```
%%capture
# Installs Unsloth, Xformers (Flash Attention) and all other packages!
!pip install unsloth
# Get latest Unsloth
!pip install --upgrade --no-deps "unsloth[colab-new] @ git+https://github.com/unslothai/unsloth.git"
```

Initializing the Model

```
from unsloth import FastLanguageModel
import torch
max_seq_length = 2048 # Choose any! We auto support RoPE Scaling internally!
dtype = None # None for auto detection. Float16 for Tesla T4, V100, Bfloat16 for Ampere+
load_in_4bit = True # Use 4bit quantization to reduce memory usage. Can be False.

# 4bit pre quantized models we support for 4x faster downloading + no OOMs.
fourbit_models = [
    "unsloth/mistral-7b-v0.3-bnb-4bit",      # New Mistral v3 2x faster!
    "unsloth/mistral-7b-instruct-v0.3-bnb-4bit",
    "unsloth/llama-3-8b-bnb-4bit",           # Llama-3 15 trillion tokens model 2x faster!
    "unsloth/llama-3-8b-Instruct-bnb-4bit",
    "unsloth/llama-3-70b-bnb-4bit",
    "unsloth/Phi-3-mini-4k-instruct",        # Phi-3 2x faster!
    "unsloth/Phi-3-medium-4k-instruct",
    "unsloth/mistral-7b-bnb-4bit",
    "unsloth/gemma-7b-bnb-4bit",             # Gemma 2.2x faster!
] # More models at https://huggingface.co/unsloth

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "unsloth/llama-3-8b-bnb-4bit",
    max_seq_length = max_seq_length,
    dtype = dtype,
    load_in_4bit = load_in_4bit,
    # token = "hf_...", # use one if using gated models like meta-llama/Llama-2-7b-hf
)
```

```
from unsloth import FastLanguageModel
import torch
max_seq_length = 2048 # Choose any! We auto support RoPE Scaling internally!
dtype = None # None for auto detection. Float16 for Tesla T4, V100, Bfloat16 for Ampere+
load_in_4bit = True # Use 4bit quantization to reduce memory usage. Can be False.

# 4bit pre quantized models we support for 4x faster downloading + no OOMs.
fourbit_models = [
    "unsloth/mistral-7b-v0.3-bnb-4bit",      # New Mistral v3 2x faster!
    "unsloth/mistral-7b-instruct-v0.3-bnb-4bit",
    "unsloth/llama-3-8b-bnb-4bit",           # Llama-3 15 trillion tokens model 2x faster!
    "unsloth/llama-3-8b-Instruct-bnb-4bit",
    "unsloth/llama-3-70b-bnb-4bit",
    "unsloth/Phi-3-mini-4k-instruct",        # Phi-3 2x faster!
    "unsloth/Phi-3-medium-4k-instruct",
    "unsloth/mistral-7b-bnb-4bit",
    "unsloth/gemma-7b-bnb-4bit",             # Gemma 2.2x faster!
] # More models at https://huggingface.co/unsloth

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "unsloth/llama-3-8b-bnb-4bit",
    max_seq_length = max_seq_length,
    dtype = dtype,
    load_in_4bit = load_in_4bit,
    # token = "hf_...", # use one if using gated models like meta-llama/Llama-2-7b-hf
)
```

```
🦥 Unsloth: Will patch your computer to enable 2x faster free finetuning.
🦥 Unsloth Zoo will now patch everything to make training faster!
==((====))==  Unsloth 2024.12.8: Fast Llama patching. Transformers: 4.46.3.
   \\   /|    GPU: Tesla T4. Max memory: 14.748 GB. Platform: Linux.
O^O/ \_/ \    Torch: 2.5.1+cu121. CUDA: 7.5. CUDA Toolkit: 12.1. Triton: 3.1.0
\        /    Bfloat16 = FALSE. FA [Xformers = 0.0.28.post3. FA2 = False]
 "-____-"     Free Apache license: http://github.com/unslothai/unsloth
Unsloth: Fast downloading is enabled - ignore downloading bars which are red colored!
model.safetensors: 100%                              5.70G/5.70G [00:14<00:00, 508MB/s]
generation_config.json: 100%                         198/198 [00:00<00:00, 15.9kB/s]
tokenizer_config.json: 100%                          50.6k/50.6k [00:00<00:00, 3.12MB/s]
tokenizer.json: 100%                                 9.09M/9.09M [00:01<00:00, 8.58MB/s]
special_tokens_map.json: 100%                        350/350 [00:00<00:00, 21.4kB/s]
==((====))==  Unsloth 2024.12.8: Fast Llama patching. Transformers: 4.46.3.
   \\   /|    GPU: Tesla T4. Max memory: 14.748 GB. Platform: Linux.
O^O/ \_/ \    Torch: 2.5.1+cu121. CUDA: 7.5. CUDA Toolkit: 12.1. Triton: 3.1.0
\        /    Bfloat16 = FALSE. FA [Xformers = 0.0.28.post3. FA2 = False]
 "-____-"     Free Apache license: http://github.com/unslothai/unsloth
Unsloth: Fast downloading is enabled - ignore downloading bars which are red colored!
```

```python
model = FastLanguageModel.get_peft_model(
    model,
    r = 16, # Choose any number > 0 ! Suggested 8, 16, 32, 64, 128
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj",
                      "gate_proj", "up_proj", "down_proj",],
    lora_alpha = 16,
    lora_dropout = 0, # Supports any, but = 0 is optimized
    bias = "none",    # Supports any, but = "none" is optimized
    # [NEW] "unsloth" uses 30% less VRAM, fits 2x larger batch sizes!
    use_gradient_checkpointing = "unsloth", # True or "unsloth" for very long context
    random_state = 3407,
    use_rslora = False,  # We support rank stabilized LoRA
    loftq_config = None, # And LoftQ
)
```

```
Unsloth 2024.12.8 patched 32 layers with 32 QKV layers, 32 O layers and 32 MLP layers.
```

Dataset Preparation :

Loading the question_with_options along with the predicted_label and the gpt_reasoning information as a csv.

```python
from datasets import load_dataset
dataset = load_dataset(
    "csv",
    data_files = "dataset.csv",
    split = "train",
)
print(dataset.column_names)
print(dataset[0])
```

```
Generating train split:     542/0 [00:00<00:00, 9367.46 examples/s]
['questions_with_options', 'predicted_label', 'gpt_reasoning']
{'questions_with_options': "A 58-year-old man comes to the physician for a 3-month history of progress
```

Converting the dataset into required format:

```
from unsloth import to_sharegpt

# Define a post-processing function to combine 'predicted_label' and 'reasoning'
def combine_columns(example):
    example["predicted_label_and_reasoning"] = f"Label: {example['predicted_label']}, Reasoning: {example[
    return example

# Apply the function to the dataset
dataset = dataset.map(combine_columns)

# Verify the formatted dataset
print(dataset[0])


# Merge "predicted_label" and "reasoning" into the response in the merged_prompt
dataset = to_sharegpt(
    dataset,
    merged_prompt="{questions_with_options}",
    output_column_name="predicted_label_and_reasoning",  # Combine predicted_label and reasoning
)


print(dataset[0])
```

```
Map: 100%                                        542/542 [00:00<00:00, 10120.89 examples/s]
{'questions_with_options': "A 58-year-old man comes to the physician for a 3-month history of progress
Merging columns: 100%                                        542/542 [00:00<00:00, 15101.26 examples/s]

Converting to ShareGPT: 100%                                        542/542 [00:00<00:00, 18717.48 examples/s]
{'conversations': [{'from': 'human', 'value': '("A 58-year-old man comes to the physician for a 3-mont
```

Standardize share-gpt

```
from unsloth import standardize_sharegpt
dataset = standardize_sharegpt(dataset)
```

```
Standardizing format: 100%                                        542/542 [00:00<00:00, 15184.67 examples/s]
```

Chat Template

```
chat_template = """Below are questions with options. Provide the predicted label and reasoning.

>>> Question:
{INPUT}

>>> Answer:
{OUTPUT}

"""

from unsloth import apply_chat_template
```

```
dataset = apply_chat_template(
    dataset,
    tokenizer = tokenizer,
    chat_template = chat_template,
    # default_system_message = "You are a helpful assistant", << [OPTIONAL]
)
```

```
Unsloth: We automatically added an EOS token to stop endless generations.
Map: 100%                                         542/542 [00:00<00:00, 9981.00 examples/s]
```

Training the Model

```
from trl import SFTTrainer
from transformers import TrainingArguments
from unsloth import is_bfloat16_supported

trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    train_dataset = dataset,
    dataset_text_field = "text",
    max_seq_length = max_seq_length,
    dataset_num_proc = 2,
    packing = False, # Can make training 5x faster for short sequences.
    args = TrainingArguments(
        per_device_train_batch_size = 2,
        gradient_accumulation_steps = 4,
        warmup_steps = 5,
        max_steps = 60, ## comment this later
        num_train_epochs = 5, ## train for 5 epochs ideally
        learning_rate = 2e-4,
        fp16 = not is_bfloat16_supported(),
        bf16 = is_bfloat16_supported(),
        logging_steps = 1,
        optim = "adamw_8bit",
        weight_decay = 0.01,
        lr_scheduler_type = "linear",
        seed = 3407,
        output_dir = "outputs",
    ),
)
```

```
Map (num_proc=2): 100%                                 542/542 [00:02<00:00, 308.26 examples/s]
max_steps is given, it will override any value given in num_train_epochs
```

```
trainer_stats = trainer.train()
```

```
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs = 1
   \\   /|    Num examples = 542 | Num Epochs = 1
O^O/ \_/ \    Batch size per device = 2 | Gradient Accumulation steps = 4
\        /    Total batch size = 8 | Total steps = 60
 "-____-"     Number of trainable parameters = 41,943,040
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`.
wandb: Using wandb-core as the SDK backend.  Please refer to https://wandb.me/wandb-core for more in
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-serv
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit: ··········
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
```

Tracking run with wandb version 0.19.1
Run data is saved locally in `/content/wandb/run-20241223_163918-em3pu99r`
Syncing run **outputs** to Weights & Biases (docs)
View project at https://wandb.ai/rahrsaxena-umass-amherst/huggingface
View run at https://wandb.ai/rahrsaxena-umass-amherst/huggingface/runs/em3pu99r

[60/60 11:17, Epoch 0/1]

Step	Training Loss
1	1.744200
2	1.792600
3	1.768500
4	1.644000
5	1.642900
6	1.637400
7	1.448900
8	1.369500
9	1.201600
10	1.111400
11	1.052500
12	1.110300
13	1.045800
14	0.956100
15	0.921200
16	0.990600
17	0.985500
18	0.977400
19	0.929700
20	0.925900
21	0.970600
22	0.943500
23	0.935800
24	0.944000
25	0.916100
26	0.867300
27	0.864300
28	0.911400

29	0.929300
30	0.817300
31	0.941400
32	0.956500
33	0.899100
34	0.856800
35	0.883500
36	0.841600
37	0.922200
38	0.841500
39	0.864100
40	0.803000
41	0.853600
42	0.799300
43	0.833000
44	0.874400
45	0.901200
46	0.838600
47	0.858400
48	0.874600
49	0.734900
50	0.971300
51	0.852700
52	0.891300
53	0.825400
54	0.881700
55	0.854400
56	0.791200
57	0.767300
58	0.871900
59	0.884800
60	0.808700

Saving the fine-tuned model

```
# Step 9: Save the fine-tuned model
model.save_pretrained("fine_tuned_llama")
tokenizer.save_pretrained("fine_tuned_llama_tokenizer")
```

```
('fine_tuned_llama_tokenizer/tokenizer_config.json',
 'fine_tuned_llama_tokenizer/special_tokens_map.json',
 'fine_tuned_llama_tokenizer/tokenizer.json')
```

Performing Inference using the excel file containing the ground truth label and reasoning

For this, we have taken the 100 records (50 easy, 50 difficult) containing the question with options, the ground truth label, as well as gpt-reasoning for that ground truth label.

```
import pandas as pd

# Read the CSV file into a DataFrame
df = pd.read_csv("inference.csv")

# Display the first few rows of the DataFrame
print(df.head())
```

```
                              questions_with_options  ground_truth_label  \
0  A 60-year-old man is brought to the emergency ...                   0   
1  A previously healthy 19-year-old man is brough...                   0   
2  A 23-year-old woman is brought to the emergenc...                   0   
3  Prior to undergoing a total knee arthroplasty,...                   0   
4  A 28-year-old woman comes to the physician bec...                   0   

                                       gpt_reasoning  
0  The question is considered to have a difficult...  
1  The question is considered to have a difficult...  
2  The question is considered to have a difficult...  
3  The question is considered to have a difficult...  
4  The question is considered to have a difficult...  
```

```
# Define the additional string that will be added to each content
additional_content = (
    "Please provide the output in the following format:\n\n"
    "Response:\n"
    "Difficulty Level: 0 or 1 (depending on difficulty)\n"
    "Reasoning: \"<Reasoning text>\"\n"
)
```

```
# Prepare the messages for inference by extracting the 'question with options' column
messages = [{"role": "user", "content": row["questions_with_options"] + additional_content} for _, row in d
```

```
from transformers import TextStreamer
import re

FastLanguageModel.for_inference(model)  # Enable faster inference
```

```python
responses = []  # To store model responses

# Define regex patterns to extract the label and reasoning
label_pattern = r"Label: (\d)"
reasoning_pattern = r"Reasoning: (.*)"

text_streamer = TextStreamer(tokenizer, skip_prompt=True)  # For streaming output

# Generate responses for each message
for idx, row in df.iterrows():
    # Prepare the message by adding additional content
    additional_content = (
        "Please provide the output in the following format:\n\n"
        "Response:\n"
        "Difficulty Level: 0 or 1 (depending on difficulty)\n"
        "Reasoning: \"<Reasoning text>\""
    )
    message = {"role": "user", "content": row["questions_with_options"] + additional_content}

    input_ids = tokenizer.apply_chat_template(
        [message],  # Each message as input
        add_generation_prompt=True,
        return_tensors="pt",
    ).to("cuda")

    generated_output = model.generate(
        input_ids,
        streamer=text_streamer,
        max_new_tokens=256,
        pad_token_id=tokenizer.eos_token_id
    )

    # Decode the generated output
    response_text = tokenizer.decode(generated_output[0], skip_special_tokens=True)

    # Use regex to extract the predicted label and reasoning
    label_match = re.search(label_pattern, response_text)
    reasoning_matches = re.findall(reasoning_pattern, response_text)

    # Extract label and reasoning
    predicted_label = label_match.group(1) if label_match else None
    reasoning = reasoning_matches[1] if len(reasoning_matches) > 1 else None

    # Append the response and additional info to the responses list
    responses.append({
        "difficulty": row["ground_truth_label"],  # Ground truth difficulty
        "difficulty_prediction": predicted_label,  # Predicted difficulty
        "reasoning": row["gpt_reasoning"],    # Ground truth reasoning
        "reasoning_prediction": reasoning  # Predicted reasoning
    })

# Convert responses to a DataFrame
responses_df = pd.DataFrame(responses)
```

```
    <|end_of_text|>
    Label: 0, Reasoning: determine the difficulty. The question presents a clinical scenario involving a

    <|end_of_text|>
    Label: 0, Reasoning: determine the difficulty. The question presents a clinical scenario involving a

    <|end_of_text|>
    Label: 0, Reasoning: determine the difficulty. The question presents a clinical scenario involving a

    <|end_of_text|>
    Label: 0, Reasoning: determine the difficulty. The question presents a clinical scenario involving a

    <|end_of_text|>
    Label: 0, Reasoning: determine the difficulty. The question presents a clinical scenario involving a

    <|end_of_text|>
    Label: 0, Reasoning: determine the difficulty. The question presents a clinical scenario involving a

    <|end_of_text|>
    Label: 0, Reasoning: determine the difficulty. The question presents a clinical scenario involving a

    <|end_of_text|>
    Label: 1, Reasoning: determine the difficulty. The question presents a clinical scenario involving a

    <|end_of_text|>
    Label: 0, Reasoning: determine the difficulty. The question presents a clinical scenario involving a

    <|end_of_text|>
    Label: 0, Reasoning: determine the difficulty. The question presents a clinical scenario involving a

    <|end_of_text|>
    Label: 0, Reasoning: determine the difficulty. The question presents a clinical scenario involving a

    <|end_of_text|>
    Label: 0, Reasoning: determine the difficulty. The question presents a clinical scenario involving a

    <|end_of_text|>
    Label: 0, Reasoning: determine the difficulty. The question presents a clinical scenario involving a

    <|end_of_text|>
    Label: 0, Reasoning: determine the difficulty. The question presents a clinical scenario involving a

    <|end_of_text|>
    Label: 1, Reasoning: determine the difficulty. The question presents a clinical scenario involving a

    <|end_of_text|>
```

```
# Convert responses to a DataFrame
responses_df = pd.DataFrame(responses)
responses_df
```

	difficulty	difficulty_prediction	reasoning	reasoning_prediction
0	0	0	The question is considered to have a difficult...	determine the difficulty. The question present...
1	0	1	The question is considered to have a difficult...	determine the difficulty. The question present...
2	0	0	The question is considered to have a difficult...	determine the difficulty. The question present...
3	0	0	The question is considered to have a difficult...	determine the difficulty. The question involve...
4	0	1	The question is considered to have a difficult...	determine the difficulty. The question present...
...	...	...	...	...
94	1	0	The question is considered to have a difficult...	determine the difficulty. The question present...
95	1	0	The question is considered to have a difficult...	determine the difficulty. The question present...

Next steps: Generate code with `responses_df` View recommended plots New interactive sheet

Evaluation Metrics and pre-defined function

```
!pip install bert-score
!pip install nltk
import nltk
nltk.download('punkt_tab')
nltk.download('wordnet')
```

```
Collecting bert-score
  Downloading bert_score-0.3.13-py3-none-any.whl.metadata (15 kB)
Requirement already satisfied: torch>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from bert-scor
Requirement already satisfied: pandas>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from bert-sco
Requirement already satisfied: transformers>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from be
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from bert-score) (1.2
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from bert-score) (
Requirement already satisfied: tqdm>=4.31.1 in /usr/local/lib/python3.10/dist-packages (from bert-scor
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from bert-score)
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.10/dist-packages (from bert-s
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.0.0-
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.0.0-
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.0.0->b
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch>=1.0.0->b
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages (from torch>=1
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sym
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transforme
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from tran
Requirement already satisfied: tokenizers<0.21,>=0.20 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.10/dist-packages (from tra
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matpl
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotli
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matp
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matp
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotl
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matpl
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests-
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from req
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from req
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateut
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2
Downloading bert_score-0.3.13-py3-none-any.whl (61 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.1/61.1 kB 2.4 MB/s eta 0:00:00
Installing collected packages: bert-score
Successfully installed bert-score-0.3.13
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.9.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.67.1)
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```python
import requests
import re
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, confusion_matrix
from bert_score import score
from nltk.translate import meteor
from nltk.tokenize import word_tokenize

def get_openai_response(prompt, model):
    api_key = "sk-proj-2zohbQKony_xvesrkQ8KxphWk5uIJ_IJ37cX5w3EZ6w7PD2lOG7m8wVYp0xYypHFlSAJyh5-FFT3BlbkFJs
    org_key = "org-ewbrRzXdrHxv7hV0WyCFzGdD"

    """
    Sends a prompt to OpenAI's API and retrieves the response.

    Args:
        prompt (str): The prompt for the LLM.
        model (str): The model to use (e.g., 'gpt-4').
        api_key (str): OpenAI API key.
        org_key (str): OpenAI organization key.

    Returns:
        str: The response text from the LLM.
    """
    url = 'https://api.openai.com/v1/chat/completions'
    headers = {
        'Content-Type': 'application/json',
        'Authorization': f'Bearer {api_key}',
        'OpenAI-Organization': org_key
    }
    data = {
        'messages': [{'role': 'system', 'content': prompt}],
        'model': model,
        'temperature': 0.0
    }
    response = requests.post(url, headers=headers, json=data)
    if response.status_code == 200:
        return response.json()['choices'][0]['message']['content']
    else:
        raise Exception(f"API request failed with status code {response.status_code}: {response.text}")
```

```
def llm_as_a_judge_prompt(conversation, reasoning_summary):
    """
    Generates a prompt for the LLM-as-a-judge evaluation.

    Args:
        conversation (str): The conversation or context text.
        reasoning_summary (str): The reasoning summary text.

    Returns:
        str: The generated prompt.
    """
    prompt = f"""### Instruction: Evaluate the reasoning for predicting the difficulty of medical question

### Scoring Criteria:

**Case 1: Simple Questionnaire (Low Difficulty)**
- **2 points** if the reasoning clearly indicates that the questionnaire is simple with few questions, mir
- **1 point** if the reasoning indicates that the questionnaire might be simple, but lacks clarity or supp
- **0 points** if no reasoning is provided or it contradicts the idea of simplicity.

**Case 2: Complex Questionnaire (High Difficulty)**
- **2 points** if the reasoning clearly indicates that the questionnaire is complex with multiple question
- **1 point** if the reasoning indicates that the questionnaire might be complex, but lacks enough support
- **0 points** if no reasoning is provided or it contradicts the idea of complexity.

**General Evaluation Criteria:**
- **Clarity and Coherence**: 0.5 points for clear, well-structured reasoning.
- **Relevance**: 0.5 points if the reasoning is relevant to predicting the difficulty of the questionnaire
- **Accuracy**: 1 point if the difficulty prediction aligns with the conversation content.

### Input:
- **Conversation**:
{conversation}

- **Summary (Reasoning for difficulty prediction)**:
{reasoning_summary}

### Output:
- "score: <total points>"
- Briefly justify your score, up to 50 words.
"""
    return prompt

def evaluate_difficulty_and_reasoning(df, model):
    """
    Evaluates the dataframe, including `LLM as a judge` metric.

    Args:
        df (pd.DataFrame): Input dataframe with columns for difficulty, predictions, and reasoning.
        model (str): The LLM model to use (e.g., 'gpt-4').
        api_key (str): OpenAI API key.
        org_key (str): OpenAI organization key.

    Returns:
        dict: A dictionary of metrics for difficulty, reasoning, and LLM as a judge.
    """
    # Validate required columns
    required_columns = ['difficulty', 'difficulty_prediction', 'reasoning', 'reasoning_prediction']
    if not all(col in df.columns for col in required_columns):
        raise ValueError(f"Dataframe must contain the following columns: {required_columns}")

    # Metrics for difficulty predictions
    difficulty_metrics = {
        "accuracy": accuracy_score(df['difficulty'], df['difficulty_prediction']),
        "f1": f1_score(df['difficulty'], df['difficulty_prediction'], average='weighted'),
        "precision": precision_score(df['difficulty'], df['difficulty_prediction'], average='weighted'),
        "recall": recall_score(df['difficulty'], df['difficulty_prediction'], average='weighted'),
        "confusion_matrix": confusion_matrix(df['difficulty'], df['difficulty_prediction']).tolist()
    }

    # Metrics for reasoning predictions
    reasoning_metrics = {
        "bert_score": {"precision": [], "recall": [], "f1": []},
        "meteor_score": []
    }
    llm_judge_scores = []
    for _, row in df.iterrows():
        ref = row['reasoning']
        pred = row['reasoning_prediction']
        # Compute BERTScore
        P, R, F1 = score([pred], [ref], lang='en', verbose=False)
        reasoning_metrics["bert_score"]["precision"].append(P.mean().item())
        reasoning_metrics["bert_score"]["recall"].append(R.mean().item())
        reasoning_metrics["bert_score"]["f1"].append(F1.mean().item())

        # Compute METEOR
        meteor_score_value = meteor([word_tokenize(ref)], word_tokenize(pred))
        reasoning_metrics["meteor_score"].append(meteor_score_value)

        # Generate LLM-as-a-judge prompt
        prompt = llm_as_a_judge_prompt(ref, pred)
        try:
            llm_response = get_openai_response(prompt, model)
            llm_score = extract_score_from_llm_response(llm_response)
        except Exception as e:
            print(f"Error in LLM scoring: {e}")
            llm_score = None
        llm_judge_scores.append(llm_score)

    # Aggregate BERTScore and METEOR
    reasoning_metrics["bert_score"] = {
        "precision": np.mean(reasoning_metrics["bert_score"]["precision"]),
        "recall": np.mean(reasoning_metrics["bert_score"]["recall"]),
        "f1": np.mean(reasoning_metrics["bert_score"]["f1"])
    }
    reasoning_metrics["meteor_score"] = np.mean(reasoning_metrics["meteor_score"])

    # Combine all metrics
    return {
        "difficulty_metrics": difficulty_metrics,
        "reasoning_metrics": reasoning_metrics,
        "llm_judge_scores": {
            "mean_score": np.nanmean(llm_judge_scores),
            "scores": llm_judge_scores
        }
    }

def extract_score_from_llm_response(response):
    """
    Extracts the score from LLM response text.

    Args:
        response (str): The text response from the LLM.

    Returns:
        float: The extracted score.
    """
    pattern = r"score:\s*(\d+(\.\d+)?)"
```