3a_Gemma_CG

November 12, 2024

Code Gemma LLM setup

This notebook should be run in Google Colab or similar site, where high GPU processing power is available. In Google Colab, the A100 GPU works best.

Loading packages, libraries and secrets into notebook

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

[ ]:
```
# Installing the required packages
!pip install pandas==2.1.4 numpy==1.23.5 gradio sentence_transformers␣
↪tensorflow==2.15
!pip install -U transformers
!pip install torch torchvision torchaudio --index-url https://download.pytorch.
↪org/whl/cu118
# install below if using GPU
!pip install accelerate
```

[2]:
```
# Importing the required functions and modules
import gradio as gr
from gradio.themes.base import Base
from sentence_transformers import SentenceTransformer # https://huggingface.co/
↪thenlper/gte-large
from transformers import AutoTokenizer, AutoModelForCausalLM
from transformers import AutoConfig
import torch
import gc
```

Accessing secrets

[4]:
```
# Accessing the secrets from the environment variables
#load_dotenv()
#HF_Token = os.getenv("HF_TOKEN")

# In Google Colab, you can use the following code to access the secret
from google.colab import userdata
HF_Token = userdata.get('HF_TOKEN')
```

Loading the Tokenizer and LLM-Model

[ ]:
```
tokenizer = AutoTokenizer.from_pretrained("google/codegemma-7b-it")
# CPU Enabled uncomment below
# model = AutoModelForCausalLM.from_pretrained("google/codegemma-7b-it")
# GPU Enabled use below
model = AutoModelForCausalLM.from_pretrained("google/codegemma-7b-it",␣
↪device_map="auto")
```

Chain Setup

[6]:
```
query = ""

output_length = len(query.split())*3 # word count of SQL query multiplied by␣
↪four

def process_query(query):
    # Generate response
    def generate_response(query):
        combined_information = (
            f"Instructions: Generate a natural language Translation stating␣
↪what the Query wants to achieve followed by an Explanation stating how the␣
↪Query is composed and how it works."
            f"Go through it step by step and formulate the Translation and␣
↪Explanation in simple and concise language."
            f"Keep the word count in line with the Length number.\n\n"
            f"Length: {output_length}"
            f"Query: {query}\n\n"
            f"Response:\n"
        )

        # Moving tensors to GPU and generating a response
        input_ids = tokenizer(combined_information, return_tensors="pt").
↪to("cuda")
        response = model.generate(**input_ids, max_new_tokens=1000)
        decoded_response = tokenizer.decode(response[0],␣
↪skip_special_tokens=True).strip()

        # Post-processing: Extracting the content after 'Response:\n'
        if "Response:" in decoded_response:
            decoded_response = decoded_response.split("Response:", 1)[-1].
↪strip()

        # Clear GPU memory for `input_ids` and `response`
        del input_ids, response
        torch.cuda.empty_cache()
        gc.collect()

        return decoded_response
```

```
    # Return the final generated response
    return generate_response(query)
```

Chat interface setup

Markdown format of Chat interface setup for testing.

Change cell type below to Python, when running only this script.

[ ]:
```
# Create a web interface for the app, using Gradio
with gr.Blocks(theme=Base(), title="Question Answering App using Vector Search␣
 ↪+ RAG") as demo:
    gr.Markdown(
        """
        # Question Answering App using Atlas Vector Search + RAG Architecture
        """)
    textbox = gr.Textbox(label="Enter your SQL statement:")
    with gr.Row():
        button = gr.Button("Submit", variant="primary")
    with gr.Column():
        output = gr.Textbox(lines=1, max_lines=30, label="Natural language␣
 ↪translation and explanation:")

# Call chain_invoke function upon clicking the Submit button

    button.click(process_query, textbox, outputs=output)

demo.launch()
```