hugomcp@di.ubi.pt, 2024/25

Interaction with Large Scale Models

Practical Sheet 4

Tutorial

Some platforms (like Hugging Face Inference API) provide access to LLaVA models without needing local setup.

Option 1: Using LLaVA via an Online API

Step 1: Get API Access

1. Sign up on Hugging Face.
2. Go to LLaVA models.
3. If there's an API endpoint, obtain the token.

Step 2: Use Python to Call the API

```
import requests
API_URL = "https://api-inference.huggingface.co/models/liuhaotian/llava-v1.5-7b"
HEADERS = {"Authorization": "Bearer YOUR_HF_API_KEY"}
def query_llava(text):
    payload = {"inputs": text}
    response = requests.post(API_URL, headers=HEADERS, json=payload)
    return response.json()
print(query_llava("Describe the content of an image."))
```

Option 2: Using LLaVA Locally

Step 1: Install Dependencies

Ensure you have PyTorch installed:

```
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121
```

Install other dependencies:

```
pip install transformers accelerate bitsandbytes
pip install git+https://github.com/huggingface/transformers.git
```

Step 2: Download and Load LlaVA

```
from transformers import AutoProcessor, AutoModelForCausalLM
import torch
model_name = "liuhaotian/llava-v1.5-7b"
device = "cuda" if torch.cuda.is_available() else "cpu"
processor = AutoProcessor.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name,
torch_dtype=torch.float16).to(device)
def llava_inference(prompt):
   inputs = processor(text=prompt, return_tensors="pt").to(device)
   output = model.generate(**inputs, max_new_tokens=100)
   return processor.batch_decode(output, skip_special_tokens=True)[0]
print(llava_inference("Hello, LLaVA!"))
```

Exercises

1. Multi-1. Prompt Evaluation. Implement an automated evaluation pipeline to compare different prompts. You should implement a Python script that automates prompt evaluation and reports the best-performing prompt.

1. Choose three different prompts designed to extract similar information from an LLM (e.g., “Explain quicksort” in different phrasings).
2. Use a local LLM API (**LLAVA** or **QWEEn-vl**) to generate responses for each prompt.
3. Obtain reference answers from reliable sources (e.g., textbooks, official documentation).
4. Compute similarity between generated responses and reference answers using cosine similarity with sentence embeddings (e.g., sentence-transformers).
5. Rank the prompts based on similarity scores.
6. Interpret the results and discuss why certain prompts performed better than others.

2. Preventing Hallucinations. Compare LLM responses with and without external knowledge retrieval. For such, develop a a Python script implementing RAG and comparing response accuracy should be obtained.

1. Select a set of fact-based queries requiring precise answers (e.g., “What is the latest breakthrough in deep learning?”).
2. Run these queries through an LLM without additional context and analyze the responses.
3. Use FAISS to create a vector database with relevant documents (e.g., research papers, Wikipedia extracts).
4. Implement a retrieval-augmented generation (RAG) system:
    - Convert queries into embeddings.
    - Retrieve relevant documents from FAISS.

   - Inject retrieved documents into the prompt before sending it to the LLM.
5. Compare responses before and after RAG and assess hallucination reduction.

3. Quantitative and Qualitative Metrics for Prompt Output. Measure LLM responses using BLEU, ROUGE, and human evaluation, and develop a Python script calculating BLEU/ROUGE scores, along with a brief report comparing them to human evaluations.

1. Generate responses for at least three different prompts.
2. Collect reference responses for each prompt.
3. Compute BLEU and ROUGE scores using nltk.translate or rouge_score in Python.
4. Conduct a human evaluation:
   - Rate responses on coherence, fluency, and informativeness (scale of 1–5).
5. Compare automated metrics with human ratings.
6. Discuss cases where metrics and human ratings diverge.

4. Logical Consistency. Detect contradictions in LLM-generated responses. The goal in this exercice is to implementa Python script detecting contradictions and a summary of findings.

1. Create a set of logically structured prompts (e.g., “If A is true, what follows?”).
2. Generate multiple responses from an LLM.
3. Use a pre-trained Natural Language Inference (NLI) model (e.g., facebook/bart-large-mnli) to classify contradictions.
4. Implement a script that flags responses contradicting prior statements.
5. Analyze which prompts produce the most inconsistencies and why.

5. Diversity. Evaluate response diversity using embedding similarity. Implement a Python script measuring response diversity and a refined prompt.

1. Write prompts that request diverse outputs (e.g., “List three applications of reinforcement learning”).
2. Generate multiple responses from an LLM.
3. Convert responses into embeddings using sentence-transformers.
4. Compute cosine similarity between different responses.
5. If similarity is high (e.g., >0.9), refine the prompt to encourage diversity.
6. Repeat the process until the responses are sufficiently varied.

6. Coverage. Ensure responses contain key concepts and implement a script evaluating coverage and an improved prompt.

1. Choose a topic with well-defined subtopics (e.g., “Explain the TCP/IP model” should cover layers, protocols, and security concerns).
2. Generate LLM responses to an initial prompt.
3. Define a checklist of essential concepts.
4. Implement a keyword-matching script to verify concept coverage.
5. Compute a coverage score (e.g., % of expected keywords present).
6. Modify the prompt to improve coverage and test again.

7. Prompt Refinement. Iteratively improve a suboptimal prompt. At the end, obtain a document showing the original and refined prompts with output comparisons.

1. Start with a vague prompt (e.g., “Explain blockchain”).

2. Analyze its shortcomings (e.g., lack of depth, missing aspects).
3. Apply refinement techniques:
    - Specify required details (e.g., “Explain blockchain focusing on consensus mechanisms”).
    - Use delimiters (e.g., “Provide a structured response: Introduction | Key Components | Challenges”).
    - Add role-based instructions (e.g., “Act as a cybersecurity expert”).
4. Compare responses before and after refinement.
5. Repeat until an optimal prompt is achieved.