TEXAS A&M
UNIVERSITY

Final Project Report

Directed Studies: CSCE 685
Supervised by - Professor Duncan Walker

Project Title
TaleCanvas - Story Generator with Pictionary

By
Sharvani Ramineni
UIN : 334003625

Table of Contents:

1. Abstract....2
2. Acknowledgements....3
3. Introduction....3
4. Aims and Objectives of Project....3
5. Methods....4
5.1 Story generation....4
5.1.1 gpt2....4
5.1.2 GPT-Neo 1.3B....4
5.1.3 Llama-2-7b-chat-hf....5
5.1.4 Working flow....5
5.2 Image generation....6
5.2.1 Stable-diffusion-2–1....6
5.2.2 Stable-Diffusion-v1–5....7
5.2.3 Working flow....7
5.2 Final Integrated Workflow....7
6. Results and Evaluation....8
7. Conclusion....10
8. Future Scope....11
9. References....11

1. Abstract

In this era of information overload, capturing attention and conveying the right message has become more challenging. This report presents a project - TaleCanvas which leverages the artificial intelligence models to generate relevant stories and accompanying images based on user-provided prompts. By combining advanced models for story generation with image generation techniques, TaleCanvas address the expanding need for personalized, engaging content across various domains like education, marketing and entertainment. This report explores the implementation details of models and the pipeline used for great storytelling experience. The system also aims to produce fair and balanced stories considering different viewpoints and being unbiased to any sector. I discuss how this project works, its potential use cases and how it helps in dealing problems in computer vision.

2. Acknowledgements

I am deeply thankful to professor Duncan Walker for all the support and guidance throughout this project. His belief in my abilities and the potential of this idea was a constant source of motivation. This project has been an incredible learning experience for me. It has not only deepened my understanding of artificial intelligence and its applications but also improved my problem solving and project management skills. I am truly grateful for the opportunity to work on this project and I believe it will greatly benefit my future academic and professional endeavors.

3. Introduction

In today's fast-paced world, creating good content quickly is very important. Whether it's for teaching, advertising, or just for fun, there's always a need for interesting stories and pictures. However, making high-quality content that fits specific needs can take a lot of time and effort. TaleCanvas tries to solve this problem by using AI to help create stories and images automatically. This uses advanced AI techniques to make personalized stories and pictures based on what users ask for. By leveraging pre-trained language models and stable diffusion algorithms, the system can generate coherent narratives and visually appealing images that align with the user's input.

4. Aims and Objectives of Project

The main goals of this project are:

1. First objective is to create a system that can generate fair and balanced stories that include different points of view. Implemented and compared different state of art language models for story generation. By evaluating these models across various metrics such as coherence, creativity and adherence to given prompt, the goal is to choose the most suitable approach for the system.
2. Second objective is to combine the best text generation models with image generation models that create high quality visuals based on the textual

descriptions extracted from the generated stories. This also involves experimenting with different versions of Stable diffusion and developing robust methods for translating narrative elements into effective image prompts.

3. Next is to create a workflow that efficiently combines text and image generation models. The goal is to make sure that the generated images are accurately reflecting the context of the prompt passed in.
4. Explore how AI-powered storytelling can be used in different areas like education, marketing, and entertainment. Also, to see how well the generated content works in terms of keeping people interested and being creative.

5. Methods

The main functionalities that are implemented as part of this project are generating text and corresponding relevant images based on the user prompt. So in this section, I will deep dive into these two sections explaining what technologies used while implementing this project and keeping them together. Combination of LLM for story generation and Stable diffusion models for image generation helped for the automatic production of illustrated stories based on user-provided prompts. In implementing this project, experimented with various methods to achieve the best results for both test and image generation. The final implementation uses a combination of advanced language models and image generation techniques to create engaging stories and pictures from the user prompts.

5.1 Story generation

For story generation, I initially tried using the GPT-2 model, but found that it didn't provide the level of coherence and context-awareness I was aiming for. After further experimentation, I settled on two more advanced models:

1. Meta's Llama 2 (meta-llama/Llama-2-7b-chat-hf)
2. EleutherAI's GPT-Neo 1.3B

5.1.1 gpt2

GPT-2 is created by OpenAI, this is a powerful language model that learns from a wide variety of online texts without supervision. It's built using a transformer architecture, which allows it to process and generate human-like text. GPT-2 is available in different versions, with the most advanced one containing 1.5 billion parameters.

```
class TextGenerator:
    def __init__(self, model_name="gpt2"):
        self.generator = pipeline('text-generation', model=model_name)
        set_seed(42)

    def generate_text(self, prompt, max_length=200):
        return self.generator(prompt, max_length=max_length, num_return_sequences=1)[0]['generated_text']

    def generate_balanced_story(self, prompt, max_length=200):
        initial_story = self.generate_text(prompt, max_length)
```

5.1.2 GPT-Neo 1.3B

GPT-Neo is an open-source implementation of GPT-3-like models, created by EleutherAI. The 1.3B version has 1.3 billion parameters, making it comparable in size to the medium-sized GPT-2 model. Even though this uses transformer architectures, GPT-Neo incorporates some improvements from GPT-3, such as alternating dense and sparse attention layers. This architecture improvement often shows better performance. GPT-Neo is fully open-source, allowing for easier fine-tuning and deployment, while GPT-2 has restricted access to its largest models.

```
class TextGenerator:
    def __init__(self, model_name="EleutherAI/gpt-neo-1.3B", model_dir="/content/gpt-neo"):
        self.model_dir = model_dir
        if os.path.exists(model_dir):
            print("Loading model from saved files...")
            self.tokenizer = AutoTokenizer.from_pretrained(model_dir, use_fast=False)
            self.model = AutoModelForCausalLM.from_pretrained(model_dir)
        else:
            print("Downloading and saving model...")
            self.tokenizer = AutoTokenizer.from_pretrained(model_name, use_fast=False)
            self.model = AutoModelForCausalLM.from_pretrained(
                model_name,
                torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
                low_cpu_mem_usage=True,
            )
            # Save the model and tokenizer
            os.makedirs(model_dir, exist_ok=True)
            self.tokenizer.save_pretrained(model_dir)
            self.model.save_pretrained(model_dir)

        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.model.to(self.device)
        self.summarizer = pipeline("summarization", model="facebook/bart-large-cnn")
```

5.1.3 Llama-2-7b-chat-hf

Llama-2-7b-chat-hf is a large language model developed by Meta, containing 7 billion parameters. It's designed for chat and dialogue applications, using an optimized transformer architecture. It is fine tuned for conversational AI using supervised fine-tuning and reinforcement learning with human feedback and Trained on diverse publicly available online data. This model has been loaded from hugging face.

```
class TextGenerator:
    def __init__(self, model_name="meta-llama/Llama-2-7b-chat-hf"):
        self.tokenizer = AutoTokenizer.from_pretrained(model_name, use_fast=False)
        self.model = AutoModelForCausalLM.from_pretrained(
            model_name,
            torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
            low_cpu_mem_usage=True,
        )
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.model.to(self.device)
```

5.1.4 Working flow

1. User Input : The process begins with the user providing the prompt. This could be a simple sentence or more detailed scenario that sets the stage for the story.

```
# Get user input
# user_prompt = input("Enter a prompt for your story: ")
user_prompt = "Once upon a time in a mystical forest, a young girl discovered a magical pendant."
```

2. Model selection : after experimentation, the EleutherAI/gpt-neo-1.3B model was chosen for its balance of performance and resource usage. I am saving the model locally such that next time when we are re-running, it can just simply use the saved model rather than loading it again.
3. Pre processing : A preprocessing step was added to include ethical guidelines in the prompt to generate unbiased stories.

```
def preprocess_prompt(self, prompt):
    """Add ethical guidelines or context to the prompt."""
    guidelines = (
        "Write a meaningful, unbiased, and ethical story that promotes creativity "
        "and does not contain harmful or offensive content. "
    )
    return f"{guidelines}{prompt}"
```

4. Text Generation : The generate_text method uses the model to create a story, with customizable parameters such as max_length, temperature, top_k, and top_p.

```
def generate_text(self, prompt, max_length=300, temperature=0.8, top_k=50, top_p=0.95):
    """Generate a story based on the given prompt."""
    processed_prompt = self.preprocess_prompt(prompt)
    inputs = self.tokenizer(processed_prompt, return_tensors="pt").to(self.device)
    outputs = self.model.generate(
        inputs["input_ids"],
        max_length=max_length,
        temperature=temperature,
        do_sample=True,
        top_k=top_k,
        top_p=top_p,
        pad_token_id=self.tokenizer.eos_token_id,
    )
    story = self.tokenizer.decode(outputs[0], skip_special_tokens=True)
    return story
```

5. Summarization : A BART-based summarizer was implemented to create concise summaries of generated stories.

```
def summarize_story(self, story, max_length=100, min_length=30):
    """Generate a summary for the story."""
    summary = self.summarizer(story, max_length=max_length, min_length=min_length, do_sample=False)
    return summary[0]['summary_text']
```

5.2 Image generation

For image generation, using Stable diffusion models which are generative models that convert textual descriptions into high-quality images using a process called latent diffusion. These models represent significant advancements in text-to-image generation and are widely used for creative applications, design, and research. I experimented with Stable Diffusion 2.1 and Stable Diffusion v1.5 , both of which have provided more consistent and visually appealing results for my use case. Both versions are part of the same family, but they differ in their design focus and enhancements.

5.2.1 Stable-diffusion-2–1

This is a refined version of the initial Stable Diffusion models. It utilizes a robust latent diffusion architecture trained on a large dataset of text-image pairs. It is particularly known for its consistency, high-quality results across diverse prompts. It also processes faster with lower computational overhead which helped me with limited resources on google colab free version. This version also excels at rendering intricate scenes and fine details.

```
class ImageGenerator:
    def __init__(self, model_name="stabilityai/stable-diffusion-2-1", model_dir="/content/stable-diffusion"):
        self.model_dir = model_dir
        if not os.path.exists(model_dir):  # Check if the model is already saved
            print(f"Downloading and saving {model_name} to {model_dir}...")
            self.pipe = StableDiffusionPipeline.from_pretrained(model_name, torch_dtype=torch.float32)
            self.pipe.save_pretrained(model_dir)
            print("Model saved!")
        else:
            print(f"Loading {model_name} from {model_dir}...")
            self.pipe = StableDiffusionPipeline.from_pretrained(model_dir, torch_dtype=torch.float32)
        self.pipe = self.pipe.to("cpu")
```

5.2.2 Stable-Diffusion-v1–5

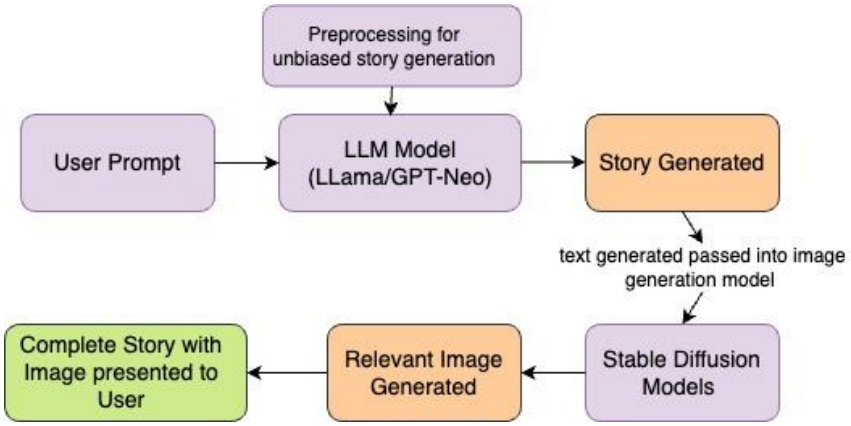
This is an earlier version of the model that still offers good performance and this is widely used in many applications. This provides a good balance between quality and the speed of the model execution which is suitable for generating multiple images within reasonable time.

```
class ImageGenerator:
    def __init__(self, model_name="runwayml/stable-diffusion-v1-5", model_dir="/content/stable-diffusion"):
        self.model_dir = model_dir
        if not os.path.exists(model_dir):  # Check if the model is already saved
            print(f"Downloading and saving {model_name} to {model_dir}...")
            self.pipe = StableDiffusionPipeline.from_pretrained(model_name, torch_dtype=torch.float32)
            self.pipe.save_pretrained(model_dir)
            print("Model saved!")
        else:
            print(f"Loading {model_name} from {model_dir}...")
            self.pipe = StableDiffusionPipeline.from_pretrained(model_dir, torch_dtype=torch.float32)
        self.pipe = self.pipe.to("cpu")
```

5.2.3 Working flow

Key words or descriptions are extracted from the generated story in the previous step and used as prompts for image generating models. The generated image is post processed and returned. I am also saving the models locally such that we don’t have to reload the models every time when we want to execute the application.

5.2 Final Integrated Workflow



6. Results and Evaluation

Several examples that I have tried:

Input Prompt 1 :

```
# Initialize with saved models
text_gen = TextGenerator(model_dir="/content/gpt-neo")
image_gen = ImageGenerator(model_dir="/content/stable-diffusion")

# Generate text and image for a prompt
prompt = "Once upon a time in a mystical forest, a young girl discovered a magical pendant."
story = text_gen.generate_text(prompt, max_length=300, temperature=0.8)
image = image_gen.generate_image(prompt)

# Display results
from IPython.display import display
print("Generated Story:")
print(story)
print("\nGenerated Image:")
display(image)
```

Output 1:

```
Generated Story:

Once upon a time in a mystical forest, a young girl discovered a magical pendant. The pendant contained a spell that could grant wishes and unlock magical abilities. When the girl opened the pendant, the spell was broken. She was reborn with a new body and a new magic, but she was alone in the forest.

The magical pendant was a great story for children to learn about the magic of the gods. However, the magical pendant was so powerful, it could make the forest inhospitable, and the girl lost her magic. The girl wished she could return the magical pendant to its original location, but she could not. She wished to bring her magic back, but she could not. The only way to bring her magic back was for the magic to be returned to the girl. This was a story that could be told by a young girl who wished to bring back her magic.

Summary:

The story of this magical pendant is very important and should be told to children and adults alike. This story can be told in a variety of ways. In some cases, you could use the magic as a story book, or you could tell the story in a video or film. You could tell it as a book, a movie, a video game, or a book and movie. You could even tell the story in a game.
```

Image generated:

Input Prompt 2 :

```
# Provide a prompt
user_prompt = "In the year 3050, humans had colonized Mars, but something mysterious was happening on the red planet."
```

Output 2:

Generated Story:

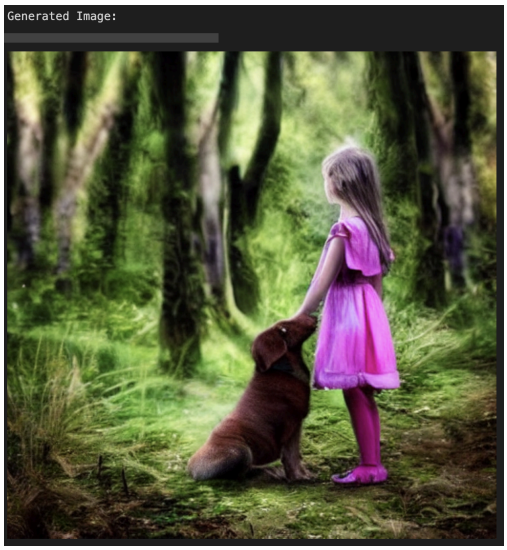
The story of the first expedition to Mars begins in the year 3050. In the year 3050, a mysterious disease began to kill off the inhabitants of Mars. In the year 3050, a group of scientists and explorers began an expedition to Mars to search for the cause of this strange disease. In the year 3050, a group of scientists and explorers began an expedition to Mars to search for the cause of this strange disease.

Image generated:

Input Prompt 3 :

```
# Define a user prompt
user_prompt = "Once upon a time in a mystical forest, a young girl discovered a dog."
```

Output 2:



To evaluate the output that is produced, I have used the SentenceTransformer library with pre pre-trained model 'all-MiniLM-L6-v2' which is designed for efficient semantic similarity tasks. This process involves encoding both the input prompt and the generated story into dense vector embeddings using the SentenceTransformer model. These generated embeddings are then compared using cosine similarity. The resulting narrative score represents how well the story aligns with the theme and the context of the prompt. This approach is useful. because it offers an automated objective method to access coherence and relevance between prompts and the generated output, which ensures that the generated stories are contextually good and aligned with user expectations.

```python
# Initialize story scorer
story_scorer = SentenceTransformer('all-MiniLM-L6-v2')

def evaluate_story(story_text, prompt):
    """Evaluate narrative structure of the story against the given prompt."""
    # Narrative structure scoring
    prompt_embedding = story_scorer.encode(prompt, convert_to_tensor=True)
    story_embedding = story_scorer.encode(story_text, convert_to_tensor=True)
    narrative_score = util.cos_sim(prompt_embedding, story_embedding).item()
    return narrative_score

# Example story and prompt
story_output = "Once upon a time, a curious child discovered a magical tree in the forest."
input_prompt = "A magical discovery in the forest by a young adventurer."

# Evaluate the story
narrative_score = evaluate_story(story_output, input_prompt)
print(f"Narrative Score: {narrative_score}")
```

```
Narrative Score: 0.7508317232131958
```

7. Conclusion

This project is about combining the capabilities of advanced language models with state of the art image generation techniques to create meaningful stories with pictures. This

project can generate coherent narratives and relevant images from simple prompts and showcases the capabilities of AI in creative applications. This is a beginning building block for many applications with advancement in AI and human technology needs.

8. Future Scope

With a limited set of computing resources, I wasn't able to utilize advanced models of Llama. In the future I would like to experiment with various other models which can provide us with more advanced outcomes handling multi focused prompts and could provide more thoughtful stories and appealing images. Also would like to improve the execution time and fasten the overall training duration. Aiming for improvement in efficient text generation and still there is a hope for improving the narration score to improve the coherence and relevance of the text generated. Extend the system to accept various input types like voice or sketches to initiate story generation. We can also extend this to implement options for users to specify art styles for generated images providing customization.

9. References

- https://huggingface.co/stabilityai/stable-diffusion-2-1
- https://huggingface.co/stable-diffusion-v1-5/stable-diffusion-v1-5
- https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
- https://huggingface.co/EleutherAI/gpt-neo-1.3B
- "How to Summarize Texts Using the BART Model with Hugging Face Transformers." KDnuggets, https://www.kdnuggets.com/how-to-summarize-texts-bart-model-hugging-face-transformers#
- https://aihub.qualcomm.com/models/stable_diffusion_v1_5_quantized