***INTEL Unnati Industrial Training - 2025***

**Project Report**

**Team no: 5**

**Kanam Khushi Kumari | Pooja Basavaraj Nilugal**

**Title: Comic Crafter AI**

**Problem Statement:**

ComicCrafter AI is a generative AI based comic generator running locally on edge devices that generates a comic style story based on the input prompts given by the user.

**Project Overview:**

The AI Comic Generator is a creative tool that uses artificial intelligence to automatically generate a 4-panel comic strip based on a user-provided story prompt. It seamlessly combines text generation (powered by models like Mistral-7B or GPT-2) and image generation (using Stable Diffusion) to craft engaging comics with a charming retro cartoon aesthetic.

This project highlights how AI can enhance storytelling and visual creativity, blending natural language processing (NLP) with generative AI for an interactive comic-making experience.

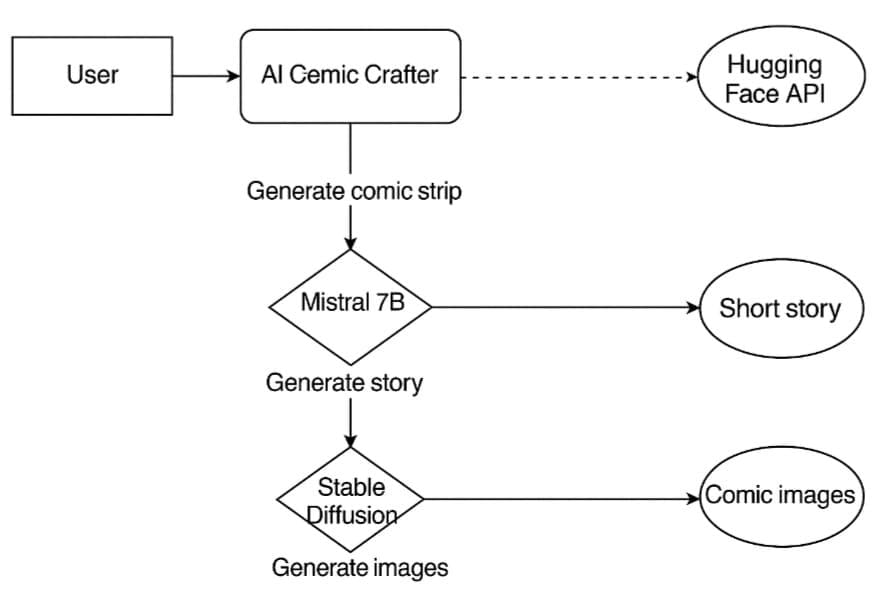
**Key Features:**

* **AI-Powered Storytelling:** Generates a short narrative based on user input.
* **Panel Segmentation:** Splits the story into four parts, each serving as a comic panel caption.
* **Vintage Cartoon Art:** Uses Stable Diffusion to create 1940s-style illustrations that complement the text.
* **Automated Comic Layout:** Seamlessly arranges captions and images into a structured comic strip.
* **Intuitive Web Interface:** Built with Gradio for smooth and interactive browser-based access.

**Technical Highlights:**

* **Hugging Face Transformers** – Drives text generation.
* **Stable Diffusion** – Produces retro-style cartoon visuals.
* **4-bit Quantization** – Enhances GPU efficiency.
* **Gradio UI** – Enables an interactive web experience.

**Flow Diagram:**



**Code:**

# Install required libraries

!pip install -q transformers diffusers torch matplotlib bitsandbytes gradio

# Import libraries

from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig

from diffusers import StableDiffusionPipeline

import torch

import gc

from PIL import Image, ImageDraw, ImageFont

import gradio as gr

import random

# Verify GPU availability

device = "cuda" if torch.cuda.is\_available() else "cpu"

print(f"Using device: {device}")

# Hugging Face token for gated models (replace with your token)

hf\_token = "paste your hf\_token"

# Configure 4-bit quantization for text generation (only if CUDA is available)

quantization\_config = None

if device == "cuda":

try:

quantization\_config = BitsAndBytesConfig(

load\_in\_4bit=True,

bnb\_4bit\_use\_double\_quant=True,

bnb\_4bit\_quant\_type="nf4",

bnb\_4bit\_compute\_dtype=torch.float16

)

except Exception as e:

print(f"Quantization not supported: {e}. Falling back to CPU.")

device = "cpu"

quantization\_config = None

# Load the text generation model (Mistral 7B)

try:

text\_model\_name = "mistralai/Mistral-7B-v0.1"

text\_tokenizer = AutoTokenizer.from\_pretrained(text\_model\_name, use\_auth\_token=hf\_token)

text\_model = AutoModelForCausalLM.from\_pretrained(

text\_model\_name,

quantization\_config=quantization\_config,

device\_map="auto",

use\_auth\_token=hf\_token

)

except Exception as e:

print(f"Failed to load text model: {e}")

# Fallback to a smaller model

text\_model\_name = "gpt2"

text\_tokenizer = AutoTokenizer.from\_pretrained(text\_model\_name)

text\_model = AutoModelForCausalLM.from\_pretrained(text\_model\_name).to(device)

# Load the image generation model (Stable Diffusion)

try:

image\_model\_name = "runwayml/stable-diffusion-v1-5"

image\_pipe = StableDiffusionPipeline.from\_pretrained(image\_model\_name, torch\_dtype=torch.float16)

image\_pipe = image\_pipe.to(device)

except Exception as e:

print(f"Failed to load image model: {e}")

raise RuntimeError("Image model could not be loaded. Please check your setup.")

# Function to generate text (story or dialogue)

def generate\_text(prompt):

try:

with torch.no\_grad():

inputs = text\_tokenizer(prompt, return\_tensors="pt").to(device)

outputs = text\_model.generate(

\*\*inputs,

max\_length=300,

num\_return\_sequences=1,

temperature=0.7,

top\_k=50,

top\_p=0.9,

do\_sample=True,

pad\_token\_id=text\_tokenizer.eos\_token\_id

)

generated\_text = text\_tokenizer.decode(outputs[0], skip\_special\_tokens=True)

# Clean up

del inputs, outputs

gc.collect()

torch.cuda.empty\_cache()

return generated\_text

except Exception as e:

print(f"Error generating text: {e}")

return f"Error generating story: {str(e)}"

# Function to generate a modern-style image

def generate\_modern\_image(prompt):

try:

modern\_prompt = (

f"{prompt}, modern digital art, clean lines, contemporary style, "

"smooth shading, vibrant colors, high detail, trending on artstation, "

"ultra HD, 8k resolution"

)

with torch.no\_grad():

image = image\_pipe(

modern\_prompt,

height=512,

width=512,

num\_inference\_steps=30

).images[0]

return image

except Exception as e:

print(f"Error generating image: {e}")

# Create error placeholder image

img = Image.new('RGB', (512, 512), color=(230, 240, 255))

draw = ImageDraw.Draw(img)

draw.text((50, 250), "Failed to generate image", fill=(50, 50, 50))

return img

# Function to split story into 4 parts for panels

def split\_story(story):

sentences = [s.strip() for s in story.split('.') if s.strip()]

if len(sentences) < 4:

# If not enough sentences, duplicate some

sentences = sentences \* (4 // len(sentences) + 1)

# Distribute sentences across 4 panels

panel\_texts = []

for i in range(4):

start = i \* len(sentences) // 4

end = (i + 1) \* len(sentences) // 4

panel\_text = '. '.join(sentences[start:end]) + '.'

panel\_texts.append(panel\_text)

return panel\_texts

# Function to create modern comic panel with text

def create\_comic\_panel(text, image, panel\_size=(512, 512)):

try:

# Modern color scheme

bg\_color = (240, 245, 255) # Very light blue

text\_bg\_color = (255, 255, 255, 200) # Semi-transparent white

text\_color = (30, 30, 30) # Dark gray

panel = Image.new("RGB", panel\_size, bg\_color)

draw = ImageDraw.Draw(panel)

# Resize and paste image (top 80% of panel)

img\_height = int(panel\_size[1] \* 0.80)

panel.paste(image.resize((panel\_size[0], img\_height)), (0, 0))

# Add text area (bottom 20%)

text\_area\_height = panel\_size[1] - img\_height

text\_area = Image.new('RGBA', (panel\_size[0], text\_area\_height), text\_bg\_color)

panel.paste(text\_area, (0, img\_height), text\_area)

# Add text

font = ImageFont.load\_default()

text\_position = (10, img\_height + 10)

draw.text(text\_position, text, fill=text\_color, font=font)

return panel

except Exception as e:

print(f"Error creating panel: {e}")

error\_img = Image.new('RGB', panel\_size, color=(230, 240, 255))

draw = ImageDraw.Draw(error\_img)

draw.text((50, 250), "Panel creation error", fill=(50, 50, 50))

return error\_img

# Main generation function

def generate\_comic(story\_prompt):

try:

# Generate story

story = generate\_text(f"Write a short story about: {story\_prompt}")

# Split into 4 parts

panel\_texts = split\_story(story)

# Generate panels

panels = []

for i, text in enumerate(panel\_texts):

image = generate\_modern\_image(text)

panel = create\_comic\_panel(text, image)

panels.append(panel)

return story, \*panels

except Exception as e:

print(f"Error in comic generation: {e}")

error\_img = Image.new('RGB', (512, 512), color=(230, 240, 255))

draw = ImageDraw.Draw(error\_img)

draw.text((50, 250), "Generation error", fill=(50, 50, 50))

return f"Error: {str(e)}", error\_img, error\_img, error\_img, error\_img

# Modern CSS styling

css = """

.gradio-container {

max-width: 1200px !important;

font-family: 'Segoe UI', sans-serif;

background: linear-gradient(135deg, #f5f7fa 0%, #e4e8f0 100%);

}

.panel-container {

display: flex;

flex-wrap: wrap;

justify-content: center;

gap: 15px;

margin-top: 20px;

}

.panel {

border: 1px solid #d1d9e6;

border-radius: 12px;

box-shadow: 8px 8px 16px #d1d9e6, -8px -8px 16px #ffffff;

transition: transform 0.3s ease;

}

.panel:hover {

transform: translateY(-5px);

}

.story-box {

background: rgba(255, 255, 255, 0.9);

padding: 20px;

border-radius: 12px;

margin-bottom: 25px;

box-shadow: 4px 4px 10px rgba(0,0,0,0.05);

border: 1px solid rgba(0,0,0,0.05);

}

h1 {

color: #4a6baf;

text-align: center;

margin-bottom: 25px;

}

button {

background: linear-gradient(135deg, #667eea 0%, #764ba2 100%) !important;

color: white !important;

border: none !important;

border-radius: 8px !important;

padding: 12px 24px !important;

font-weight: 600 !important;

box-shadow: 0 4px 6px rgba(0,0,0,0.1) !important;

transition: all 0.3s ease !important;

}

button:hover {

transform: translateY(-2px);

box-shadow: 0 7px 14px rgba(0,0,0,0.1) !important;

}

.textbox {

border-radius: 10px !important;

border: 1px solid #d1d9e6 !important;

box-shadow: inset 3px 3px 6px #d1d9e6, inset -3px -3px 6px #ffffff !important;

}

"""

# Gradio interface with modern theme

with gr.Blocks(css=css, title="Modern AI Comic Generator") as demo:

gr.Markdown("""

# 🚀 Modern AI Comic Generator

Enter a story idea below and the AI will generate a sleek 4-panel comic strip!

""")

with gr.Row():

with gr.Column():

story\_prompt = gr.Textbox(

label="Story Prompt",

placeholder="e.g., 'A robot who wants to be a chef'",

lines=3,

elem\_classes=["textbox"]

)

generate\_btn = gr.Button("Generate Comic", variant="primary")

with gr.Row():

story\_output = gr.Textbox(

label="Generated Story",

interactive=False,

elem\_classes=["story-box"]

)

with gr.Row(elem\_classes=["panel-container"]):

panel\_outputs = []

for i in range(4):

panel\_outputs.append(

gr.Image(

label=f"Panel {i+1}",

elem\_classes=["panel"],

width=512,

height=512

)

)

generate\_btn.click(

fn=generate\_comic,

inputs=story\_prompt,

outputs=[story\_output] + panel\_outputs

)

# Launch the app

demo.launch(share=True)

**Code Description:**

**AI Model Setup**

* **Text Generation (Mistral-7B/GPT-2):**
  + Loads **Mistral-7B** for AI-driven storytelling.
  + Utilizes **4-bit quantization** (if CUDA is available) for efficient memory use.
  + Falls back to **GPT-2** if Mistral-7B fails to load.
* **Image Generation (Stable Diffusion v1.5):**
  + Loads **Stable Diffusion** to generate high-quality AI illustrations.
  + Uses **FP16 precision** for faster GPU inference.
  + Produces images in a **modern digital art style**.

**Story Creation Process**

* Accepts a **user prompt** (e.g., *"A robot who dreams of becoming a chef"*).
* Creates a short AI-written story (up to **300 tokens**) with controlled randomness:
  + **Temperature:** 0.7
  + **Top-k:** 50
  + **Top-p:** 0.9
* Optimizes GPU memory by **clearing the cache** after processing.

**Illustration Generation**

* Converts each story segment into an **AI-generated illustration** in a **modern digital style**.
* Enhances the prompt with keywords such as:
  + *"Modern digital art, clean lines, contemporary style, vibrant colors, ultra HD, trending on ArtStation."*
* Generates **512x512 resolution** images with **30 inference steps** for high-quality results.
* If image generation fails, returns a **placeholder error image**.

**Comic Panel Assembly**

* Splits the story into **four segments** for comic panels.
  + If the story is too short, sentences are **repeated** to ensure four panels.
* Combines text and images into **visually structured comic panels**:
  + **Top 80%** – AI-generated illustration.
  + **Bottom 20%** – Story text (overlayed using PIL).

**User Interface & Interaction**

* A **modern UI** built with **Gradio Blocks**, featuring:
  + **Story input box** for user prompts.
  + **"Generate Comic" button** to trigger the process.
  + **Story output box** displaying the full AI-generated narrative.
  + **4 image panels** showcasing the completed comic strip.
* Custom **CSS styling** for a sleek and engaging user experience.

**Robust Error Handling**

* Handles failures gracefully:
  + **Falls back to GPT-2** if Mistral-7B encounters issues.
  + Displays **error messages and placeholder images** instead of crashing.

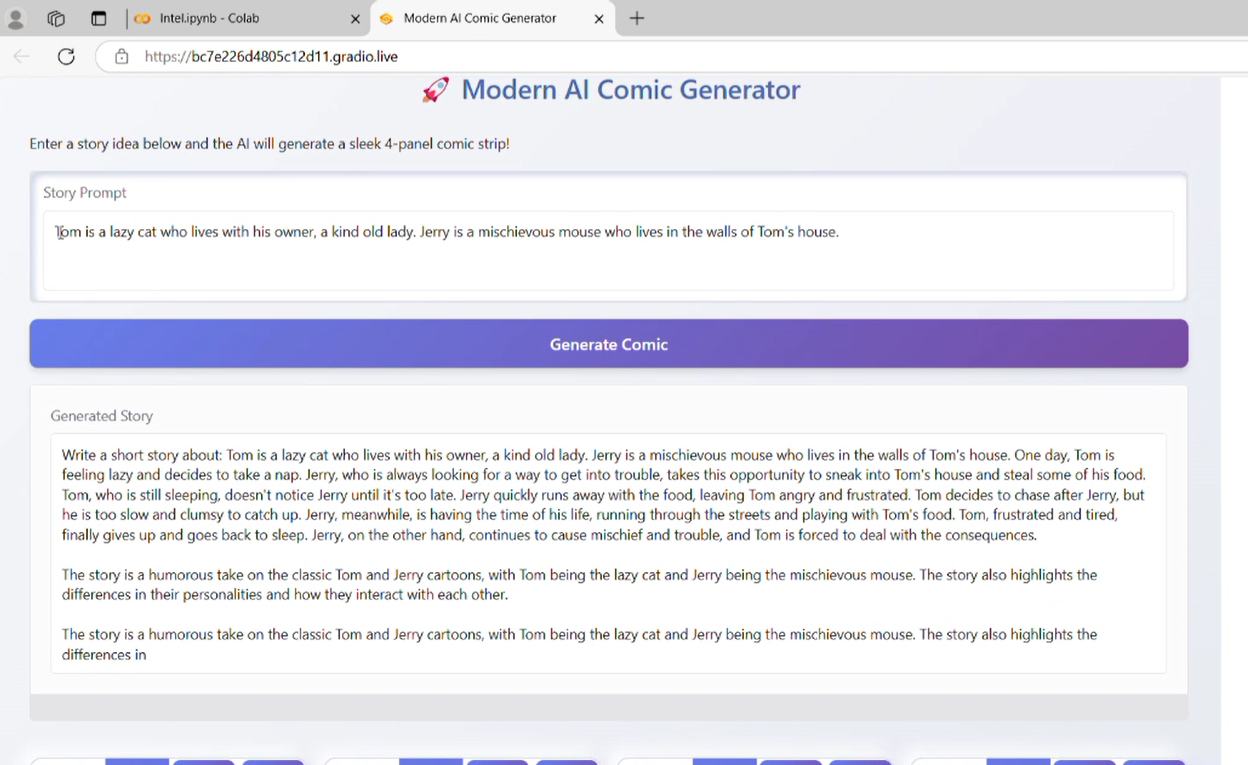
**Performance Enhancements**

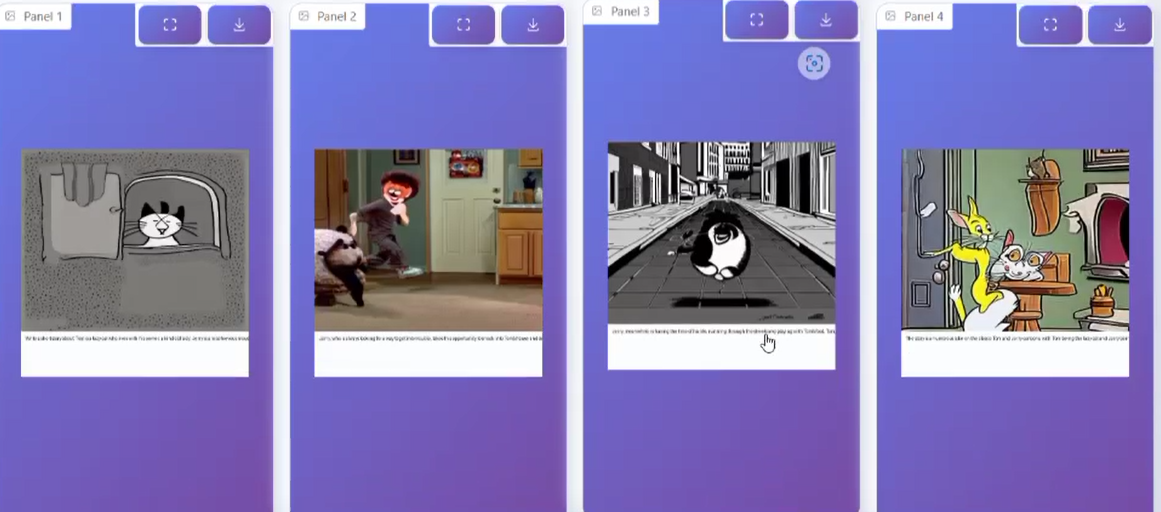
* **Clears GPU cache** after each generation to prevent memory leaks.
* Uses **4-bit quantization** to minimize VRAM usage.
* Efficiently batches operations (**Text → Split → Images → Panels**).

**Final Output & Deployment**

* Generates a **4-panel AI comic strip** featuring:
  + A **coherent, AI-crafted story**.
  + **High-quality modern digital illustrations**.
  + **Text captions** beneath each image.
* Deployed via **Gradio**, enabling users to create AI-powered comics with **a single click**

**Output 1:**





Output 2:

