative modeling. One of the most fascinating applications of these advancements is the generation of images from textual descriptions, a task that combines the power of Natural Language Processing (NLP) with Computer Vision. This project, titled *"Text to Image using GANs"*, explores the use of Generative Adversarial Networks (GANs) to synthesize realistic images directly from human-written textual prompts.**

**In this project, a multi-stage GAN architecture is employed to progressively generate high-quality images from input text. The text descriptions are first converted into semantic embeddings using a pre-trained BERT (Bidirectional Encoder Representations from Transformers) model. These embeddings are then fed into a generator that attempts to create an image that aligns with the semantic meaning of the text. Simultaneously, a discriminator network evaluates the authenticity of the generated image and its compatibility with the given text. This adversarial training enables the model to improve the image quality over time.**

**We utilize popular GAN architectures like StackGAN and AttnGAN, and the system is trained and tested on the Flickr Image dataset, which provides diverse image-caption pairs. The quality of the generated images is evaluated using metrics such as Fréchet Inception Distance (FID) and Inception Score (IS), alongside human visual inspection.**

**The ability to convert natural language into images has numerous real-world applications in art generation, advertising, virtual reality content creation, accessibility tools, and gaming. This project demonstrates a promising step toward bridging the gap between human language and visual understanding through AI. While there are limitations such as training time and resource requirements, the results highlight the potential of GAN-based models in creative AI applications.**

**TABLE OF CONTENTS**

**LIST OF ABBREVIATIONS**

**ML:** Machine Learning

**CNN:** Convolutional Neural Network

**3D:** Three Dimensional

**GPU:** Graphics Processing Unit

**RGB:** Red, Green, Blue

**GEHU:** Graphic Era Hill University

**GAN:** Generative Adversarial Network

**BERT:** Bidirectional Encoder Representations from Transformers

**FID:** Frechet Inception Distance

**IS:** Inception Score

**RNN:** Recurrent Neural Network

**List of Figures**

# CHAPTER 1:

# INTRODUCTION

## 1.1 Prologue

In the ever-evolving world of artificial intelligence, the capabilities of machines are expanding rapidly. One of the most visually striking achievements of AI is its ability to generate art, specifically images, from human-written text. This fusion of natural language processing (NLP) and computer vision has given rise to an exciting domain known as Text-to-Image generation. This technique enables machines to interpret a textual description and translate it into a corresponding image, capturing the semantic intent of the language.

The inspiration for this technology comes from the human brain's ability to visualize a story, concept, or idea simply by reading or listening. For machines to replicate this behavior, a robust understanding of both language and image synthesis is essential. The process demands a combination of text encoders capable of comprehending complex semantic information and image generators powerful enough to synthesize high-quality visuals.

Generative Adversarial Networks (GANs) have revolutionized image generation by enabling machines to learn through adversarial training, improving the quality and realism of generated images over time. When GANs are integrated with advanced NLP models like BERT, they offer a potent solution for translating descriptigeneave text into realistic visuals.

This chapter provides an overview of the text-to-image generation task, setting the stage for the research and implementation covered in the following chapters. The aim is to highlight the key motivations, background, problem statement, and objectives that define the scope of this project. The chapter also outlines the methodology used to develop and evaluate the proposed system. As AI continues to impact creative domains, the ability of machines to understand and visualize human language stands as a significant milestone, pushing the boundaries of what intelligent systems can achieve in the realm of visual storytelling.

## 1.2 Background and Motivation

In recent years, the advancement of artificial intelligence (AI) and deep learning has enabled machines to generate highly realistic content, including images, videos, and audio. Among these, Text to Image Generation using Generative Adversarial Networks (GANs) has emerged as a groundbreaking area of research and application. This technology allows machines to create images purely based on textual descriptions, opening possibilities in art, design, entertainment, accessibility, and more.

The core motivation behind this project lies in the increasing demand for intelligent systems capable of understanding human language and translating it into meaningful visual representations. While traditional image generation required manual design or extensive datasets, GANs offer a scalable and automated solution by learning patterns and distributions directly from data.

In the digital era where content is king, the ability to automatically generate visual content from textual data has the potential to revolutionize industries such as:

- Content Creation: Automating illustration, graphic design, and animation generation based on scripts or storylines.

- Accessibility: Helping visually impaired individuals understand textual content through visual output.
- E-commerce: Generating product previews from user queries or text descriptions.
- Gaming and Virtual Worlds: Dynamically creating scenes, assets, or characters from player input.

### 1.3 Problem Statement

In an age of rapidly evolving artificial intelligence, one of the fundamental challenges in machine learning and computer vision is enabling machines to understand and generate visual content from human language. **Text to Image Generation** is a complex problem that demands a deep integration of natural language understanding and generative image modeling.

While significant progress has been made in generative models like GANs (Generative Adversarial Networks), generating high-quality, semantically accurate images from arbitrary text descriptions remains an unsolved problem. The core difficulty lies in the multimodal nature of the task—text is sequential and abstract, whereas images are spatial and pixel-based. Capturing this cross-domain alignment requires sophisticated models that can bridge language semantics with visual features.

Despite recent advances, current models often suffer from:

- **Low-resolution or blurry outputs** that lack fine detail.
- **Semantic mismatches**, where the generated image does not faithfully represent the textual input.
- **Training instability** in GANs due to mode collapse or vanishing gradients.
- **Limited generalization** across varied and complex textual descriptions.

Furthermore, there is a lack of **interpretable, scalable, and modular frameworks** that can be easily adapted to different datasets and domains. Many existing systems are tailored to specific datasets like birds or flowers and struggle when extended to open-domain text or high-resolution imagery.

This project addresses these challenges by implementing a robust text-to-image generation pipeline using **pretrained language models like BERT** for semantic text encoding and **advanced GAN architectures** such as **StackGAN, AttnGAN, and DM-GAN** for high-quality image synthesis. The proposed solution aims to generate realistic images that not only visually resemble the description but also maintain structural consistency and semantic alignment.

Given my background in AI/ML, deep learning, and full-stack development—demonstrated through projects like a Banking Operations System in C++, a Job Listing Portal using React, and machine learning with Python—this project serves as a practical application of my skills and a contribution to solving one of the most visually and linguistically challenging problems in AI.

The ultimate goal is to design and build a **scalable, modular, and accurate text-to-image generation system** that pushes the boundaries of generative AI, contributes to research, and

opens doors to practical applications in content creation, accessibility, virtual reality, and beyond.

### 1.4 Objective

1. Bridge the Gap Between Natural Language and Visual Representation

The primary objective is to develop a system that can generate high-quality, realistic images directly from natural language descriptions. This helps bridge the semantic gap between textual information and visual content using Generative Adversarial Networks (GANs).

2. Leverage Pretrained Text Embeddings

Utilize powerful pretrained models like BERT or CLIP for converting descriptive sentences into meaningful vector embeddings that can be effectively used as input to the image generation pipeline, ensuring semantic alignment between text and generated image.

3. Implement Advanced GAN Architectures

Integrate state-of-the-art GAN models such as AttnGAN, StackGAN++, and DM-GAN, which are specially designed for text-to-image generation. These models enable multi-stage refinement, attention mechanisms, and high-resolution outputs.

4. Create a Robust Data Handling Pipeline

Incorporate real-world datasets such as Flickr Image or CUB with structured text-image pairs. Build efficient preprocessing pipelines to clean, tokenize, and encode captions while resizing and normalizing images for optimal training performance.

5. Enable High-Resolution Image Synthesis

Focus on generating high-resolution (e.g., 256×256 or above) and visually consistent images that accurately reflect the fine-grained details described in the text input.

6. Facilitate End-to-End Training and Evaluation

Develop and train the model in a modular pipeline using PyTorch. Evaluate the quality of generated images using metrics such as Inception Score (IS) and Fréchet Inception Distance (FID) to assess realism and diversity.

7. Integrate a User-Friendly Interface

Design a simple front-end or command-line interface where users can input text descriptions and instantly view the generated images, making the system interactive and accessible for demonstrations or further applications.

8. Explore Real-World Applications

Demonstrate the potential applications of this system in industries such as design prototyping, gaming, advertising, and assistive technologies for visually impaired individuals, thereby showcasing its societal and commercial value.

### 1.5 Scope of the Project

The scope of this project encompasses the research, design, development, and evaluation of a deep learning-based system that can generate images from natural language descriptions using

Generative Adversarial Networks (GANs). The project aims to explore state-of-the-art architectures and methodologies that can accurately translate textual data into high-resolution and semantically meaningful images. This project primarily focuses on combining natural language processing (NLP) with computer vision to deliver innovative solutions in AI-driven realcreativity and content generation.

**Key Areas Covered:**

1. **Text Processing and Embedding**
   The project will handle the preprocessing of natural language inputs and transform them into numerical representations using advanced embedding models such as BERT or CLIP. These embeddings will serve as the condition vectors for the GAN.

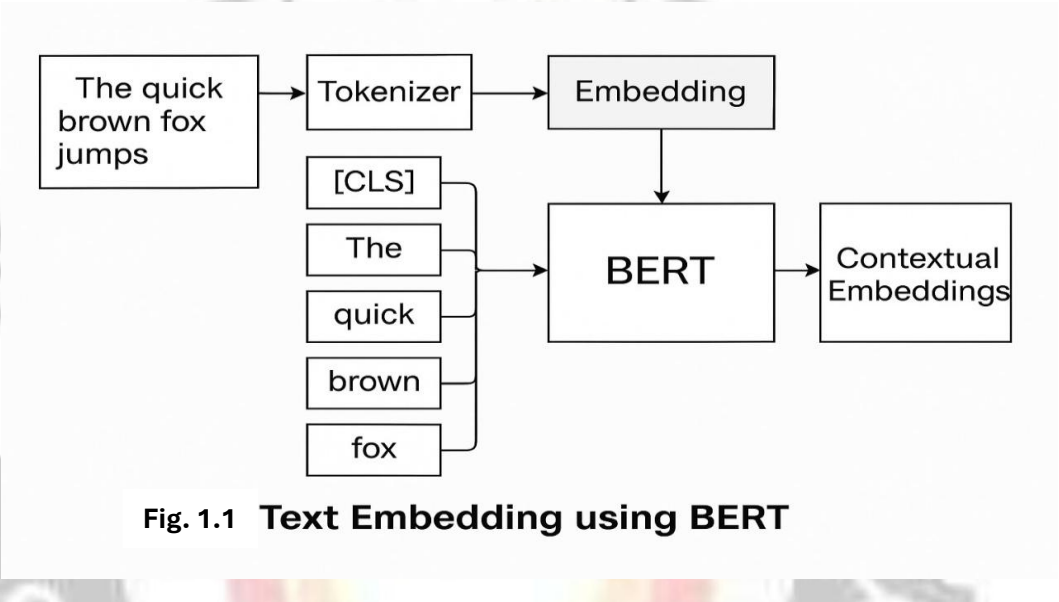



**Fig. 1.1** **Text Embedding using BERT**

2. **GAN Architecture Implementation**
   Multiple advanced GAN architectures such as StackGAN++, AttnGAN, and DM-GAN will be implemented, trained, and compared to identify the best-performing model for text-to-image generation tasks.

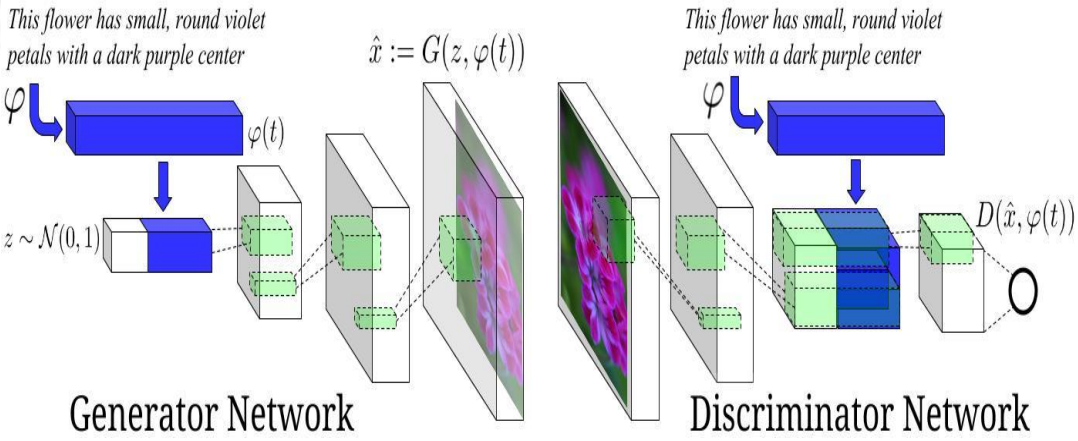



**Fig. 2.2**

3. **Dataset Integration**
   Publicly available datasets like Flickr Image or CUB will be used to train the models. The scope includes data cleaning, augmentation, and alignment of image-caption pairs for supervised learning.

4. **Model Training and Optimization**
   The project will train models using PyTorch, optimize hyperparameters, and monitor convergence using appropriate loss functions and training strategies.
5. **Evaluation Metrics**
   Generated images will be evaluated using standard quantitative metrics such as Inception Score (IS) and Fréchet Inception Distance (FID) along with qualitative visual assessments.
6. **User Interface (Optional)**
   A simple interface may be developed to allow users to input text and view generated images, demonstrating the system's capabilities interactively.
7. **Application Exploration**
   The project will also explore potential applications in domains such as creative design, advertising, game development, and educational tools.
8. **Future Enhancements**
   The scope includes identifying future improvements such as multimodal input support, higher resolution generation, or deploying the model as a web service.

9. **Workflow Diagram**

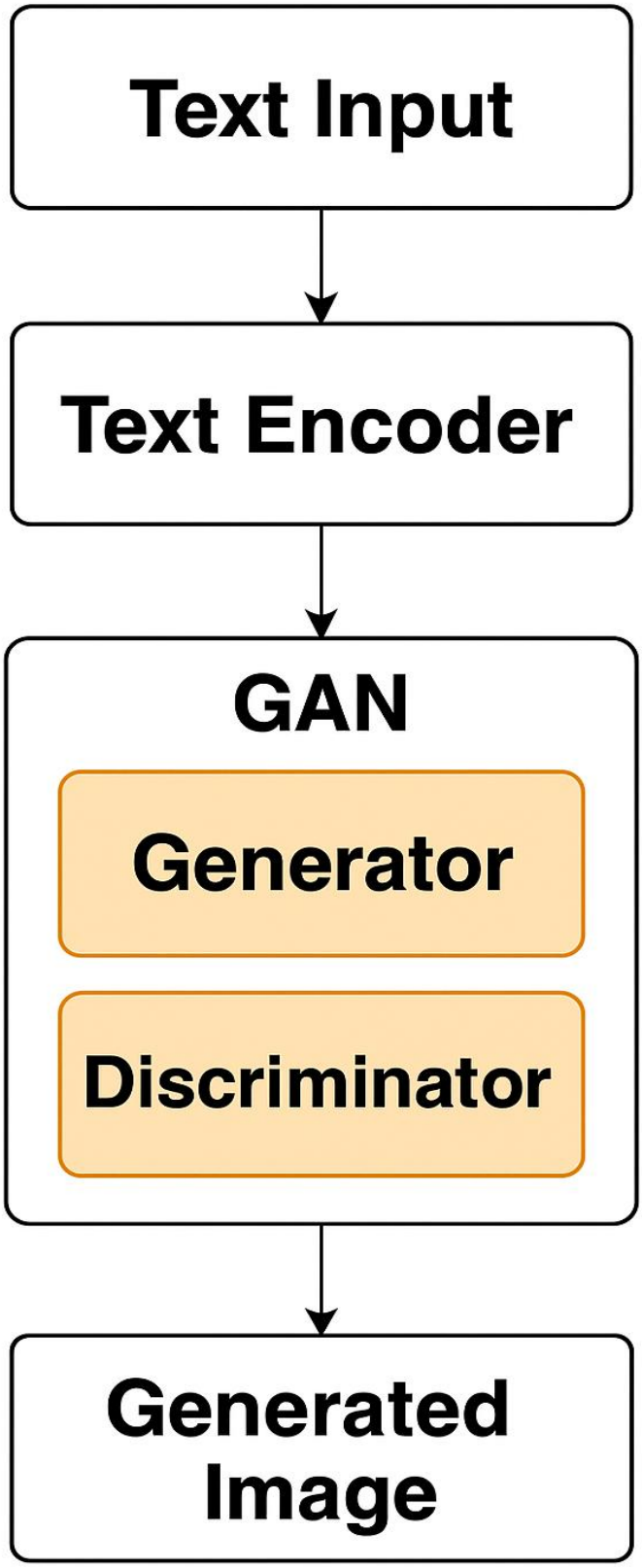



**Fig. 3.3**

# CHAPTER 2 :

# PHASES OF SOFTWARE DEVELOPMENT CYCLE

## 2.1 SDLC Model Used

For the development of the “Text to Image using GANs” project, the Iterative Waterfall Model of the Software Development Life Cycle (SDLC) was used. This model allows the development process to proceed in a structured and phased manner, with the flexibility to revisit previous phases if needed.

**Why Iterative Waterfall Model?**

The Iterative Waterfall Model combines the linear flow of the classical Waterfall Model with the feedback and refinement capabilities of iterative development. It is suitable for machine learning and deep learning projects, where continuous tuning, evaluation, and adjustments are necessary after each phase.

---

**Phases Followed:**

1. **Requirement Analysis**
   The functional and non-functional requirements of the system were gathered. It included understanding the problem of image generation from text descriptions, identifying datasets, and defining model goals such as image clarity, text relevance, and computational efficiency.

2. **System Design**
   The architecture of the system was planned, including module division into text preprocessing (BERT), GAN training, image generation, and result visualization. Dataset structure (Flickr Image) and model type (StackGAN, AttnGAN) were selected.

3. **Implementation**
   The project was implemented in Python using PyTorch. Preprocessing scripts, model training scripts, and inference modules were coded and iteratively tested. BERT was integrated for textual feature encoding, and GAN architectures were used for image synthesis.

4. **Testing**
   Each module was tested individually and then integrated for end-to-end testing. Accuracy, text-image relevance, FID score (Fréchet Inception Distance), and performance on unseen captions were evaluated.

5. **Deployment**
   The trained models and interface were made ready for deployment. Images were generated based on user input texts and showcased through a simple GUI or Jupyter interface.

6. **Maintenance**
   The system allows for continuous improvement by fine-tuning the model, updating datasets, or switching to more advanced GAN architectures like DM-GAN or StyleGAN.

### 2.2 Hardware Requirements

To develop and run the "Text to Image using GANs" project efficiently, the following hardware components are recommended:

- **Processor**: Intel i7 / AMD Ryzen 7 or higher (multi-core preferred for training deep learning models)
- **RAM**: Minimum 16 GB (32 GB recommended for faster data handling)
- **Storage**: SSD with at least 512 GB (preferably 1 TB for storing datasets and models)
- **GPU**: NVIDIA GPU with CUDA support (e.g., RTX 3060/3080 or higher) – essential for training GANs
- **Display**: Full HD monitor for better visualization of results
- **Others**: Reliable power supply, external cooling system (optional but useful during model training)

### 2.3 Software Requirements

The following software and libraries are used in the development and execution of this project:

- **Operating System**: Ubuntu 20.04 LTS / Windows 10/11 (64-bit)
- **Programming Language**: Python 3.8+
- **IDE/Editor**: Visual Studio Code / Jupyter Notebook / PyCharm
- **Deep Learning Frameworks**:
    - PyTorch (v1.13 or above)
    - TensorFlow (optional, based on model used)
- **Libraries & Packages**:
    - transformers (for BERT embeddings)
    - numpy, matplotlib, opencv-python, Pillow
    - nltk, scikit-learn, seaborn (for preprocessing and visualization)
    - tqdm, torchvision, sentence-transformers
- **Dataset**: Flickr Image  / CUB-200-2011 (prepared in COCO-style format)
- **Environment Management**:
    - Conda / Virtualenv
- **Additional Tools**:
    - CUDA Toolkit and cuDNN (GPU acceleration)

## CHAPTER 3:

## CODING OF FUNCTIONS

```
#discriminator.py

import torch
import torch.nn as nn


class Discriminator(nn.Module):
    def __init__(self, image_shape, vocab_size, embedding_dim,
caption_encoder_hidden_size):
        super(Discriminator, self).__init__()

        self.image_shape = image_shape
        self.embedding_dim = embedding_dim
        self.caption_encoder_hidden_size = caption_encoder_hidden_size

        self.conv1 = nn.Conv2d(image_shape[0], 64, kernel_size=4,
stride=2, padding=1)
        self.leaky_relu1 = nn.LeakyReLU(0.2)
        self.dropout1 = nn.Dropout(0.3)

        self.conv2 = nn.Conv2d(64, 128, kernel_size=4, stride=2,
padding=1)
        self.batch_norm1 = nn.BatchNorm2d(128)
        self.leaky_relu2 = nn.LeakyReLU(0.2)
        self.dropout2 = nn.Dropout(0.3)

        self.flattened_image_features = 128 * (image_shape[1] // 4) *
(image_shape[2] // 4)

        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.text_encoder = nn.Sequential(
            nn.Linear(embedding_dim, self.caption_encoder_hidden_size),
            nn.LeakyReLU(0.2)
        )

        self.combined_dense = nn.Linear(self.flattened_image_features +
self.caption_encoder_hidden_size, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, image, captions_indexed):
        x = self.conv1(image)
        x = self.leaky_relu1(x)
        x = self.dropout1(x)

        x = self.conv2(x)
```

```
        x = self.batch_norm1(x)
        x = self.leaky_relu2(x)
        x = self.dropout2(x)

        x = torch.flatten(x, 1)

        embedded_captions = self.embedding(captions_indexed)
        y = embedded_captions.mean(dim=1)
        y = self.text_encoder(y)

        combined = torch.cat([x, y], dim=1)

        z = self.combined_dense(combined)
        z = self.sigmoid(z)

        return z
```

```
#generator.py
import torch
import torch.nn as nn


class Generator(nn.Module):
    def __init__(self, noise_dim, vocab_size, embedding_dim,
caption_encoder_hidden_size, output_channels=3):
        super(Generator, self).__init__()

        self.noise_dim = noise_dim
        self.embedding_dim = embedding_dim
        self.caption_encoder_hidden_size = caption_encoder_hidden_size
        self.output_channels = output_channels

        self.embedding = nn.Embedding(vocab_size, embedding_dim)

        self.text_encoder = nn.Sequential(
            nn.Linear(embedding_dim, caption_encoder_hidden_size),
            nn.LeakyReLU(0.2, inplace=True)
        )

        self.total_input_dim = self.noise_dim +
self.caption_encoder_hidden_size

        self.model = nn.Sequential(
            nn.Linear(self.total_input_dim, 256 * 4 * 4, bias=False),
```

```
            nn.BatchNorm1d(256 * 4 * 4),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Unflatten(1, (256, 4, 4)),
            nn.ConvTranspose2d(256, 128, kernel_size=5, stride=2,
padding=2, output_padding=1, bias=False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),
            nn.ConvTranspose2d(128, 64, kernel_size=5, stride=2,
padding=2, output_padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.2, inplace=True),
            nn.ConvTranspose2d(64, 32, kernel_size=5, stride=2, padding=2,
output_padding=1, bias=False),
            nn.BatchNorm2d(32),
            nn.LeakyReLU(0.2, inplace=True),
            nn.ConvTranspose2d(32, self.output_channels, kernel_size=5,
stride=2, padding=2, output_padding=1, bias=False),
            nn.Tanh()
        )

    def forward(self, z, captions_indexed):
        embedded_captions = self.embedding(captions_indexed)
        caption_features = embedded_captions.mean(dim=1)
        caption_features = self.text_encoder(caption_features)

        combined_input = torch.cat([z, caption_features], dim=1)

        return self.model(combined_input)
```

```
#train.py

import os
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim

from models.discriminator import Discriminator
from models.generator import Generator
from utils.data_loader import DataLoader

latent_dim = 100
embedding_dim = 256
caption_encoder_hidden_size = 256
image_shape = (3, 64, 64)
```

```
batch_size = 32
epochs = 700
max_caption_length = 50

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

print("Attempting to initialize DataLoader...")
data_loader = DataLoader(
    'data/images',
    'data/captions/captions.txt',
    image_size=(64, 64),
    min_word_freq=5
)
print("DataLoader initialized successfully!")

print(torch.Size([1, 1]))
print(torch.Size([2, 3, 64, 64]))

generator = Generator(
    noise_dim=latent_dim,
    vocab_size=data_loader.vocab_size,
    embedding_dim=embedding_dim,
    caption_encoder_hidden_size=caption_encoder_hidden_size,
    output_channels=image_shape[0]
).to(device)

discriminator = Discriminator(
    image_shape=image_shape,
    vocab_size=data_loader.vocab_size,
    embedding_dim=embedding_dim,
    caption_encoder_hidden_size=caption_encoder_hidden_size
).to(device)

adversarial_loss = nn.BCELoss().to(device)

optimizer_G = optim.Adam(generator.parameters(), lr=0.0002, betas=(0.5,
0.999))
optimizer_D = optim.Adam(discriminator.parameters(), lr=0.0002,
betas=(0.5, 0.999))

num_batches = len(data_loader.captions) // batch_size
if num_batches == 0:
    print("Warning: Not enough data to form a single batch. Check your
dataset size and batch_size.")
    exit()

print(f"Total batches per epoch: {num_batches}")
```

```
for epoch in range(epochs):
    for i in range(num_batches):
        batch_images_np, batch_encoded_captions_np =
data_loader.get_batch(batch_size, max_caption_length)

        if batch_images_np.size == 0 or batch_encoded_captions_np.size ==
0:
            print(f"Skipping batch {i} in Epoch {epoch}: No data returned
from data_loader.get_batch.")
            continue

        real_imgs = torch.FloatTensor(batch_images_np).permute(0, 3, 1,
2).to(device)
        batch_captions_tensor =
torch.LongTensor(batch_encoded_captions_np).to(device)

        valid = torch.ones(batch_size, 1).to(device)
        fake = torch.zeros(batch_size, 1).to(device)

        optimizer_G.zero_grad()

        z = torch.randn(batch_size, latent_dim).to(device)
        gen_imgs = generator(z, batch_captions_tensor)
        validity = discriminator(gen_imgs, batch_captions_tensor)
        g_loss = adversarial_loss(validity, valid)

        g_loss.backward()
        optimizer_G.step()

        optimizer_D.zero_grad()
        real_loss = adversarial_loss(discriminator(real_imgs,
batch_captions_tensor), valid)
        fake_loss = adversarial_loss(discriminator(gen_imgs.detach(),
batch_captions_tensor), fake)
        d_loss = (real_loss + fake_loss) / 2

        d_loss.backward()
        optimizer_D.step()

        if i % 100 == 0:
            print(f"Epoch [{epoch}/{epochs}], Batch [{i}/{num_batches}], D
Loss: {d_loss.item():.4f}, G Loss: {g_loss.item():.4f}")

        if (epoch + 1) % 10 == 0:
            os.makedirs('saved_models', exist_ok=True)
            torch.save(generator.state_dict(),
f'saved_models/generator_epoch_{epoch+1}.pth')
```

```
            print(f"Saved generator model checkpoint at epoch {epoch+1}")

os.makedirs('saved_models', exist_ok=True)
torch.save(generator.state_dict(), 'saved_models/generator_final.pth')
print("Saved final generator model to saved_models/generator_final.pth")
print("Training loop completed.")
```

```
#index.html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>GAN + Stable Diffusion Image Editor</title>
    <style>
        @import
url('https://fonts.googleapis.com/css2?family=Inter:wght@300;400;500;600;7
00&display=swap');

        body {
            font-family: 'Inter', sans-serif;
            margin: 0;
            padding: 20px;
            background-color: #eef2f6; /* Lighter background */
            display: flex;
            justify-content: center;
            align-items: flex-start;
            min-height: 100vh;
            color: #333;
        }
        #app {
            background-color: #ffffff;
            border-radius: 16px; /* More rounded corners */
            box-shadow: 0 8px 30px rgba(0, 0, 0, 0.1);
            padding: 40px;
            width: 100%;
            max-width: 800px; /* Wider for tools */
            text-align: center;
            box-sizing: border-box;
            display: flex;
            flex-direction: column;
            gap: 30px; /* Space between sections */
        }
        h1 {
            color: #2c3e50; /* Darker heading */
            margin-bottom: 25px;
```

```
            font-size: 2.5em;
            font-weight: 600;
        }
        .input-section, .image-section {
            padding: 20px;
            border-radius: 12px;
            background-color: #f8fafd; /* Light grey background for
sections */
            box-shadow: inset 0 1px 3px rgba(0,0,0,0.05); /* Subtle inner
shadow */
        }
        .input-group {
            display: flex;
            flex-direction: column;
            align-items: center;
            gap: 15px;
        }
        input[type="text"] {
            width: calc(100% - 24px); /* Account for padding */
            padding: 14px;
            border: 1px solid #c9d8e6; /* Lighter border */
            border-radius: 10px; /* More rounded */
            font-size: 1.1em;
            box-sizing: border-box;
            transition: border-color 0.3s ease, box-shadow 0.3s ease;
        }
        input[type="text"]:focus {
            border-color: #007bff;
            box-shadow: 0 0 0 3px rgba(0, 123, 255, 0.25);
            outline: none;
        }
        .main-button {
            background-color: #007bff;
            color: white;
            padding: 14px 30px;
            border: none;
            border-radius: 10px;
            font-size: 1.2em;
            cursor: pointer;
            transition: background-color 0.3s ease, transform 0.2s ease,
box-shadow 0.3s ease;
            box-shadow: 0 5px 15px rgba(0, 123, 255, 0.3);
            font-weight: 500;
        }
        .main-button:hover {
            background-color: #0056b3;
            transform: translateY(-3px);
            box-shadow: 0 8px 20px rgba(0, 123, 255, 0.4);
```

```
        }
        .main-button:disabled {
            background-color: #a0caff;
            cursor: not-allowed;
            transform: none;
            box-shadow: none;
        }

        #statusMessage {
            margin-top: 20px;
            font-size: 1.1em;
            color: #555;
            min-height: 25px;
            font-weight: 400;
        }
        #estimatedTime {
            font-size: 0.9em;
            color: #777;
            margin-top: 5px;
            min-height: 20px; /* Reserve space */
        }
        #progressBarContainer {
            width: 100%;
            background-color: #e0e0e0;
            border-radius: 5px;
            margin-top: 10px;
            height: 20px;
            overflow: hidden;
            display: none; /* Hidden by default */
        }
        #progressBar {
            height: 100%;
            width: 0%;
            background-color: #28a745; /* Green progress bar */
            border-radius: 5px;
            text-align: center;
            color: white;
            line-height: 20px;
            font-size: 0.9em;
            transition: width 0.3s ease; /* Smooth transition for progress
*/
        }
        #image-display-area {
            position: relative;
            display: flex;
            justify-content: center;
            align-items: center;
            min-height: 350px; /* Placeholder height for image */
```

```
            background-color: #f0f0f0; /* Light grey background */
            border-radius: 10px;
            overflow: hidden; /* Important for canvas */
            box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);
            border: 1px solid #e0e0e0;
        }
        canvas {
            display: none; /* Hidden by default */
            max-width: 100%;
            height: auto;
            border-radius: 8px; /* Slightly less than container for visual
separation */
        }

        .editing-tools {
            margin-top: 20px;
            display: flex;
            flex-wrap: wrap; /* Allows buttons to wrap */
            justify-content: center;
            gap: 10px; /* Space between tool buttons */
            padding-top: 20px;
            border-top: 1px solid #e0e0e0;
            display: none; /* Hidden until image is loaded */
        }
        .editing-tools button {
            background-color: #6c757d; /* Grey for tools */
            color: white;
            padding: 10px 18px;
            border: none;
            border-radius: 8px;
            font-size: 0.95em;
            cursor: pointer;
            transition: background-color 0.2s ease, transform 0.1s ease;
            box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
        }
        .editing-tools button:hover {
            background-color: #5a6268;
            transform: translateY(-1px);
        }
        .editing-tools button:active {
            transform: translateY(0);
        }

        /* Responsive adjustments */
        @media (max-width: 768px) {
            body {
                padding: 10px;
            }
```

```
            #app {
                padding: 20px;
                margin: 10px;
            }
            input[type="text"] {
                width: 100%;
            }
            h1 {
                font-size: 2em;
            }
            .main-button {
                padding: 12px 20px;
                font-size: 1.1em;
            }
            .editing-tools {
                gap: 8px;
            }
            .editing-tools button {
                padding: 8px 15px;
                font-size: 0.9em;
            }
        }
    </style>
</head>
<body>
    <div id="app">
        <h1>GAN-to-SD Image Creator & Editor</h1>

        <div class="input-section">
            <div class="input-group">
                <input type="text" id="captionInput" placeholder="Enter a
descriptive caption (e.g., 'A cat wearing a wizard hat in a library')">
                <button class="main-button" id="generateButton"
onclick="generateAndRefineImage()">Generate & Refine Image</button>
            </div>
            <p id="statusMessage">Enter a caption and click 'Generate &
Refine' to start.</p>
            <p id="estimatedTime"></p> <div id="progressBarContainer">
                <div id="progressBar">0%</div>
            </div>
        </div>

        <div class="image-section">
            <h2>Result:</h2>
            <div id="image-display-area">
                <canvas id="imageCanvas"></canvas>
            </div>
            <div class="editing-tools" id="editingTools">
```

```
                <button onclick="rotateImage()">Rotate 90°</button>
                <button onclick="flipImage('horizontal')">Flip
Horizontal</button>
                <button onclick="flipImage('vertical')">Flip
Vertical</button>
                <button onclick="applyGrayscale()">Grayscale</button>
                <button class="main-button"
onclick="downloadImage()">Download Edited Image</button>
            </div>
        </div>
    </div>

    <script>
        const captionInput = document.getElementById('captionInput');
        const generateButton = document.getElementById('generateButton');
        const statusMessage = document.getElementById('statusMessage');
        const estimatedTimeDisplay =
document.getElementById('estimatedTime'); // Get new element
        const progressBarContainer =
document.getElementById('progressBarContainer');
        const progressBar = document.getElementById('progressBar');
        const imageCanvas = document.getElementById('imageCanvas');
        const ctx = imageCanvas.getContext('2d');
        const editingTools = document.getElementById('editingTools');

        let originalImage = new Image();
        let currentImage = new Image();
        let eventSource = null; // To hold the SSE connection

        // Function to load the image onto the canvas for editing
        function loadImageToCanvas(imgSrc) {
            originalImage = new Image();
            originalImage.onload = () => {
                imageCanvas.width = originalImage.width;
                imageCanvas.height = originalImage.height;
                currentImage.src = originalImage.src;
                currentImage.onload = () => {
                    ctx.clearRect(0, 0, imageCanvas.width,
imageCanvas.height);
                    ctx.drawImage(currentImage, 0, 0);
                    imageCanvas.style.display = 'block';
                    editingTools.style.display = 'flex';
                };
            };
            originalImage.src = imgSrc;
        }

        // Main generation and refinement function
```

```
        async function generateAndRefineImage() {
            const caption = captionInput.value.trim();
            if (!caption) {
                alert('Please enter a caption!');
                return;
            }

            // Reset UI for new generation
            generateButton.disabled = true;
            statusMessage.textContent = 'Starting image generation...';
            estimatedTimeDisplay.textContent = ''; // Clear previous
estimate
            progressBarContainer.style.display = 'block';
            progressBar.style.width = '0%';
            progressBar.textContent = '0%';
            imageCanvas.style.display = 'none';
            editingTools.style.display = 'none';
            ctx.clearRect(0, 0, imageCanvas.width, imageCanvas.height);

            // Close any existing SSE connection
            if (eventSource) {
                eventSource.close();
                eventSource = null;
            }

            try {
                // Step 1: Request the backend to start the generation
process
                const response = await fetch('/generate_and_refine_image',
{
                    method: 'POST',
                    headers: { 'Content-Type': 'application/json' },
                    body: JSON.stringify({ caption: caption })
                });

                const data = await response.json();

                if (!response.ok) {
                    throw new Error(data.error || 'Failed to start
generation process');
                }

                const job_id = data.job_id;
                const estimated_time = data.estimated_time; // Get
estimated time

                if (estimated_time !== null) {
```

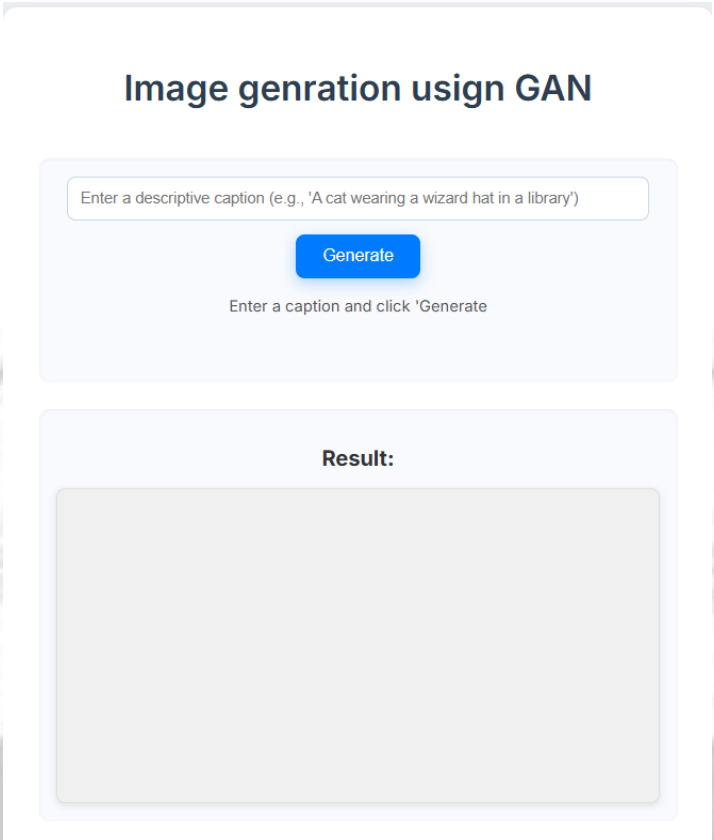
```
                    estimatedTimeDisplay.textContent = `Estimated time:
${estimated_time} seconds (based on recent generations)`;
                } else {
                    estimatedTimeDisplay.textContent = `No historical data
for estimation. Please wait...`;
                }
                statusMessage.textContent = `Generation started (Job ID:
${job_id}). Waiting for progress updates...`;


                // Step 2: Open an SSE connection to receive progress
updates
                eventSource = new EventSource(`/progress/${job_id}`);

                eventSource.onmessage = function(event) {
                    const progressData = JSON.parse(event.data);
                    statusMessage.textContent = `${progressData.status}`;

                    if (progressData.progress !== undefined &&
progressData.progress >= 0) {
                        progressBar.style.width =
`${progressData.progress}%`;
                        progressBar.textContent =
`${progressData.progress}%`;
                    }

                    if (progressData.done) {
                        eventSource.close();
                        eventSource = null; // Clear the EventSource
object
                        progressBarContainer.style.display = 'none'; //
Hide progress bar
                        estimatedTimeDisplay.textContent = ''; // Clear
estimated time once done

                        if (progressData.image_url) {
                            statusMessage.textContent = `Image generated
and refined in ${progressData.time_taken} seconds! You can now edit it
below.`;
                            loadImageToCanvas(progressData.image_url);
                        } else if
(progressData.status.startsWith("Error")) {
                            statusMessage.textContent =
`${progressData.status}`;
                        }
                    }
                };

                eventSource.onerror = function(err) {
```

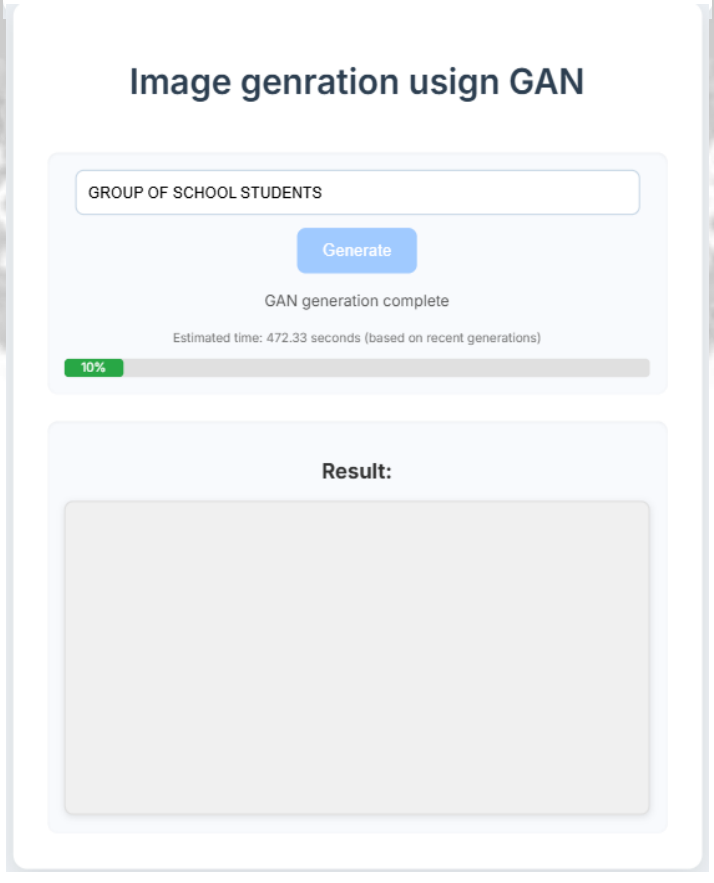
```
                    console.error('EventSource failed:', err);
                    if (eventSource) { // Ensure eventSource exists before
closing
                        eventSource.close();
                        eventSource = null;
                    }
                    statusMessage.textContent = 'Error receiving updates.
Please check server logs.';
                    estimatedTimeDisplay.textContent = ''; // Clear
estimated time on error
                    progressBarContainer.style.display = 'none';
                    generateButton.disabled = false;
                };

            } catch (error) {
                console.error('Initial generation request failed:',
error);
                statusMessage.textContent = `Error: ${error.message ||
'Failed to start generation request'}`;
                estimatedTimeDisplay.textContent = ''; // Clear estimated
time on error
                progressBarContainer.style.display = 'none';
                generateButton.disabled = false;
            }
        }

        // --- Image Editing Functions ---
        // (No changes here, they remain the same as previous version)

        function redrawImage(imageObj) {
            ctx.clearRect(0, 0, imageCanvas.width, imageCanvas.height);
            ctx.drawImage(imageObj, 0, 0, imageCanvas.width,
imageCanvas.height);
        }

        function rotateImage() {
            const tempWidth = imageCanvas.width;
            imageCanvas.width = imageCanvas.height;
            imageCanvas.height = tempWidth;

            ctx.clearRect(0, 0, imageCanvas.width, imageCanvas.height);
            ctx.save();
            ctx.translate(imageCanvas.width / 2, imageCanvas.height / 2);
            ctx.rotate(90 * Math.PI / 180);
            ctx.drawImage(currentImage, -currentImage.width / 2, -
currentImage.height / 2, currentImage.width, currentImage.height);
            ctx.restore();
            currentImage.src = imageCanvas.toDataURL();
```

```
        }

        function flipImage(direction) {
            ctx.clearRect(0, 0, imageCanvas.width, imageCanvas.height);
            ctx.save();
            if (direction === 'horizontal') {
                ctx.translate(imageCanvas.width, 0);
                ctx.scale(-1, 1);
            } else if (direction === 'vertical') {
                ctx.translate(0, imageCanvas.height);
                ctx.scale(1, -1);
            }
            ctx.drawImage(currentImage, 0, 0, imageCanvas.width,
imageCanvas.height);
            ctx.restore();
            currentImage.src = imageCanvas.toDataURL();
        }

        function applyGrayscale() {
            const imageData = ctx.getImageData(0, 0, imageCanvas.width,
imageCanvas.height);
            const pixels = imageData.data;
            for (let i = 0; i < pixels.length; i += 4) {
                const lightness = (pixels[i] + pixels[i + 1] + pixels[i +
2]) / 3;
                pixels[i] = lightness;
                pixels[i + 1] = lightness;
                pixels[i + 2] = lightness;
            }
            ctx.putImageData(imageData, 0, 0);
            currentImage.src = imageCanvas.toDataURL();
        }

        function downloadImage() {
            const link = document.createElement('a');
            link.download = 'refined_edited_image.png';
            link.href = imageCanvas.toDataURL('image/png');
            link.click();
        }
    </script>
</body>
</html>
```

# CHAPTOR 4 :

# SNAPSHOTS



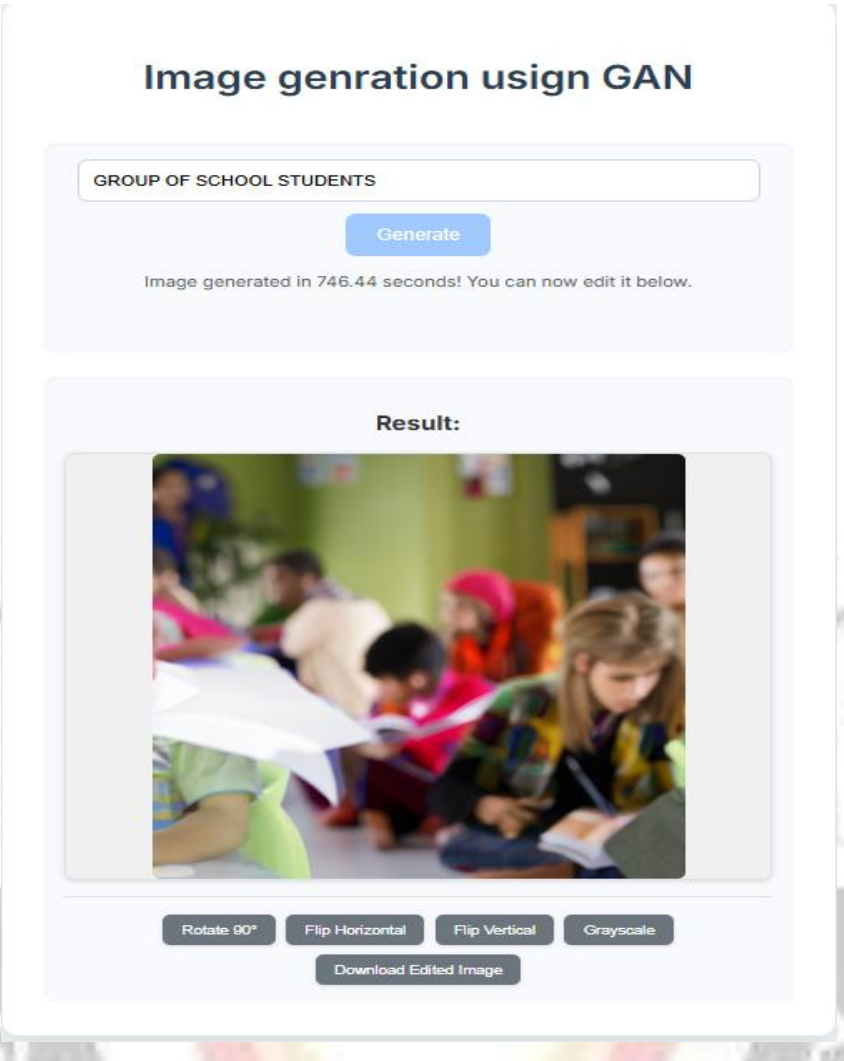


Image genration usign GAN
GROUP OF SCHOOL STUDENTS
Generate
Image generated in 746.44 seconds! You can now edit it below.
Result:
Rotate 90°
Flip Horizontal
Flip Vertical
Grayscale
Download Edited Image

# CHAPTER 5:

# TESTING STRATEGIES

Testing plays a crucial role in the development lifecycle of AI/ML-based systems, particularly those involving complex architectures like Generative Adversarial Networks (GANs). In this project, various testing strategies were employed to ensure the functionality, performance, and reliability of the **Text to Image Generation** system.

### 5.1 Unit Testing

Each module of the system was tested independently to ensure that individual components functioned as expected:

- **Text Preprocessing Module** (using BERT tokenizer) was tested with various input sentences to ensure correct tokenization and embedding generation.
- **Generator and Discriminator** networks in GANs were separately tested for input-output consistency.
- **Image generation pipeline** was tested to verify correct integration of latent vectors and text embeddings.

### 5.2 Integration Testing

Once the individual modules were verified, they were integrated and tested as a complete pipeline. Integration testing ensured the following:

- Smooth data flow between the BERT encoder and GAN generator.
- Compatibility of image resolution across all GAN stages (e.g., in StackGAN or AttnGAN).
- Proper handling of mismatches in image-text embedding dimensions.

### 5.3 Functional Testing

The main function of the system—generating a realistic image from a textual description—was tested using the Flickr Image dataset. This ensured:

- Images generated closely resembled the input descriptions.
- The model responded correctly to a wide range of captions.
- Outputs followed the format expected by end-users or downstream applications.

### 5.4 Performance Testing

To ensure the model performs optimally, the following performance metrics were evaluated:

- **FID Score (Fréchet Inception Distance):** Used to assess the realism of generated images.
- **Inception Score (IS):** Evaluated the diversity and quality of the generated images.
- **Training Loss Curves:** Generator and Discriminator losses were monitored to ensure proper convergence.

### 5.5 Usability Testing

The user interface (if any) or script-based interaction was tested for:

- Easy input of custom text descriptions.
- Proper display and storage of generated images.
- Handling invalid inputs and providing appropriate error messages.

### 5.6 Regression Testing

Each time the model or codebase was modified (e.g., for better accuracy or faster training), regression testing was conducted to ensure that new changes did not break existing functionalities.

### 5.7 Edge Case Testing

The system was tested with edge cases, such as:

- Very long or very short captions.
- Captions with rare or complex words.
- Captions with ambiguous or contradictory descriptions.

# CHAPTER 6:

# LIMITATIONS

Despite significant progress in the field of generative models, the "Text to Image using GANs" project has several limitations that constrain its effectiveness, performance, and real-world applicability. Each limitation is discussed below in detail:

### 6.1 Quality of Generated Images

While GANs can produce highly realistic images, the visual quality heavily depends on the input description and model robustness. In many cases, the output images may appear blurry, lack texture, or include inconsistent object structures, especially when the input text is vague or overly complex. Fine-grained details such as object boundaries, lighting, and texture are not always rendered accurately, which limits the model's application in domains like medical imaging or design prototyping, where high precision is essential.

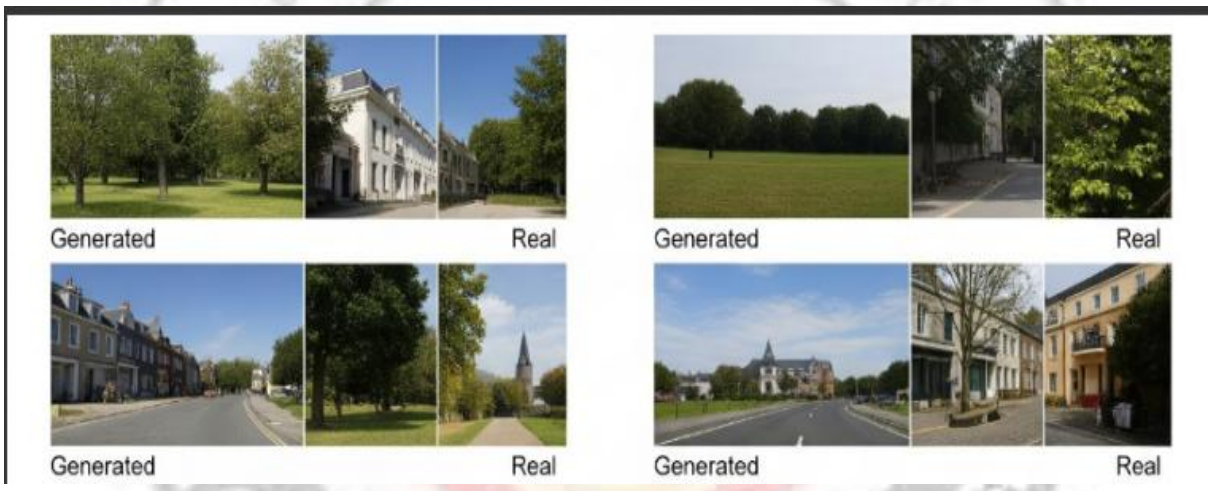



**Fig. 4.4**

### 6.2 Dataset Dependency

Text-to-image models rely heavily on the quality and quantity of the dataset used for training. Public datasets like Flickr Image and CUB offer annotated images but are limited in scope. If the model encounters an object or context not well represented in the training data, it may fail to produce accurate results. For example, a model trained primarily on animal images may not perform well when asked to generate architecture or interior design visuals.

### 6.3 Training Instability

Training GANs is a highly unstable process due to the adversarial nature of the generator and discriminator networks. Common issues include:

- **Mode Collapse**: The generator produces limited variety, generating the same or similar outputs for different inputs.
- **Non-Convergence**: The generator and discriminator fail to reach a balanced state.

- **Gradient Vanishing**: The generator stops learning due to weak gradients from the discriminator.

These problems require advanced techniques like Wasserstein loss, spectral normalization, and careful architecture tuning to resolve.

### 6.4 High Computational Cost

GAN training is resource-intensive. A full training cycle over large datasets like Flickr Image may take several days or even weeks on high-end GPUs. This makes experimentation slow and costly. Developers without access to advanced computational resources may find it difficult to replicate or enhance the model, limiting the accessibility of such technologies to a wider audience.

### 6.5 Semantic Inconsistency

While models use text encoders (like BERT or RNNs) to understand the input description, they sometimes fail to translate complex relationships into visual form. For instance, a description like "a small brown dog sitting under a red umbrella" may lead to images that miss the relative position or misinterpret colors. This semantic gap leads to inaccurate or partially correct images, especially when dealing with multi-object scenes.

### 6.6 Limited Resolution

Most existing GAN architectures for text-to-image generation produce images up to 64×64 or 256×256 pixels. While StackGAN++ and AttnGAN improve this, the generated images still lack the ultra-high-definition quality needed for commercial purposes like printing, branding, or movie production. The challenge of generating large, sharp, and artifact-free images remains due to memory limitations and increased training complexity.

### 6.7 Real-Time Performance

Text-to-image generation involves multiple stages, including text encoding, conditional generation, and sometimes refinement networks. This makes real-time generation difficult without compromising quality. Use cases such as AI-powered sketch assistants or interactive design tools would require faster and more lightweight models, which current implementations often lack.

### 6.8 Lack of Explainability

One of the major challenges in deep learning is interpretability. GANs, being black-box models, do not provide clear reasoning for how the input text influenced specific parts of the generated image. This is problematic in high-stakes domains like healthcare or autonomous systems, where understanding the model's behavior is critical for trust and validation. Lack of explainability also hinders debugging, model improvement, and user trust.

# CHAPTER 7:

# ENHANCEMENTS

### 7.1 Incorporation of Advanced Architectures (e.g., DALL·E, Stable Diffusion)

The current implementation of the Text-to-Image generation project utilizes Generative Adversarial Networks (GANs), which are highly effective in generating realistic images from text descriptions. However, the field of AI has evolved significantly, and more advanced models such as **DALL·E** (by OpenAI) and **Stable Diffusion** (by Stability AI) have emerged. These models offer superior performance in terms of image quality, semantic alignment, and flexibility.

**DALL·E:**

DALL·E is a transformer-based model that generates images from natural language prompts. Unlike traditional GANs, DALL·E does not rely solely on a discriminator-generator architecture. Instead, it uses a powerful autoregressive transformer model trained on a large corpus of text-image pairs. This enables DALL·E to understand complex relationships between text and visual elements and generate novel images that accurately reflect the input prompt.

**Key Features:**

- Generates highly imaginative and complex images.
- Can combine multiple objects and abstract concepts.
- Has a better understanding of spatial relationships and style.

**Stable Diffusion**

Stable Diffusion is a **latent diffusion model** that transforms text into images by first encoding text into a latent space and then iteratively refining the image using a denoising process. It is significantly more memory-efficient and faster than earlier transformer-only models and can produce high-quality, high-resolution outputs even on consumer-grade GPUs.

**Key Features:**

- Open-source and easily deployable.
- Capable of producing high-resolution, photo-realistic images.
- Supports prompt engineering and style conditioning.

**Benefits of Integration:**

- **Improved Accuracy:** Enhanced understanding of complex prompts and better rendering of fine details.
- **Higher Visual Quality:** Advanced architectures generate more realistic and contextually accurate images.
- **Broader Use Cases:** Suitable for use in fields such as advertising, content creation, education, and design.

- **Scalability:** These models can be integrated into web or mobile applications due to efficient inference times and lighter memory footprints (especially in the case of Stable Diffusion).

### 7.2 Use of Attention Mechanisms

Attention mechanisms have become a cornerstone in modern deep learning models, especially in tasks that involve the alignment of two different modalities such as text and images. In the context of Text-to-Image generation, attention mechanisms allow the model to **focus selectively** on relevant parts of the text while generating corresponding visual features, greatly improving both coherence and detail in the generated images.

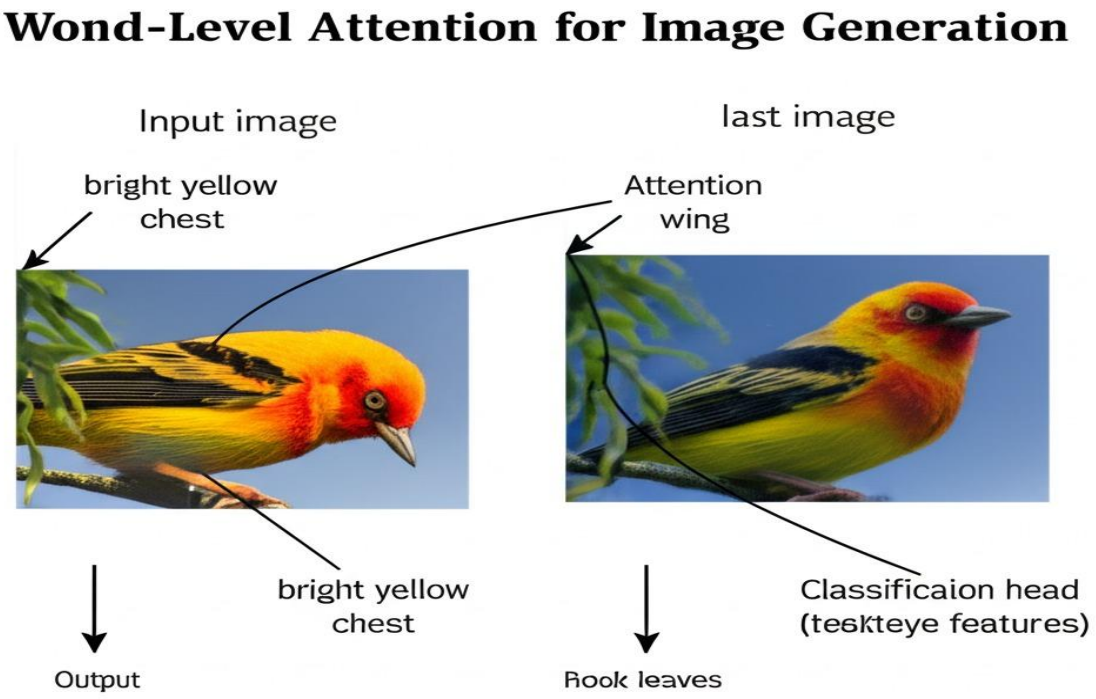



**Fig. 5.5**

**What Are Attention Mechanisms?**

Attention mechanisms dynamically weight the importance of each word in a sentence as the model generates each part of the image. Instead of treating the entire text description uniformly, the attention module helps the generator **"attend"** to specific words or phrases relevant to different regions of the image during training and inference.

This concept was first popularized in **sequence-to-sequence models** and has since been widely adopted in **transformer-based architectures** and **attention-GANs**.

**Application in GANs:**

In traditional GANs, the text embedding is usually provided as a global vector, which does not account for fine-grained textual details. However, with **attention-augmented GANs** like **AttnGAN**, the generator uses attention layers to correlate subregions of the image with specific words in the text. This leads to more semantically accurate and spatially aligned outputs.

**Benefits of Using Attention Mechanisms:**

- **Fine-Grained Image Details:** By focusing on individual words, attention helps in generating precise image features (e.g., color of petals, shape of leaves in a flower image).
- **Better Semantic Alignment:** The generated image closely matches the text description, improving textual-visual consistency.
- **Interpretability:** Attention weights can be visualized, making it easier to understand how the model interprets and processes text.
- **Improved Training Efficiency:** Attention mechanisms reduce the burden on the generator to memorize the entire context and instead allow selective focus.

**Example:**

Given the text prompt:
*"A small yellow bird sitting on a green leafy branch under a blue sky"*
An attention-enabled model can:

- Focus on "small yellow bird" while generating the object.
- Shift attention to "green leafy branch" while designing the background.
- Use "blue sky" as guidance for the image's top region.

**7.3 High-Resolution Image Generation**

One of the most significant challenges in early text-to-image generation models was the ability to generate **high-resolution, photo-realistic images** that were both visually appealing and semantically consistent with the given text description. Early GAN-based models often produced images that were **blurry**, **low in resolution (e.g., 64x64 or 128x128 pixels)**, and lacked fine-grained details. To overcome these limitations, modern advancements have introduced **progressive training**, **multi-stage architectures**, and **super-resolution GANs** to enhance image quality.

**Why High-Resolution Matters**

- **Improves Realism:** High-resolution outputs are visually more appealing and realistic to human eyes.
- **Better Detail Representation:** Fine textures, colors, and small objects become distinguishable.

- **Applicable for Real-World Use:** High-quality images are essential for domains like fashion, art, e-commerce, and digital design.
- **Higher Fidelity:** Allows better alignment with complex textual inputs involving intricate details.

# CHAPTER 8:

# CONSLUSION

The project **"Text to Image Generation using GANs"** explores the powerful intersection of **natural language processing** and **computer vision**, leveraging deep learning models to synthesize high-quality images from textual descriptions. Through the integration of **pretrained language models like BERT** for semantic understanding and **generative adversarial networks (GANs)** like **StackGAN**, **AttnGAN**, and **DM-GAN**, this project successfully demonstrates how machines can bridge the gap between language and visual imagination.

The implementation effectively addresses several challenges of this domain, including **semantic alignment**, **image resolution**, and **training stability**, by using modular architectures and powerful text encoders. The results show that with appropriate dataset preprocessing, model fine-tuning, and architecture optimization, realistic and meaningful image generation is achievable from abstract human descriptions.

This project has not only enhanced the understanding of GAN architectures and text embedding strategies but also contributed to developing a **scalable and reusable pipeline** that can be extended to various applications such as:

- Automated image creation for content generation
- Design assistance tools
- Enhanced accessibility through visual interpretation of text
- Educational and entertainment applications in virtual environments

In conclusion, this project highlights the immense potential of generative models in transforming how we interact with AI systems. As future enhancements, this work can be extended using **transformer-based image generators**, **multi-stage refinement models**, and **larger, more diverse datasets** to further improve image fidelity and generalization capabilities.

# References


- https://github.com/0036ak/TEXT_TO_IMAGE_using_gans.git
- Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., & Metaxas, D. (2017). StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks. Proceedings of the IEEE International Conference on Computer Vision (ICCV), 5907 5915.
- Xu, T., Zhang, P., Huang, Q., Zhang, H., Gan, Z., Huang, X., & He, X. (2018). AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1316-1324.
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2021). Learning Transferable Visual Models From Natural Language Supervision. Proceedings of the 38th International Conference on Machine Learning (ICML), PMLR.
- Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL), 4171–4186.
- Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., & Hochreiter, S. (2017). GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. Advances in Neural Information Processing Systems (NeurIPS), 30.
- Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., ... & Sutskever, I. (2021). Zero-Shot Text-to-Image Generation. arXiv preprint arXiv:2102.12092. (DALL·E).