```
# ComicCrafter AI - Implementation for Google Colab

!pip install -q gradio diffusers transformers accelerate safetensors
!pip install -q xformers controlnet-aux opencv-python
!pip install -q huggingface_hub
!pip install bitsandbytes

import os
import sys
import torch
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image, ImageDraw, ImageFont
import textwrap
import json
import gradio as gr
import time
import traceback
import logging
import cv2
import re
from diffusers import (
    StableDiffusionXLPipeline,
    DPMSolverMultistepScheduler,
    ControlNetModel,
    StableDiffusionControlNetPipeline,
    UniPCMultistepScheduler
)
from huggingface_hub import login
from transformers import (
    AutoTokenizer,
    AutoModelForCausalLM,
    pipeline,
    T5Tokenizer,
    T5ForConditionalGeneration
)
```

```
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 46.5/46.5 MB 23.2 MB/s eta 0:
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 322.2/322.2 kB 22.8 MB/s eta
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 95.2/95.2 kB 8.4 MB/s eta 0:0
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.3/11.3 MB 116.8 MB/s eta 0
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 72.0/72.0 kB 6.7 MB/s eta 0:0
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 3.6 MB/s eta 0
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 27.9 MB/s eta 0:
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 20.2 MB/s eta 0:
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 45.7 MB/s eta
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 2.2 MB/s eta 0
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 4.3 MB/s eta 0
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 8.7 MB/s eta 0:0
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 5.9 MB/s eta 0
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 4.7 MB/s eta 0
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 47.3 MB/s eta 0:
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.3/62.3 kB 3.4 MB/s eta 0:0
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.4/43.4 MB 15.4 MB/s eta 0:
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 282.4/282.4 kB 22.9 MB/s eta
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 510.0/510.0 kB 37.1 MB/s eta
  Collecting bitsandbytes
    Downloading bitsandbytes-0.45.4-py3-none-manylinux_2_24_x86_64.whl.metadata (5
  Requirement already satisfied: torch<3,>=2.0 in /usr/local/lib/python3.11/dist-p
  Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-pac
  Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packag
  Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/pytho
  Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packag
  Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages
  Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages
  Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local/li
  Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/local/
  Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/li
  Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/pyt
  Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib/py
  Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/pyt
  Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/lib/
  Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/lib/
  Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local/li
  Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/p
  Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python
  Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/pyth
  Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/lib
  Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-p
  Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-p
  Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/d
  Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist
  Downloading bitsandbytes-0.45.4-py3-none-manylinux_2_24_x86_64.whl (76.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 76.0/76.0 MB 10.1 MB/s eta 0:
  Installing collected packages: bitsandbytes
  Successfully installed bitsandbytes-0.45.4
```

```
# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Check GPU status
!nvidia-smi
```

```
Wed Apr  2 11:17:10 2025
+-----------------------------------------------------------------------------------
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version:
|-----------------------------------------+------------------------+-------------
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Un
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  C
|                                         |                        |
|=========================================+========================+=============
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |
| N/A   48C    P8              9W /   70W |       2MiB /  15360MiB |      0%
|                                         |                        |
+-----------------------------------------+------------------------+-------------

+-----------------------------------------------------------------------------------
| Processes:
|  GPU   GI   CI        PID   Type   Process name                              G
|        ID   ID                                                               U
|===================================================================================
|  No running processes found
```

```
    +----------------------------------------------------------------------------------------



# 1. STORY GENERATION MODEL
def setup_story_model():
    logger.info("Setting up enhanced story generation model...")
    try:
        # Use Mistral-7B-Instruct for enhanced story generation
        model_name = "filipealmeida/Mistral-7B-Instruct-v0.1-sharded"

        # Authenticate with Hugging Face
        from huggingface_hub import login
        from getpass import getpass
        hf_token = getpass("Enter your Hugging Face token: ")  # Hidden input
        login(token=hf_token)


        # Check for available compute resources
        device = "cuda" if torch.cuda.is_available() else "cpu"
        logger.info(f"Using device: {device}")

        # Use 4-bit quantization to reduce memory requirements
        try:
            from transformers import BitsAndBytesConfig
            import bitsandbytes as bnb

            # Configure quantization parameters for efficient loading
            bnb_config = BitsAndBytesConfig(
                load_in_4bit=True,
                bnb_4bit_quant_type="nf4",
                bnb_4bit_compute_dtype=torch.float16
            )

            # Load the tokenizer
            tokenizer = AutoTokenizer.from_pretrained(model_name, use_fast=False)

            # Load model with quantization
            model = AutoModelForCausalLM.from_pretrained(
                model_name,
                device_map="auto",
                quantization_config=bnb_config,
                trust_remote_code=True
            )

            logger.info(f"Successfully loaded {model_name} with 4-bit quantization")

        except ImportError:
            # Fall back to regular loading if bitsandbytes is not available
            logger.warning("BitsAndBytes not available, loading with standard configuration

            # Configure torch settings for efficiency
            torch_dtype = torch.float16 if torch.cuda.is_available() else torch.float32

            # Load tokenizer
            tokenizer = AutoTokenizer.from_pretrained(model_name, use_fast=False)

            # Load model with appropriate settings
            model = AutoModelForCausalLM.from_pretrained(
                model_name,
                torch_dtype=torch_dtype,
                device_map="auto" if device == "cuda" else None,
                trust_remote_code=True
            )

            # Apply padding settings needed for Mistral
            tokenizer.pad_token = tokenizer.eos_token
            if tokenizer.pad_token is None:
                tokenizer.pad_token_id = 0  # Use an alternative padding token if needed

            logger.info(f"Successfully loaded {model_name} with standard configuration")

        # Prepare model for generation with appropriate configuration for Mistral
        generation_config = model.generation_config
        generation_config.max_new_tokens = 512
        generation_config.temperature = 0.7
        generation_config.top_p = 0.9
        generation_config.repetition_penalty = 1.1
        generation_config.pad_token_id = tokenizer.pad_token_id
        generation_config.eos_token_id = tokenizer.eos_token_id
        model.generation_config = generation_config

        # Add a helper function for text generation
        def generate_story(prompt, max_length=512, temperature=0.7):
            """Generate a story using the Mistral-7B model with the given prompt."""
            formatted_prompt = f"<s>[INST] {prompt} [/INST]"
            inputs = tokenizer(formatted_prompt, return_tensors="pt").to(device)

            with torch.no_grad():
                outputs = model.generate(
                    **inputs,
                    max_new_tokens=max_length,
                    temperature=temperature,
                    top_p=0.9,
                    repetition_penalty=1.1,
                    pad_token_id=tokenizer.pad_token_id,
                    eos_token_id=tokenizer.eos_token_id,
                )

            story = tokenizer.decode(outputs[0], skip_special_tokens=True)

            # Split story into narration and dialogues
            story_lines = story.split("\n")
            narration = "\n".join([line for line in story_lines if not line.startswith('"')
            dialogues = "\n".join([line for line in story_lines if line.startswith('"')])

            return {"narration": narration, "dialogues": dialogues}

        # Attach the helper function to the model
        model.generate_story = generate_story

        return tokenizer, model

    except Exception as e:
        error_msg = f"Error setting up Mistral-7B model: {str(e)}"
        logger.error(error_msg)
        traceback.print_exc()

        # First fallback: T5 model
        try:
            logger.warning("Falling back to T5 model")
            model_name = "google/flan-t5-base"
            tokenizer = T5Tokenizer.from_pretrained(model_name)
            model = T5ForConditionalGeneration.from_pretrained(model_name)
            logger.info(f"Fallback to {model_name} successful")
            return tokenizer, model
        except Exception as t5_error:
            logger.error(f"T5 fallback failed: {str(t5_error)}")

            # Second fallback: distilGPT2
            try:
                logger.warning("Falling back to distilGPT2")
                model_name = "distilgpt2"
                tokenizer = AutoTokenizer.from_pretrained(model_name, use_fast=False)
                model = AutoModelForCausalLM.from_pretrained(model_name)
                logger.info(f"Fallback to {model_name} successful")
                return tokenizer, model
            except Exception as gpt2_error:
                logger.error(f"All fallbacks failed: {str(gpt2_error)}")
                raise Exception(f"Failed to set up any story generation model: {error_msg}"



# 2. IMAGE GENERATION MODEL
def setup_image_model():
    logger.info("Setting up enhanced image generation model...")
    try:
        # Check CUDA availability
        if not torch.cuda.is_available():
            logger.warning("CUDA not available! Falling back to CPU (this will be slow)")

        # Use Stable Diffusion XL Turbo for faster generation
        model_id = "stabilityai/stable-diffusion-xl-base-1.0"

        # Use efficient scheduler
        scheduler = DPMSolverMultistepScheduler.from_pretrained(
            model_id,
            subfolder="scheduler"
        )

        # Load pipeline with optimizations
        pipe = StableDiffusionXLPipeline.from_pretrained(
            model_id,
            scheduler=scheduler,
            torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
            variant="fp16" if torch.cuda.is_available() else None,
            use_safetensors=True
        )

        # Move to appropriate device
        device = "cuda" if torch.cuda.is_available() else "cpu"
        pipe = pipe.to(device)

        # Enable optimizations
        if device == "cuda":
            # Try to enable xformers
            try:
                import xformers
                pipe.enable_xformers_memory_efficient_attention()
                logger.info("Successfully enabled xformers")
            except (ImportError, AttributeError):
                logger.warning("xformers not available, using attention slicing")
                pipe.enable_attention_slicing(1)

        logger.info(f"Successfully loaded Stable Diffusion XL on {device}")
        return pipe
    except Exception as e:
        error_msg = f"Error setting up image model: {str(e)}"
        logger.error(error_msg)
        traceback.print_exc()

        # Fallback to smaller model
        try:
            model_id = "CompVis/stable-diffusion-v1-4"
            pipe = StableDiffusionPipeline.from_pretrained(
                model_id,
                torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
                safety_checker=None,
                requires_safety_checker=False
            ).to("cuda" if torch.cuda.is_available() else "cpu")
            pipe.enable_attention_slicing()
            logger.info(f"Fallback to {model_id} successful")
            return pipe
        except:
            raise Exception(error_msg)



# 3. STORY GENERATION WITH 4-PART NARRATIVE
# Enhance story generation for better descriptive text like in examples
def generate_story(prompt, num_panels=4, tokenizer=None, model=None):
    logger.info(f"Generating structured story from prompt: {prompt}")

    try:
        # For Mistral model
        if hasattr(model, 'generate_story'):
            # Create structured storytelling prompts for each panel
            panel_prompts = [
                f"Create an introduction for a comic story about {prompt}. Set the scene an
                f"Continue the story about {prompt} with rising action and challenges.",
                f"Write the climactic moment of the story about {prompt}.",
                f"Conclude the story about {prompt} with a resolution and lesson learned."
            ]

            story_data = {}
            for i, panel_prompt in enumerate(panel_prompts[:num_panels], 1):
                # Generate content for each panel
                result = model.generate_story(panel_prompt)

                # Part titles
                part_titles = ["INTRODUCTION", "DEVELOPMENT", "CLIMAX", "RESOLUTION"]
                part_title = part_titles[i-1] if i <= len(part_titles) else f"PART {i}"

                # Store panel data
                story_data[f"panel_{i}"] = {
                    "part": part_title,
                    "scene": result["narration"].strip(),
                    "characters": ["Character", "Supporting Character"],
                    "dialogue": result["dialogues"].strip() if result["dialogues"] else "Th
                }
            return story_data

            # Try to extract JSON structure
            try:
                # Find anything that looks like JSON
                json_match = re.search(r'(\{.*\})', story_text, re.DOTALL)
                if json_match:
                    story_json = json.loads(json_match.group(1))
                    return story_json
            except:
                # If JSON parsing fails, proceed with manual structuring
                pass

        # Detect story type from prompt
        is_fable = any(word in prompt.lower() for word in ["fable", "moral", "lesson", "tor
        is_jungle = any(word in prompt.lower() for word in ["jungle", "mowgli", "wolf", "ti

        # Create structured panel data
        story_data = {}

        if "hare and tortoise" in prompt.lower() or "tortoise and hare" in prompt.lower():
            # Specific story for Hare and Tortoise
            story_data = {
                "panel_1": {
                    "part": "INTRODUCTION",
                    "scene": f"{prompt} - In a sun-drenched meadow, a speedy hare named Har
                    "characters": ["Hare", "Tortoise"],
                    "dialogue": "I'm the fastest in the meadow! None of you could ever beat
                },
                "panel_2": {
                    "part": "CHALLENGE",
                    "scene": f"{prompt} - One day, Toby, determined to put an end to Harry'
                    "characters": ["Hare", "Tortoise"],
                    "dialogue": "I accept your challenge! This will be the easiest race eve
                },
                "panel_3": {
                    "part": "CONFLICT",
                    "scene": f"{prompt} - The hare shot off like a bullet, his legs pumping
                    "characters": ["Hare", "Tortoise"],
                    "dialogue": "I'm so far ahead, I can take a quick break!"
                },
                "panel_4": {
                    "part": "RESOLUTION",
                    "scene": f"{prompt} - In the end, it was Toby, not Harry, who crossed t
                    "characters": ["Hare", "Tortoise"],
                    "dialogue": "Slow and steady wins the race!"
                }
            }
        elif "mowgli" in prompt.lower() or "jungle book" in prompt.lower():
            # Specific story for Mowgli
            story_data = {
                "panel_1": {
                    "part": "INTRODUCTION",
                    "scene": f"{prompt} - A young boy named Mowgli is raised by wolves in t
                    "characters": ["Mowgli", "Wolf"],
                    "dialogue": "The jungle is my home, and the wolves are my family."
                },
                "panel_2": {
                    "part": "DEVELOPMENT",
                    "scene": f"{prompt} - Mowgli grows up in the jungle, learning the ways
                    "characters": ["Mowgli", "Bagheera", "Baloo"],
                    "dialogue": "The jungle has many lessons to teach you, little brother."
                },
                "panel_3": {
                    "part": "CONFLICT",
                    "scene": f"{prompt} - Shere Khan attacks during a drought when the anim
                    "characters": ["Mowgli", "Shere Khan"],
                    "dialogue": "I will not run from you, Shere Khan!"
                },
                "panel_4": {
                    "part": "RESOLUTION",
                    "scene": f"{prompt} - Ultimately, Mowgli defeats Shere Khan and the tig
                    "characters": ["Mowgli", "Villagers"],
                    "dialogue": "I may live among humans, but the jungle will always be a p
                }
            }
        else:
            # Generic 4-part narrative structure with richly detailed scenes
            # Create story narrative details based on the prompt
            if "superhero" in prompt.lower():
                narrative_details = [
                    f"In the towering metropolis of New Horizon City, chaos erupts as a men
                    f"The situation intensifies as our hero confronts the mounting threat.
                    f"In a heart-stopping moment, hero and villain clash in an epic confron
                    f"With courage and determination, our hero prevails against the formida
                ]
            elif "robot" in prompt.lower() or "ai" in prompt.lower():
                narrative_details = [
                    f"In a world where technology has advanced beyond imagination, a unique
                    f"The robot begins to explore its environment and capabilities, encount
                    f"A critical moment arrives when the robot must make a choice that will
                    f"The robot's choice reveals a profound understanding of what it means
                ]
            elif "detective" in prompt.lower() or "mystery" in prompt.lower():
                narrative_details = [
                    f"In the shadows of a rain-slicked city, a determined detective examine
                    f"Diving deeper into the investigation, the detective follows a trail o
                    f"In a tense confrontation, the detective comes face-to-face with the m
                    f"As dawn breaks over the city, the case reaches its conclusion. The de
                ]
            elif any(word in prompt.lower() for word in ["magic", "wizard", "fantasy", "dra
                narrative_details = [
                    f"In a realm where magic flows like rivers through ancient lands, our p
                    f"As our hero learns to harness their newfound abilities, dark forces t
                    f"Magic crackles through the air as our hero confronts the ultimate mag
                    f"The dust settles on a world forever changed by the magical conflict.
                ]
            else:
                # Generic narrative for any other type of story
                narrative_details = [
                    f"Our story begins with the introduction of our main character in their
                    f"The adventure truly begins as our main character faces their first re
                    f"Everything comes to a head in a dramatic confrontation. The stakes ha
                    f"The dust settles as our story reaches its meaningful conclusion. Our
                ]

            # Create structured panel data
            for i in range(1, num_panels + 1):
                part_idx = min(i-1, len(narrative_details)-1)
            # Part titles
            part_titles = ["INTRODUCTION", "DEVELOPMENT", "CLIMAX", "RESOLUTION"]
            part_title = part_titles[part_idx] if part_idx < len(part_titles) else f"PA

            # Characters based on story content
            if "superhero" in prompt.lower():
                if i == 1:
                    characters = ["Hero", "Citizens"]
                elif i == num_panels:
                    characters = ["Triumphant Hero", "Grateful Citizens"]
                else:
                    characters = ["Hero", "Villain"]
            elif "robot" in prompt.lower() or "ai" in prompt.lower():
                if i == 1:
                    characters = ["Robot", "Human Observer"]
                elif i == num_panels:
                    characters = ["Evolved Robot", "Humans"]
                else:
                    characters = ["Robot", "Human Companion"]
            elif "detective" in prompt.lower():
                if i == 1:
                    characters = ["Detective", "Police Officer"]
                elif i == num_panels:
                    characters = ["Detective", "Culprit"]
                else:
                    characters = ["Detective", "Witness"]
            else:
                characters = ["Protagonist", "Supporting Character"]

            # Dialogues matching narrative tone
            if part_idx == 0:
                dialogue = "This is just the beginning of something extraordinary."
            elif part_idx == 1:
                dialogue = "I never expected it would be this challenging!"
            elif part_idx == 2:
                dialogue = "This is the moment that changes everything!"
            elif part_idx == 3:
                dialogue = "We've learned that the greatest power comes from within."

            # Store panel data with rich scene description
            story_data[f"panel_{i}"] = {
                "part": part_title,
                "scene": f"{prompt} - {narrative_details[part_idx]}",
                "characters": characters,
                "dialogue": dialogue
            }

        logger.info(f"Generated rich narrative structure with {num_panels} panels")
        return story_data

    except Exception as e:
        error_msg = f"Error generating story: {str(e)}"
        logger.error(error_msg)
        traceback.print_exc()

        # Fallback to simple structure with some narrative
        fallback_story = {}
        for i in range(1, num_panels + 1):
            if i == 1:
                part = "Introduction"
                desc = "Our story begins as characters are introduced and the setting is es
            elif i == num_panels:
                part = "Resolution"
                desc = "The story concludes with a meaningful resolution and lesson learned
            elif i == 2 and num_panels >= 3:
                part = "Conflict"
                desc = "Challenges arise as our characters face obstacles in their journey.
            else:
                part = "Development"
                desc = "The plot thickens as events unfold in unexpected ways."

            fallback_story[f"panel_{i}"] = {
                "part": part,
                "scene": f"{prompt} - {desc}",
                "characters": ["Character"],
                "dialogue": f"This is where our story {'begins' if i == 1 else 'concludes'
            }
        return fallback_story




# 4. IMAGE GENERATION WITH BETTER PROMPTS
def generate_panel_image(panel_data, style="comic book", pipe=None):
    scene = panel_data['scene']
    part = panel_data.get('part', '')
    logger.info(f"Generating enhanced image for panel: {scene}")

    try:
        # Construct detailed prompt for better image quality
        characters = ", ".join(panel_data["characters"])

        # Enhanced prompt engineering
        style_prompts = {
            "comic book": "detailed comic book style art, vibrant colors, dynamic compositi
            "manga": "manga style artwork, black and white, dramatic lighting, speed lines,
            "cartoon": "cartoon style, vibrant colors, exaggerated expressions, simple back
            "sketch": "detailed sketch, pencil drawing, dynamic lines, cross-hatching, comi
            "pixel art": "pixel art style, 16-bit game aesthetic, limited color palette, cl
        }

        # Add emotional tone based on part
        emotional_tone = ""
        if "INTRODUCTION" in part:
            emotional_tone = "establishing shot, introduction scene, character introduction
        elif "CONFLICT" in part or "STORYLINE" in part:
            emotional_tone = "dramatic tension, conflict visible, characters in challenging
        elif "CLIMAX" in part:
            emotional_tone = "peak action, intense moment, dramatic lighting, dynamic pose,
        elif "RESOLUTION" in part:
            emotional_tone = "resolving action, emotional conclusion, characters showing re

        # Complete prompt construction
        style_prompt = style_prompts.get(style, style_prompts["comic book"])
        prompt = f"{scene} with {characters}, {emotional_tone}, {style_prompt}, highly deta

        # Negative prompt to avoid common issues
        negative_prompt = "deformed, blurry, bad anatomy, disfigured, poorly drawn face, mu
        # Generate image with optimized parameters
        image = pipe(
            prompt=prompt,
            negative_prompt=negative_prompt,
            num_inference_steps=30,
            guidance_scale=7.5,
            height=512,
            width=512,
        ).images[0]

        logger.info("Successfully generated enhanced panel image")
        return image

    except Exception as e:
        error_msg = f"Error generating panel image: {str(e)}"
        logger.error(error_msg)
        traceback.print_exc()

        # Return error image
        img = Image.new('RGB', (512, 512), color='white')
        d = ImageDraw.Draw(img)
        d.text((10, 10), f"Error: {str(e)}", fill=(0, 0, 0))
        d.text((10, 50), f"Prompt: {scene}", fill=(0, 0, 0))
        return img



# 5. SPEECH BUBBLE IMPLEMENTATION
def add_speech_bubble(image, text, position="top"):
    if not text or text.strip() == "":
        return image  # Skip if no dialogue

    logger.info(f"Adding enhanced speech bubble: {text}")

    try:
        # Convert to PIL if needed
        if not isinstance(image, Image.Image):
            image = Image.fromarray(image)

        # Create a copy to draw on
        img_with_text = image.copy()
        draw = ImageDraw.Draw(img_with_text)

        # Get image dimensions
        width, height = image.size

        # Wrap text for better appearance
        wrapped_text = textwrap.fill(text, width=25)
        lines = wrapped_text.split('\n')

        # Calculate bubble dimensions
        line_height = 20
        text_height = line_height * len(lines)
        text_width = max(len(line) * 10 for line in lines)

        # Position the bubble based on parameter
        margin = 20
        padding = 10
```

```
if position == "top":
    bubble_x = (width - text_width) // 2
    bubble_y = margin
elif position == "bottom":
    bubble_x = (width - text_width) // 2
    bubble_y = height - margin - text_height - 2*padding
elif position == "left":
    bubble_x = margin
    bubble_y = (height - text_height) // 2
elif position == "right":
    bubble_x = width - margin - text_width - 2*padding
    bubble_y = (height - text_height) // 2
else:
    bubble_x = (width - text_width) // 2
    bubble_y = margin

# Make sure bubble is within image bounds
bubble_x = max(margin, min(bubble_x, width - text_width - margin - 2*padding))
bubble_y = max(margin, min(bubble_y, height - text_height - margin - 2*padding))

# Draw comic-style speech bubble with rounded corners
bubble_rect = [
    (bubble_x, bubble_y),
    (bubble_x + text_width + 2*padding, bubble_y + text_height + 2*padding)
]

# Draw white bubble with black outline
draw.rounded_rectangle(
    bubble_rect,
    radius=15,
    fill="white",
    outline="black",
    width=2
)

# Add speech bubble pointer
if position == "top":
    # Triangle pointing down
    pointer_x1 = bubble_x + text_width // 2
    pointer_y1 = bubble_y + text_height + 2*padding
    draw.polygon(
        [(pointer_x1, pointer_y1),
         (pointer_x1 - 15, pointer_y1 - 5),
         (pointer_x1 + 15, pointer_y1 - 5)],
        fill="white",
        outline="black"
    )
elif position == "bottom":
    # Triangle pointing up
    pointer_x1 = bubble_x + text_width // 2
    pointer_y1 = bubble_y
    draw.polygon(
        [(pointer_x1, pointer_y1),
         (pointer_x1 - 15, pointer_y1 + 5),
         (pointer_x1 + 15, pointer_y1 + 5)],
        fill="white",
        outline="black"
    )
```

```
        # Draw text
        try:
            # Try to load a comic font
            font = ImageFont.truetype("arial.ttf", 16)
        except:
            # Fallback to default font
            font = ImageFont.load_default()

        # Draw each line of text
        for i, line in enumerate(lines):
            text_x = bubble_x + padding
            text_y = bubble_y + padding + i * line_height
            draw.text(
                (text_x, text_y),
                line,
                fill="black",
                font=font
            )

        return img_with_text

    except Exception as e:
        logger.error(f"Error adding speech bubble: {str(e)}")
        traceback.print_exc()
        return image  # Return original image if error



# 6. IMPROVED COMIC LAYOUT WITH FRAMES
# create_comic_layout function to display panels in a 2x2 grid

def create_comic_layout(panel_images, panel_data=None, num_cols=2):  # Set default num_cols
    logger.info(f"Creating comic layout with {len(panel_images)} panels in 2x2 grid format"

    try:
        # Handle case with no images
        if not panel_images:
            img = Image.new('RGB', (512, 512), color='white')
            d = ImageDraw.Draw(img)
            d.text((10, 10), "No panels generated", fill=(0, 0, 0))
            return img

        # Set up 2x2 layout
        num_panels = len(panel_images)
        num_cols = min(num_cols, 2)  # Ensure we use a maximum of 2 columns
        num_rows = (num_panels + num_cols - 1) // num_cols  # Calculate required rows (ceil

        # Get dimensions from first panel
        panel_width, panel_height = panel_images[0].size

        # Add margin between panels
        margin = 10
        title_height = 80  # Space for title at top
        story_text_height = 150  # Height allocation for story text below panels

        # Create canvas with navy header and space below for stories
        comic_width = panel_width * num_cols + margin * (num_cols + 1)
        comic_height = (panel_height + story_text_height) * num_rows + title_height + margi
        comic = Image.new('RGB', (comic_width, comic_height), color='white')
    # Add title banner (navy blue)
    draw = ImageDraw.Draw(comic)
    draw.rectangle(
        [(0, 0), (comic_width, title_height)],
        fill="navy"
    )

    # Setup fonts
    try:
        title_font = ImageFont.truetype("arial.ttf", 36)
        story_font = ImageFont.truetype("arial.ttf", 12)
        part_font = ImageFont.truetype("arial.ttf", 14)
    except:
        title_font = ImageFont.load_default()
        story_font = ImageFont.load_default()
        part_font = ImageFont.load_default()

    # Get title from prompt
    title = "Comic Story"
    if panel_data and len(panel_data) > 0:
        # Extract title from first panel's scene
        first_panel = list(panel_data.values())[0]
        title = first_panel.get("scene", "").split("-")[0].strip()

    # Draw title
    draw.text(
        (comic_width // 2, title_height // 2),
        title,
        fill="white",
        font=title_font,
        anchor="mm"
    )

    # Place panels in a 2x2 grid
    for i, img in enumerate(panel_images):
        if i >= num_rows * num_cols:
            break  # Skip if we have more panels than grid positions

        # Calculate row and column position
        row = i // num_cols
        col = i % num_cols

        # Calculate x and y coordinates
        x = margin + col * (panel_width + margin)
        y = title_height + margin + row * (panel_height + story_text_height + margin)

        # Paste panel image
        comic.paste(img, (x, y))

        # Add thin border around panel
        draw.rectangle(
            [(x, y), (x + panel_width - 1, y + panel_height - 1)],
            outline="black",
            width=1
        )

        # Add story text below each panel
        panel_key = f"panel_{i+1}"
        if panel_data and panel_key in panel_data:
            panel_info = panel_data[panel_key]
            # Position for story text below the panel
            text_x = x
            text_y = y + panel_height + 5

            # Calculate max width to keep text aligned with panel above
            max_text_width = panel_width

            # First, add the panel part title in bold
            part_title = panel_info.get("part", f"Panel {i+1}")
            draw.text(
                (text_x, text_y),
                part_title,
                fill="black",
                font=part_font
            )

            # Now add the main story content below the title
            scene_details = panel_info.get("scene", "").split("-")
            if len(scene_details) > 1:
                scene_desc = scene_details[1].strip()
            else:
                scene_desc = "The story continues..."

            # Wrap text to fit panel width
            wrapped_text = textwrap.fill(scene_desc, width=40)

            # Draw the wrapped text below the part title
            draw.multiline_text(
                (text_x, text_y + 20),  # Offset below the part title
                wrapped_text,
                fill="black",
                font=story_font,
                align="left"
            )

    # Add footer with slide number and confidential text
    footer_height = 30
    draw.rectangle(
        [(0, comic_height - footer_height), (comic_width, comic_height)],
        fill="white"
    )

    # Add "Intel Confidential" text on right
    draw.text(
        (comic_width - margin * 2, comic_height - footer_height // 2),
        "Intel Confidential",
        fill="black",
        font=story_font,
        anchor="rm"
    )

    # Add slide number
    draw.text(
        (comic_width - margin, comic_height - footer_height // 2),
        str(num_panels),
        fill="black",
        font=story_font,
        anchor="rm"
    )
```

```
        return comic

    except Exception as e:
        error_msg = f"Error creating comic layout: {str(e)}"
        logger.error(error_msg)
        traceback.print_exc()

        # Return error image
        img = Image.new('RGB', (800, 600), color='white')
        d = ImageDraw.Draw(img)
        d.text((10, 10), f"Error creating layout: {str(e)}", fill=(0, 0, 0))
        return img




# 7. MAIN COMIC GENERATION FUNCTION WITH PROGRESS UPDATES
def generate_comic(prompt, num_panels=4, style="comic book", progress=None):
    logger.info(f"Starting enhanced comic generation for prompt: '{prompt}'")

    try:
        # Update progress
        if progress:
            progress(0.1, "Initializing models...")

        # Setup models
        tokenizer, text_model = setup_story_model()
        if progress:
            progress(0.2, "Text model loaded")

        image_pipe = setup_image_model()
        if progress:
            progress(0.3, "Image model loaded")

        # Generate structured story
        story_data = generate_story(prompt, num_panels, tokenizer, text_model)
        if progress:
            progress(0.4, "4-part narrative story generated")

        # Generate panel images
        panel_images = []
        for i, (panel_key, panel_data) in enumerate(story_data.items()):
            if progress:
                progress_val = 0.4 + (0.5 * i / len(story_data))
                progress(progress_val, f"Generating panel {i+1}/{len(story_data)}")

            # Generate image for panel
            panel_image = generate_panel_image(panel_data, style, image_pipe)

            # Add dialogue if present
            if panel_data.get("dialogue"):
                # Position the speech bubble based on the panel number
                position = "top" if i % 2 == 0 else "bottom"
                panel_image = add_speech_bubble(panel_image, panel_data["dialogue"], positi

            panel_images.append(panel_image)

        # Create comic layout
        if progress:
            progress(0.9, "Assembling enhanced comic layout")
        comic = create_comic_layout(panel_images, story_data)

        if progress:
            progress(1.0, "Comic generated!")

        logger.info("Enhanced comic generation complete")
        return comic, story_data

    except Exception as e:
        error_msg = f"Error in comic generation: {str(e)}"
        logger.error(error_msg)
        traceback.print_exc()

        # Create error image
        img = Image.new('RGB', (800, 600), color='white')
        d = ImageDraw.Draw(img)
        d.text((10, 10), "Error generating comic:", fill=(255, 0, 0))
        d.text((10, 40), str(e), fill=(0, 0, 0))
        d.text((10, 70), f"Prompt: {prompt}", fill=(0, 0, 0))
        d.text((10, 100), "Please try again with a simpler prompt or fewer panels", fill=(0

        # Empty story data
        error_data = {"error": str(e)}

        return img, error_data


# 8. IMPROVED GRADIO UI
def create_ui():
    # Status for progress updates
    status_text = gr.State("")

    def generate_with_progress(prompt, num_panels, style, progress=gr.Progress()):
        def update_progress(progress_val, status):
            progress(progress_val, desc=status)
            return status

        try:
            comic, story_data = generate_comic(
                prompt=prompt,
                num_panels=int(num_panels),
                style=style,
                progress=update_progress
            )
            return comic, json.dumps(story_data, indent=2), "Complete"
        except Exception as e:
            error_msg = f"Error: {str(e)}"
            logger.error(error_msg)
            traceback.print_exc()
            return None, error_msg, "Error"

    # Define interface
    with gr.Blocks(theme=gr.themes.Soft()) as demo:
        gr.Markdown("# 🦸 ComicCrafter AI - Generate 4-Part Narrative Comics")

        with gr.Row():
            with gr.Column(scale=1):
                prompt_input = gr.Textbox(
                    label="Story Prompt",
                    placeholder="A superhero fights a giant robot in the city",
                    lines=3,
                    value="A superhero with a red cape fights a giant robot in a city"
                )

                with gr.Row():
                    num_panels = gr.Slider(
                        minimum=1,
                        maximum=4,
                        value=4,
                        step=1,
                        label="Number of Panels"
                    )

                    style = gr.Dropdown(
                        choices=["comic book", "manga", "cartoon", "sketch", "pixel art"],
                        value="comic book",
                        label="Art Style"
                    )

                # Add examples
                gr.Examples(
                    examples=[
                        ["A superhero with laser vision defends a city from an alien invasi
                        ["A detective solves a mystery in a rainy city at night", 4, "noir"
                        ["A robot learns to make friends with humans", 4, "cartoon"],
                        ["A ninja warrior battles a dragon in ancient Japan", 4, "manga"]
                    ],
                    inputs=[prompt_input, num_panels, style]
                )

                generate_btn = gr.Button("Generate Comic", variant="primary")
                status = gr.Textbox(label="Status", value="Ready")

                gr.Markdown("""
                ## 📝 How It Works

                This AI comic generator creates a **4-part narrative structure**:

                1. **INTRODUCTION**: Sets the scene and characters
                2. **STORYLINE**: Develops the plot with challenges
                3. **CLIMAX**: The most intense moment of conflict
                4. **RESOLUTION**: Concludes with a lesson or message

                Each image is carefully generated to match its place in the story!
                """)

            with gr.Column(scale=2):
                output_image = gr.Image(label="Generated Comic")
                output_story = gr.Textbox(label="Story Structure", lines=10)

        generate_btn.click(
            generate_with_progress,
            inputs=[prompt_input, num_panels, style],
            outputs=[output_image, output_story, status]
        )

        gr.Markdown("""
        ## 💡 Tips for Success
        - Use descriptive prompts with clear characters and action
```

```
        - Try different art styles to match your story's mood
        - For best results, use the full 4-panel story structure
        - Comics follow classic narrative patterns: introduction → conflict → climax → re
        """)

    return demo

# Run the app
if __name__ == "__main__":
    print("Starting ComicCrafter AI - Enhanced Version...")
    demo = create_ui()
    demo.launch(debug=True, share=True)
```

```
Starting ComicCrafter AI - Enhanced Version...
/usr/local/lib/python3.11/dist-packages/gradio/components/dropdown.py:227: UserW
  warnings.warn(
Colab notebook detected. This cell will run indefinitely so that you can see err
* Running on public URL: https://8026abff4bf48297c3.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades
```

gradio

No interface is running right now

```
Keyboard interruption in main thread... closing server.
Killing tunnel 127.0.0.1:7860 <> https://8026abff4bf48297c3.gradio.live
```