Generative AI interactions
our main code references: https://www.datacamp.com/tutorial/fine-tuning-llama-2

1. how to load the llama2 chat model from hugging face, give me the code

To load the Llama-2 model from Hugging Face, you will typically use the Transformers library by Hugging Face. However, as of my last update in April 2023, there isn't a model specifically called "Llama-2" publicly available in Hugging Face's model repository. But I will provide you with a generic code template for loading a model from Hugging Face, which you can use if a model by that name or similar is available.

First, ensure you have the transformers library installed. If not, you can install it using pip:

python
Copy code
```
!pip install transformers
```
Then, you can use the following code to load a model from Hugging Face's repository:

python
Copy code
```
from transformers import AutoModelForCausalLM, AutoTokenizer

model_name = "your-model-name"    # Replace with the actual model name

# Load pre-trained model tokenizer (vocabulary)
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Load pre-trained model
model = AutoModelForCausalLM.from_pretrained(model_name)
```
Please replace "your-model-name" with the actual model name you want to load. For example, if you're loading the GPT-2 model, it would be "gpt2".

If the model is a custom fine-tuned version uploaded by a user, it will typically have a username/modelname format, like "username/modelname". Make sure to use the exact name from the Hugging Face model repository.

Remember that using large language models can require a lot of resources (RAM and GPU), so make sure your environment has enough resources to load and use the model.

2. give me a coding framework of fine tuning

Fine-tuning a pre-trained model on a new dataset typically involves several steps: preparing the dataset, creating the model, defining a training loop, and running the training process. Here's a general framework for fine-tuning a model using Hugging Face's Transformers library and PyTorch.

python
Copy code
```
import torch
from torch.utils.data import DataLoader, Dataset
from transformers import AutoModelForCausalLM, AutoTokenizer, AdamW, get_scheduler

# Define your custom dataset
class CustomDataset(Dataset):
    def __init__(self, texts, tokenizer, max_length):
        self.encodings = tokenizer(texts, truncation=True, padding=True, max_length=max_length)

    def __getitem__(self, idx):
        return {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}

    def __len__(self):
        return len(self.encodings['input_ids'])

# Load the tokenizer and model
model_name = "gpt2"  # replace with the model you want to fine-tune
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

# Prepare the dataset
texts = [...]  # replace with your list of texts
max_length = 512  # the maximum length of the sequence
dataset = CustomDataset(texts, tokenizer, max_length)

# Create data loaders
train_loader = DataLoader(dataset, batch_size=8, shuffle=True)

# Define the optimizer
optimizer = AdamW(model.parameters(), lr=5e-5)

# Define the learning rate scheduler
num_epochs = 3
num_training_steps = num_epochs * len(train_loader)
lr_scheduler = get_scheduler(
    name="linear",
    optimizer=optimizer,
    num_warmup_steps=0,
    num_training_steps=num_training_steps
)
```

```
# Move model to GPU if available
device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
model.to(device)

# Training loop
model.train()
for epoch in range(num_epochs):
    for batch in train_loader:
        # Move batch to device
        batch = {k: v.to(device) for k, v in batch.items()}
        # Forward pass
        outputs = model(**batch, labels=batch["input_ids"])
        loss = outputs.loss
        # Backward pass
        loss.backward()
        # Update parameters
        optimizer.step()
        # Reset gradients
        optimizer.zero_grad()
        # Update learning rate
        lr_scheduler.step()

    print(f"Epoch {epoch} completed. Loss: {loss.item()}")

# Save the fine-tuned model
model.save_pretrained("my_fine_tuned_model")
```

This code provides a skeleton for fine-tuning a causal language model

3. show me how to do prompt engineering using coding

Prompt engineering involves designing prompts that guide a language model to produce the desired type of response. When using coding to perform prompt engineering with a language model such as GPT-2 or GPT-3, the process involves carefully crafting the text that you feed into the model.

Here's a simplified example of how you might use prompt engineering with Hugging Face's transformers library to interact with a language model:

python

Copy code

```
from transformers import pipeline

# Load the model and tokenizer
generator = pipeline('text-generation', model='gpt2')

# Define a function that adds a prompt to input text
```

```
def generate_with_prompt(input_text, prompt, max_length=50):
    # Combine the input text with the prompt
    text_with_prompt = f"{prompt}{input_text}"

    # Generate a response
    generated_text = generator(text_with_prompt,
max_length=max_length)[0]['generated_text']

    # Optionally, you can clean up the generated text by removing the input prompt
    clean_text = generated_text[len(text_with_prompt):]

    return clean_text.strip()

# Example prompts
prompt_cot = "Think step by step: "    # Chain of thoughts prompt
prompt_example = "For example: "         # Providing an example for context
prompt_instruction = "The task is to: "    # Task instruction prompt

# Example input
input_text = "What are the benefits of renewable energy?"

# Generate responses with different prompts
response_cot = generate_with_prompt(input_text, prompt_cot)
response_example = generate_with_prompt(input_text, prompt_example)
response_instruction = generate_with_prompt(input_text, prompt_instruction)

# Print the responses
print("Response with Chain of Thoughts Prompt:")
print(response_cot)
print("\nResponse with Example Prompt:")
print(response_example)
print("\nResponse with Task Instruction Prompt:")
print(response_instruction)
```

In this example, the generate_with_prompt function takes an input text and a prompt, combines them, and feeds them into the language model. The model then generates a response that follows the direction or context provided by the prompt.

4. give me the automatic evaluation of generated text coding sample

3. Prompt Engineering Example

Prompt engineering involves designing the input text (the prompt) to guide a language model to produce the desired output. Here's an example of how to perform prompt engineering with a pre-trained model from Hugging Face:

python

Copy code

```
from transformers import pipeline

# Load a pre-trained model and tokenizer from Hugging Face
model_name = "gpt2"    # replace with your model
generator = pipeline('text-generation', model=model_name)

# Define your prompt
prompt = "The current trends in AI policy indicate"

# Perform prompt engineering by adding instructions or context to the prompt
engineered_prompt = f"{prompt} a shift towards more ethical considerations. Discuss the implications of this shift:"

# Generate text with the engineered prompt
generated_texts = generator(engineered_prompt, max_length=100, num_return_sequences=1)

# Print the generated text
for generated_text in generated_texts:
        print(generated_text['generated_text'])
```

In this example, we've engineered the prompt to lead the model to discuss the implications of a specific trend in AI policy.

4. Automatic Evaluation Example

Automatic evaluation of generated text often uses metrics like BLEU, ROUGE, and METEOR. Here's how you can evaluate generated text against a reference text using these metrics:

python
Copy code

```
from nltk.translate.bleu_score import sentence_bleu
from rouge import Rouge
from nltk.translate.meteor_score import meteor_score

# Assume we have a generated text and a reference text
generated_text = "The AI policy has been evolving rapidly."
reference_text = "AI policy evolution is swift and dynamic."

# Compute BLEU score
bleu_score = sentence_bleu([reference_text.split()], generated_text.split())

# Compute ROUGE score
rouge = Rouge()
rouge_scores = rouge.get_scores(generated_text, reference_text)
```

```
# Compute METEOR score
meteor = meteor_score([reference_text.split()], generated_text.split())

# Print the evaluation scores
print(f"BLEU score: {bleu_score}")
print(f"ROUGE scores: {rouge_scores}")
print(f"METEOR score: {meteor}")
```

This code snippet assumes that you have the NLTK library installed for BLEU and METEOR (pip install nltk) and the rouge package for the ROUGE score (pip install rouge-score).

Note: These metrics have limitations and may not always correlate with human judgments of quality. They should be part of a broader evaluation strategy that includes qualitative assessments.

5. give me the coding how visualize the data

I can provide you with a general framework for each of your requests:

Prompt Engineering Using Coding:

python

Copy code

```
from transformers import pipeline, set_seed

generator = pipeline('text-generation', model='gpt2')
set_seed(42)

prompts = ["The current state of AI policy is ",
           "The ethical implications of AI are ",
           "When considering AI regulation, one must account for "]

generated_texts = []

for prompt in prompts:
    result = generator(prompt, max_length=100, num_return_sequences=1)
    generated_texts.append(result[0]['generated_text'])

for i, text in enumerate(generated_texts):
    print(f"Prompt: {prompts[i]}\nGenerated text: {text}\n")
```

Automatic Evaluation of Generated Text Coding Sample:

python

Copy code

```
from transformers import pipeline
from datasets import load_metric
```

```
bleu_metric = load_metric('bleu')
rouge_metric = load_metric('rouge')
meteor_metric = load_metric('meteor')

reference_texts = [...]    # Replace with your reference texts
generated_texts = [...]    # Replace with texts generated by your model

# Calculating BLEU score
bleu_score = bleu_metric.compute(predictions=generated_texts, references=reference_texts)

# Calculating ROUGE score
rouge_score = rouge_metric.compute(predictions=generated_texts, references=reference_texts)

# Calculating METEOR score
meteor_score = meteor_metric.compute(predictions=generated_texts, references=reference_texts)

print(f"BLEU score: {bleu_score}\nROUGE score: {rouge_score}\nMETEOR score: {meteor_score}\n")
```

Visualize the Data:

python

Copy code

```
import pandas as pd
import matplotlib.pyplot as plt

# Sample DataFrame
df = pd.DataFrame({
    'BLEU': [0.25, 0.35, 0.30],
    'ROUGE': [0.55, 0.65, 0.60],
    'METEOR': [0.40, 0.50, 0.45]
})

# Visualizing the scores
ax = df.plot(kind='bar', figsize=(10, 6))
ax.set_title('Evaluation Scores')
ax.set_xlabel('Model Versions')
ax.set_ylabel('Scores')
ax.set_xticklabels(['Model 1', 'Model 2', 'Model 3'], rotation=0)
plt.show()
```

Please replace the placeholder arrays (...) with your actual data. Each code snippet is meant to be used independently and will need to be adapted to fit the specific requirements of your project and data.