Multifunctional NLP and Image Generation Tool using Hugging Face Models

Overview

This project is a multifunctional AI tool built using **Streamlit** and **Hugging Face models**, providing multiple features such as:

- **Text Summarization**
- **Next Word Prediction**
- **Story Generation**
- **Chatbot Interaction**
- **Sentiment Analysis**
- **Question Answering**
- **Image Generation**

It leverages **transformers** from Hugging Face, enabling the use of pre-trained models for NLP and image generation tasks, powered by **PyTorch** and **Stable Diffusion**.

Installation Instructions

1. Install necessary libraries

You need to install the following libraries to run the project:

bash

Copy code

!pip install torch torchvision transformers streamlit diffusers

2. Running the Streamlit App

Once the required libraries are installed, the app can be run using the following command in the terminal:

bash

Copy code

!python -m streamlit run Hugging_face.py

Code Breakdown

Import Required Libraries

python

Copy code

import torch

from transformers import BartForConditionalGeneration, BartTokenizer

from transformers import GPT2LMHeadModel, GPT2Tokenizer

```
from transformers import pipeline
from transformers import AutoTokenizer, AutoModelForCausalLM
from transformers import AutoTokenizer, AutoModelForQuestionAnswering
from diffusers import StableDiffusionPipeline
import streamlit as st
import re
import matplotlib.pyplot as plt
from PIL import Image
import os
```

- **torch**: The core PyTorch library, used for model operations and running computations on GPU or CPU.
- **transformers**: Hugging Face library to access pre-trained NLP models like BART, GPT-2, DialoGPT, and BERT.
- **diffusers**: For image generation, specifically the **Stable Diffusion** model.
- **streamlit**: Web framework for building interactive UIs in Python.
- **re**: Regular expressions, used in text processing.
- **matplotlib.pyplot** & **PIL**: For image handling and visualization.

Checking CUDA Availability

```
python
Copy code
device = "cuda" if torch.cuda.is_available() else "cpu"
if not torch.cuda.is_available():
    st.warning("CUDA not available, running on CPU. Consider installing `accelerate` for optimized performance on CPU:\n`pip install accelerate`")
```

- This section checks if a CUDA-capable GPU is available and sets the device accordingly. If not, it warns users to consider optimizing performance using the **accelerate** library.

Streamlit App Title and Description

```
python
Copy code
st.title('🌟 Multifunctional NLP and Image Generation Tool using Hugging Face Models')
st.write(''' ... ''')
```

- Sets the title and introductory description of the app in the Streamlit sidebar.

Sidebar for Task Selection

python

Copy code

```
task = st.sidebar.selectbox('Choose a task', [
  'Text Summarization', 'Next Word Prediction', 'Story Prediction',
  'Chatbot', 'Sentiment Analysis', 'Question Answering', 'Image Generation'
])
```

- A sidebar widget to select the desired functionality (task) such as **Text Summarization**, **Chatbot**, **Image Generation**, etc.

Task Functions

Each task has a corresponding function that handles the process when the user selects it.

1. Text Summarization

python

Copy code

```
if task == 'Text Summarization':
  st.subheader('📝 Text Summarization')
  user_input = st.text_area('Enter text to summarize:')
  if st.button('Summarize'):
    model_name = "facebook/bart-large-cnn"
    tokenizer = BartTokenizer.from_pretrained(model_name)
    model = BartForConditionalGeneration.from_pretrained(model_name).to(device)

    def summarize(text, model, tokenizer):
      inputs = tokenizer.encode("summarize: " + text, return_tensors="pt", max_length=1024, truncation=True).to(device)
      summary_ids = model.generate(inputs, max_length=200, min_length=30, length_penalty=2.0, num_beams=4, early_stopping=True)
      return tokenizer.decode(summary_ids[0], skip_special_tokens=True)

    summary = summarize(user_input, model, tokenizer)
    st.write("### Summary")
```

st.write(summary)

- **BART** model is used for text summarization.
- The summarize function processes the input text and generates a summary using BART's **Conditional Generation** capability.

2. Next Word Prediction

python

Copy code

```
elif task == 'Next Word Prediction':
  st.subheader('🔮 Next Word Prediction')
  user_input = st.text_area('Enter text for prediction:')
  if st.button('Predict'):
    tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
    model = GPT2LMHeadModel.from_pretrained('gpt2').to(device)

    def predict_next_word(prompt, model, tokenizer, top_k=5):
      inputs = tokenizer(prompt, return_tensors='pt').to(device)
      outputs = model(**inputs)
      next_token_logits = outputs.logits[:, -1, :]
      top_k_tokens = torch.topk(next_token_logits, top_k).indices[0].tolist()
      return [tokenizer.decode([token]) for token in top_k_tokens]

    predicted_words = predict_next_word(user_input, model, tokenizer)
    st.write("### Next Word Predictions")
    st.write(predicted_words)
```

- Uses **GPT-2** model to predict the next word(s) in a given text input.
- The predict_next_word function processes the input prompt and generates predictions based on the top k most likely next tokens.

3. Story Generation

python

Copy code

```
elif task == 'Story Prediction':
  st.subheader('📖 Story Prediction')
```

```
    user_input = st.text_area('Enter text to continue the story:')
    if st.button('Generate'):
        with st.spinner("Generating story..."):
            story_predictor = pipeline('text-generation', model='gpt2')
            story = story_predictor(user_input, max_length=200,
clean_up_tokenization_spaces=True)[0]['generated_text']
            st.write("### Generated Story")
            st.write(story)
```

- Uses **GPT-2** for story continuation.
- The pipeline API handles the text generation, producing a story based on the user's input text.

4. Chatbot

python

Copy code

```
elif task == 'Chatbot':
    st.subheader('🤖 Chatbot')
    model_name = "microsoft/DialoGPT-medium"
    tokenizer = AutoTokenizer.from_pretrained(model_name, padding_side='left')
    model = AutoModelForCausalLM.from_pretrained(model_name).to(device)
    chat_history_ids = None
    user_input = st.text_input("You:")
    if st.button('Chat'):
        if user_input.lower() != 'quit':
            new_user_input_ids = tokenizer.encode(user_input + tokenizer.eos_token,
return_tensors='pt').to(device)
            bot_input_ids = torch.cat([chat_history_ids, new_user_input_ids], dim=-1) if
chat_history_ids is not None else new_user_input_ids
            chat_history_ids = model.generate(bot_input_ids, max_length=1000,
pad_token_id=tokenizer.eos_token_id)
            response = tokenizer.decode(chat_history_ids[:, bot_input_ids.shape[-1]:][0],
skip_special_tokens=True)
            st.write("Bot:", response)
```

- Uses **DialoGPT** for conversational AI.

- The user interacts with the chatbot, and the model responds using a conversational flow.

5. Sentiment Analysis

python

Copy code

```
elif task == 'Sentiment Analysis':
  st.subheader('😊 Sentiment Analysis')
  user_input = st.text_area('Enter text for sentiment analysis:')
  if st.button('Analyze'):
    sentiment_analysis = pipeline("sentiment-analysis")
    results = sentiment_analysis(re.split(r'([.!?])', user_input))
    for result in results:
      st.write(f"Sentiment: {result['label']}, Score: {result['score']:.4f}")
```

- Uses a pre-trained **sentiment analysis** model to determine the sentiment of the input text.
- It outputs the sentiment label (positive/negative) and its score.

6. Question Answering

python

Copy code

```
elif task == 'Question Answering':
  st.subheader('❓ Question Answering')
  context = st.text_area('Enter context text:')
  question = st.text_input('Enter question:')
  if st.button('Answer'):
    with st.spinner("Finding answer..."):
      model_name = "bert-large-uncased-whole-word-masking-finetuned-squad"
      tokenizer = AutoTokenizer.from_pretrained(model_name)
      model = AutoModelForQuestionAnswering.from_pretrained(model_name).to(device)

      inputs = tokenizer.encode_plus(question, context, return_tensors="pt").to(device)
      answer_start_scores, answer_end_scores = model(**inputs).start_logits,
model(**inputs).end_logits
```

```
        answer = tokenizer.convert_tokens_to_string(

tokenizer.convert_ids_to_tokens(inputs["input_ids"].tolist()[0][torch.argmax(answer_start_scores):torch.argmax(answer_end_scores) + 1])

        )

        st.write("Answer:", answer)
```

- Uses **BERT** for question answering.
- The model processes the context and the question to extract the answer.

7. Image Generation

python

Copy code

```
elif task == 'Image Generation':

  st.subheader('🖼️ Image Generation')

  prompt = st.text_input('Enter a prompt for image generation:')

  if st.button('Generate Image'):

    pipe = StableDiffusionPipeline.from_pretrained("CompVis/stable-diffusion-v-1-4-original").to(device)

    image = pipe(prompt).images[0]

    st.image(image)
```

- Uses **Stable Diffusion** for generating images based on a textual prompt.
- The StableDiffusionPipeline is loaded, and the generated image is displayed in the app.

Conclusion

This multifunctional tool integrates state-of-the-art NLP models for various tasks and provides an interactive interface using **Streamlit**. The app demonstrates the flexibility of Hugging Face models and offers a practical example of combining multiple AI capabilities in a single tool.