Project Report

Team no: 8

Gagan BV | Bhoomika L | Shruthi P

Title: Comic Crafter AI

Problem Statement:

Creation of An AI Based Comic Generator Capable of Running Locally on Edge devices that generates a comic Style Story based on the Input.

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 combines text generation (using Mistral-7B or GPT-2) and image generation (using Stable Diffusion) to create a cohesive comic with a retro cartoon aesthetic.

This project showcases how AI can assist in creative storytelling and visual art, blending natural language processing (NLP) and generative AI into an engaging comic-making experience.

Key Features:

- Story Generation – The AI writes a short story based on the user's input.

- Panel Splitting – The story is divided into four parts, each serving as a comic panel’s caption.
- Retro Cartoon Art – Stable Diffusion generates 1940s-style cartoon images matching each panel’s text.
- Automatic Layout – The system assembles the comic strip with captions and images.
- User-Friendly Interface – Built with Gradio, allowing easy interaction in a web browser.

Technical Highlights:

- **Hugging Face Transformers** (for text generation)
- **Stable Diffusion** (for image generation)
- **4-bit Quantization** (for efficient GPU usage)
- **Gradio UI** (for interactive web deployment)

Steps to Run on Google Colab:

- Open Google Colab: Go to Google Colab.
- Create a New Notebook: Click on "File" > "New Notebook".
- Install Required Libraries: Copy and paste the following code into a cell and run it:

```
>>!pip install -q -r requirements.txt
```

- Copy the Code: Copy the entire code provided in this repository into a new cell in the Colab notebook.
- Set Hugging Face Token: Replace the hf_token variable in the code with your Hugging Face token. You can obtain a token by creating an account on Hugging Face.

- Run the Code: Execute the cell containing the code. After a few moments, the Gradio interface will launch.
- Interact with the App: Enter a story prompt and click "Generate Comic" to see the generated comic strip.

Steps to Run Locally:

- Install Python: Ensure you have Python 3.7 or higher installed. You can download it from python.org.
- Set Up a Virtual Environment (Optional): It is recommended to create a virtual environment to manage dependencies.

  ```
  >>python -m venv comic_crafter_env
  ```

  ```
  >>source comic_crafter_env/bin/activate  # On Windows use ` comic_crafter_env\Scripts\activate`
  ```

- Install Required Libraries: Use pip to install the required libraries:

  ```
  >>pip install -r requirements.txt
  ```

- Copy the Code: Copy the entire code provided in this repository into a Python file (e.g., comic_crafter.py).
- Set Hugging Face Token: Replace the hf_token variable in the code with your Hugging Face token.
- Run the Code: Execute the Python file (terminal / Command prompt):

  ```
  python comic_crafter.py
  ```

- Access the App: After running the code, a local server will start. Open the provided URL (usually http://127.0.0.1:7860) in your web browser to access the Gradio interface.

- Interact with the App: Enter a story prompt and click "Generate Comic" to see the generated comic strip.

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 = "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"

# 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 retro cartoon-style image

def generate_retro_cartoon_image(prompt):

    try:

        retro_cartoon_prompt = (

            f"{prompt}, classic 1940s cartoon style, hand-drawn animation, "

            "bold outlines, vibrant colors, cel-shaded, exaggerated expressions, "

            "vintage animation, retro cartoon"

        )

        with torch.no_grad():

            image = image_pipe(

                retro_cartoon_prompt,

                height=512,

                width=512,

                num_inference_steps=30  # More steps for better quality
            ).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=(255, 200, 200))
        draw = ImageDraw.Draw(img)
        draw.text((50, 250), "Failed to generate image", fill=(0, 0, 0))
        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 comic panel with text
def create_comic_panel(text, image, panel_size=(512, 512)):
    try:
        panel = Image.new("RGB", panel_size, "white")
        draw = ImageDraw.Draw(panel)
        # Resize and paste image (top 75% of panel)
        img_height = int(panel_size[1] * 0.75)
        panel.paste(image.resize((panel_size[0], img_height)), (0, 0))
        # Add text (bottom 25%)
        font = ImageFont.load_default()
        text_position = (10, img_height + 10)
        draw.text(text_position, text, fill="black", font=font)
        return panel
    except Exception as e:
        print(f"Error creating panel: {e}")
        error_img = Image.new('RGB', panel_size, color=(255, 200, 200))
        draw = ImageDraw.Draw(error_img)
        draw.text((50, 250), "Panel creation error", fill=(0, 0, 0))
        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_retro_cartoon_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=(255, 200, 200))
        draw = ImageDraw.Draw(error_img)
        draw.text((50, 250), "Generation error", fill=(0, 0, 0))
        return f"Error: {str(e)}", error_img, error_img, error_img, error_img

# Custom CSS for styling
css = """
.gradio-container {
    max-width: 1200px !important;
}
.panel-container {
    display: flex;
    flex-wrap: wrap;
    justify-content: center;
    gap: 10px;
    margin-top: 20px;
}
```

```
.panel {
  border: 2px solid #ddd;
  border-radius: 8px;
  box-shadow: 0 4px 6px rgba(0,0,0,0.1);
}
.story-box {
  background: #f8f9fa;
  padding: 15px;
  border-radius: 8px;
  margin-bottom: 20px;
}
"""
# Gradio interface
with gr.Blocks(css=css, title="AI Comic Generator") as demo:
  gr.Markdown("""
  # 🎨 AI Comic Generator
  Enter a story idea below and the AI will generate a 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
      )
      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
  )
demo.launch(share=True) # Launch the app
```

Code Description:

Model Loading & Setup

- Text Generation Model (Mistral-7B/GPT-2):
    - Loads a large language model (LLM) for generating stories.
    - Uses 4-bit quantization (if GPU is available) to optimize memory usage.
    - Falls back to GPT-2 if Mistral-7B fails.
- Image Generation Model (Stable Diffusion):
    - Loads Stable Diffusion v1.5 for generating retro cartoon-style images.
    - Uses FP16 precision for faster GPU inference.

Text Generation

- Takes a user-provided prompt (e.g., "A robot who wants to be a chef").
- Generates a short story (300 tokens max) with controlled randomness (temperature, top-k, top-p sampling).
- Cleans up GPU memory afterward to avoid crashes.

Image Generation

- Converts each story segment into a retro cartoon-style image by modifying the prompt (e.g., adding "1940s cartoon style").
- Ensures 512x512 resolution with 30 inference steps for better quality.
- Returns a placeholder image if generation fails.

Comic Panel Creation

- Splits the story into 4 segments (for 4 comic panels).
- If the story is too short, repeats sentences to fill panels.
- Combines text and images into panels:
  - Top 75% = AI-generated image.
  - Bottom 25% = Story text (using PIL for drawing).

Gradio Interface

- Provides a user-friendly web UI with:
  - A text input box for the story idea.
  - A "Generate Comic" button to trigger the process.
  - A text output box showing the full story.
  - Four image panels displaying the comic.
- Uses custom CSS for a polished look.

Error Handling & Fallbacks

- Gracefully handles failures (e.g., if models don't load).
- Falls back to simpler models (GPT-2 if Mistral fails).
- Displays error images/text instead of crashing.

Performance Optimization

- Clears GPU cache after each generation to prevent memory leaks.
- Uses quantization (4-bit) to reduce VRAM usage.
- Batches operations efficiently (text → split → images → panels).

Final Output

- Returns a 4-panel comic strip with:
  - A coherent AI-generated story.
  - Vintage cartoon-style illustrations.
  - Text captions below each image.