Art and Machine Learning
CMU 2022 Spring
Final Project

Daydream Simulator

```
--> You wake up in water and you feel strange things.
--> What is it?
--> What are you doing?
--> What is your purpose in life?
--> Where are you?
--> In the moment you become aware that your purpose is
to find out about the circumstances that are driving
you through this journey.
```

Group 15
Elena Gong (yezheng), Zixuan Zou (zixuanz)
Carnegie Institute of Technology
Master in Electrical & Computer Engineering
(Background applies to both members)

DESCRIPTION

Concept

People usually find their dreams illogical, yet have some relevance to their real lives. These dreams are full of imaginations, weirdness, and creativity. However, most people would forget their dreams when they wake up due to neurochemical conditions in the brain. We feel that it is a great loss to forget these colorful dreams because we might find many new ideas and inspirations for art works from them. Inspired by a popular series of Tik Tok videos which shoot stories like dreams created by Savanah Moss, we want to build some applications that users would be able to explore their own daydreams with full consciousness. Therefore, we decided to create a text based adventure game that allows players to create their own daydream story with the help of AI and visualize their daydreams with images.

Technique

GPT-2: Generative Pre-trained Transformer 2 (GPT-2) is an open-source artificial intelligence created by OpenAI in 2019. It can translate text, answer questions, summarize passages, and generate text output. We used GPT-2 genre-based story generator [1] to generate text based on the user's text input.

DALL-E mini: DALL-E is an artificial intelligence program that creates images from textual descriptions. It uses the GPT-3 Transformer model to interpret natural language inputs and generate corresponding images. We used DALL-E Mini [2] to generate images based on the GPT-2 output.

spaCy: spaCy is an open-source software library for advanced natural language processing. We used it to determine the parts of speech for words in the GPT-2 generated text.

gTTS: Google Text-to-Speech, a Python library and CLI tool to interface with Google Translate's text-to-speech API. In our project, we used this API to read out the generated story to provide players with a more immersive experience.

Streamlit: Streamlit is an open source app framework in Python language[3]. It helps us create web apps for data science and machine learning in a short time. In our project we used Streamlit to build the web application for our text based adventure game.

Flask: Flask is another API of Python that allows us to build up web-applications[4]. Flask's framework has less base code to implement a simple web-Application and we used Flask to run our backend for Dall-E on Colab to speed up the application with GPU.

Ngrok: Ngrok is a useful utility to create secure tunnels to locally hosted applications using a reverse proxy[5]. It is a utility to expose any locally hosted application over the web. In our project we used Ngrok to provide a publicly accessible web URL to our locally hosted game application.

Process

GPT-2

We used GPT-2 genre-based story generator to generate the text in our game based on the user's input. We have tried many variants of the GPT-2 model, and this one worked the best for our project. It can generate stories based on user input and the selected genre, such as thriller, superhero, science fiction, and drama, etc. After trying each one of the genres, we decided to use "horror" as the genre for our project. Even though some of its output is weird or scary, most of the time the text it generates fits the overall daydreaming vibe very well.

We also implemented some filtering in our code to remove some results that might be too disturbing or creepy, and to remove the sentences that do not make much sense. We would regenerate the output text if some specific words or characters appeared in the generated text.

DALL-E Mini

We used DALL-E Mini to generate images based on the sentences generated by GPT-2. We have also experimented with Wombo AI, and found the DALL-E Mini can generate images that are more diverse and accurate. Since DALL-E and DALL-E 2 are not released to the public, we found DALL-E Mini, which could also generate beautiful images from text. We had lots of difficulties when first trying to run DALL-E Mini and spent lots of time adjusting it for our project. It also requires a GPU to run the generator, and eventually we got it working relatively stably by purchasing colab pro.

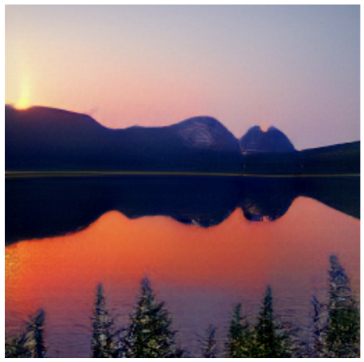
We decided to use the text generated by GPT-2, instead of the player input, as the prompt for DALL-E Mini because we want the image to be more relevant to the story generated. It could also surprise the players by generating images that have more than what they inputted.

CLIP

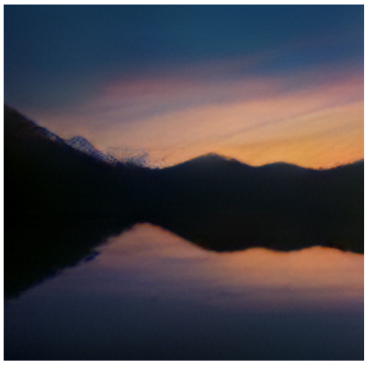
At the beginning, we would generate multiple images using DALL-E Mini and use CLIP, a pre-trained model used for robust automatic evaluation of image captioning, to evaluate the image-text compatibility for the results, and would select the image with the highest score to be displayed in our game.

However, after some testing we found that it took some time to generate multiple images and compute the scores for each one of them. To reduce the runtime and provide the player with a better game experience, we compared the results and the CLIP scores. We found that most of the time, all images generated by DALL-E Mini can accurately reflect the input prompt. The difference between CLIP scores is not big, and the quality of images are close to each other. Thus, we decided to generate one image using DALL-E Mini each time, and use that image for our game directly, which shortened the run time a lot.

Below are images generated by DALL-E Mini with the prompt of "sunset over a lake in the mountains". As we can see, all of the images depict the scene accurately, and their CLIP scores are very close to each other.

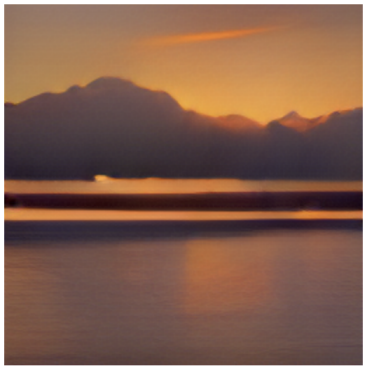

Score: 28.90


Score: 27.56


Score: 27.34


Score: 26.27


Score: 24.23


Score: 23.99

spaCy

We used spaCy to label the parts of speech for all of the words in the sentences generated by GPT-2. We store the nouns, verbs, and adjectives for the results and use them to form the ending of the game.

gTTS

After implementing the main functionalities of generating story and images with the player provided prompts, we found that it might be more immersive and closer to a real game if we could read out the generated stories. Our goal is to provide a low-latency human-like voice option for the players if they find that the generated story's font is too small.

We first attempted to reuse the YourTTS model, which was one of the tools that we used to synthesize singing in our Project 3. However, one disadvantage was that while it provides high quality human voice synthesis, it requires human voice input as a sample. Thus, we decided to switch to gTTS, which provides very low-latency standardized voice output.

AWS

To deploy our game as a web application, we first tried to run the DALL-E Mini model on AWS as our backend, and use HTTP requests to send text prompts and receive images to and from AWS. We tried to deploy the model using EC2, SageMaker, CloudFormation, and unfortunately they all failed. Fortunately we found other methods to run the backend for our game.

Streamlit

In our proposal, we planned to build our project with Django Framework as backend server and React.js as front end. However, one problem that we ran into was that our backend needs to support running pretrained ML models with GPU, and it is very difficult to do so with Django and React, which has limited support running ML models.

We then found Streamlit, which is specifically designed for building and sharing data apps. We decided to move our application to Streamlit, and in general our game includes three types of pages: game start page, in game page, and game end page. In the game start page we introduced what experience the players would be able to expect during the game, and some instructions and warnings that players need to read before starting the game. In the game play page, the players would be able to input any prompt that may connect the previously generated story or simply any prompts that come to their minds, and they would be able to listen to the story generated with their input prompts and images that reflects the story content within several seconds.

One problem that we encountered when merging the pre-trained models with Streamlit is that Streamlit unfortunately does not support GPU. While we were able to call and use the pretrained models for generating story, we were not able to do the same thing for Dall-E mini. In the following section, we would explain how we managed to solve this problem with Flask.

Flask

As described in the previous section, one issue that we faced when we were merging the image and story generation models into our web application was that Streamlit does not support GPU, which severely harmed the latency of our game, and made it necessary for us to find some other backend servers to run the models and accept the requests from Streamlit applications.

As a solution, we found Flask, a popular Python framework for developing web applications. We chose this framework because it comes with minimal built-in functionalities and requirements, making it simple to get started and flexible to use. For our purpose, we need a simple framework that supports receiving requests from our application, running the models on GPU, and sending responses with generated images back to the client. Flask satisfies all the requirements and requires minimal decorations for the web application, therefore we decided to use Flask to build our backend server.

Ngrok

With Streamlit and Flask, we were able to build the web application's backend and frontend locally. However, in order to connect them together, and possibly allow users to play the game easily with a Website URL, we need to deploy the web application to the cloud. With previous attempts with AWS, we noticed that it is very difficult to deploy backend and frontend separately and communicate with HTTP requests.

Fortunately, we found a simple tool that is compatible with Colab, Ngrok, which is the programmable network edge that needs no code changes to our application. It is very important for the backend to have cloud deployment because the front end will need its URL to send requests with generated text prompts. On Colab, we found a Python library called flask-ngrok, which deploys our backend to a public URL with

one line of command. In terms of the frontend side, we found that running the Streamlit application locally allows user to input the URL of backend in the command line in an easier way, thus we eventually decide not to use Ngrok in our frontend, and instead users would be able to download our main file and run with simple instructions.

Reflection

Result selection
We have compared results from different variants of the GPT-2 model, and selected the GPT-2 genre-based story generator for our project, with the genre "horror". The text it generates makes the most sense and does feel like actual stories compared to other models.
We chose DALL-E for image generation because the quality of the images generated is very high. We used CLIP to compute image-text compatibility scores for the images generated and the text prompt as discussed in the process section.

Playtesting and surveys
Since we are building a text based adventure game, and we are trying to provide as much freedom as possible to the players when they decide the next steps, it was initially very difficult for us to come up with quantifiable criteria to evaluate our result. After much thought and discussion, we decided to evaluate our final results from the following two methods: First, the smoothness of the game experience: is the waiting time for each step too long for the players? Second, the satisfaction level for the entire game experience: are the players satisfied with the game? Would they want to play it again?

In order to evaluate the smoothness of the game, we recorded the average time of one step's generation time, including generating the story, image and the voice, and we get an average of 7.5 seconds among 20 trials. The time might be longer in the case that the story generator regenerates the story under some circumstances. We also added a question in our user survey, asking them to evaluate the smoothness of the game, with a scale from 1 to 5. Among the 13 individual responses, we got an average score of 4.3, and most of the responses that didn't give a score of 5 met the case that the story generator regenerated several times and increased latency.

In terms of the satisfaction level, we added a question in our user survey, asking them to evaluate their satisfaction level of the game, with a scale from 1 to 5. Among the 13 responses, we got an average score of 4.6, and in general people are willing to try this game again.

From the critics above, we concluded that our game provides a satisfying experience for the users, and our future goal would be further improving the latency of content generation in our game.

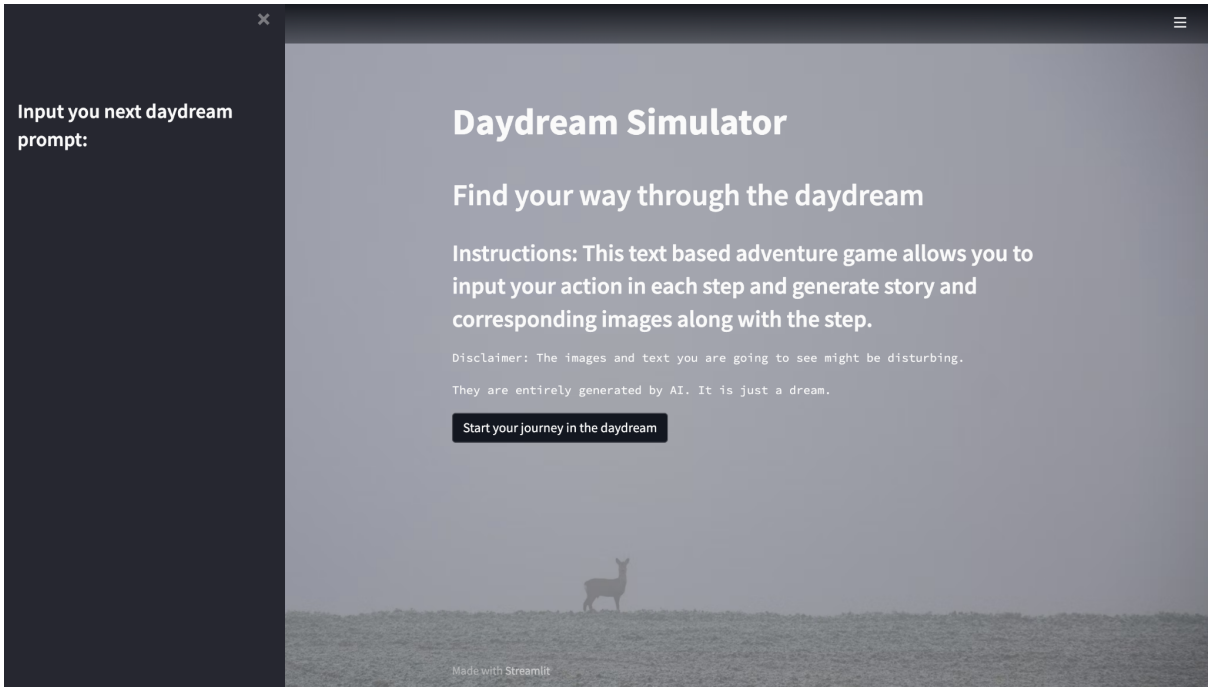
New ideas
We are very satisfied with our results, though we do believe the project has much potential to become something even better.
We could add a more cohesive storyline to the game, so players would have a more immersive experience which is more tightly related to the actions they take. We could also implement a system that keeps track of all the past player actions and story generated, and add more possibilities for the game endings.

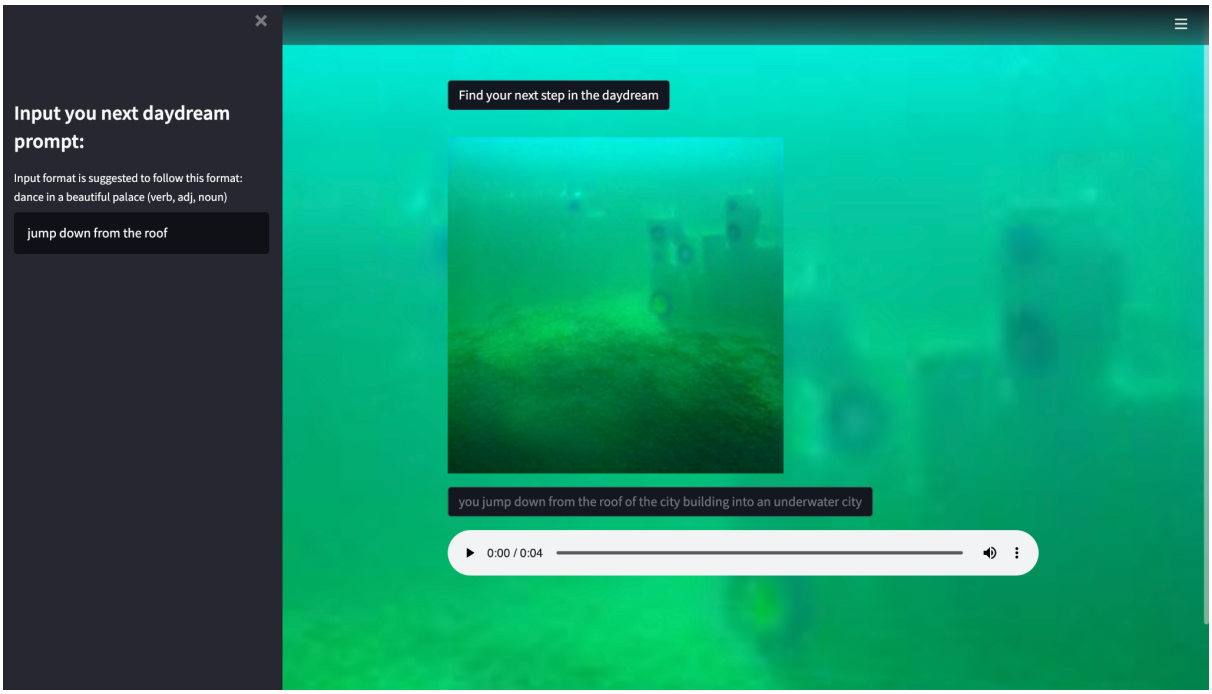
Lessons learned
In our entire process of implementation, we experienced most difficulties in merging the ML models to light-weight and low-latency web applications. And among all methods, we spent about three days figuring out how to run our models on AWS. It turned out that AWS is very complicated, and we finally found our solutions in Colab with Ngrok and Flask. Therefore, one lesson learned is to try multiple approaches before we make it work, which might save us much more time. Another challenge we encountered was to make sure that the AI generated results are comparatively controlled, as the results are very random and stories could go wild.

RESULT

Video demo
https://drive.google.com/file/d/1zn6ouyoQfg7KIxlYZ34zzASNNY1jaYAy/view?usp=sharing

Google drive folder
https://drive.google.com/drive/folders/1B_brYEgg-Yg93a7tBUq8hXHDCMAJr2Ho?usp=sharing





CODE

dalle_backend.ipynb
https://colab.research.google.com/drive/1lna6112aZ4ETU2h4Bkfg4r-Vjb1DOVYx?usp=sharing

main.py
https://drive.google.com/file/d/1Ebp1lKup5V06r3SMizs-zNALsQGkhTpn/view?usp=sharing

Backend
Run dalle_backend.ipynb on Google Colab (GPU is required)
Replace the ngrok authentication token with your own
Enter your wandb api key when prompted
Copy your ngrok url (the second one in the output) for next step

Streamlit app
Install the following packages in your environment

```
pip install datasets transformers
pip install gTTS
pip install streamlit
```

Then run main.py with the ngrok url

```
streamlit run main.py <your-ngrok-url>
```

REFERENCE

[1] https://github.com/pranavpsv/Genre-Based-Story-Generator

[2] https://github.com/borisdayma/dalle-mini

[3] https://streamlit.io/

[4] https://flask.palletsprojects.com/en/2.1.x/

[5] https://ngrok.com/