```
# ====================================================================
# PART 1: PREPARE THE DATASET
# ====================================================================
import json
from collections import defaultdict

print("--- PART 1: Preparing Dataset from utterances.jsonl ---")


DATASET_FILE = 'utterances.jsonl'
# The final file that the trainer will use
FORMATTED_DATASET_FILE = 'friends_final_conversations.jsonl'

# A dictionary to group all lines by their conversation ID
conversations = defaultdict(list)

# Read the raw .jsonl file line by line
try:
    with open(DATASET_FILE, 'r') as f:
        for line in f:
            # Load each line as a JSON object
            utterance = json.loads(line)

            # Group utterances by their 'conversation_id'
            # We also store the 'id' to sort them correctly later
            conversations[utterance['conversation_id']].append(
                {'id': utterance['id'], 'text': utterance['text']}
            )
except FileNotFoundError:
    print(f"❌ FATAL ERROR: The file '{DATASET_FILE}' was not found.")
    print("   Please make sure you've uploaded it to your Colab session.")
    raise SystemExit()

print(f"✅ Found {len(conversations)} conversations.")

# This will hold our final training data (input/response pairs)
training_data = []

# Process each conversation
for conv_id, utterances in conversations.items():
    # Sort the utterances in the conversation by their ID to get the correct order
    sorted_utterances = sorted(utterances, key=lambda x: x['id'])

    # Extract just the text
    conversation_texts = [utt['text'] for utt in sorted_utterances if utt['text'].strip()]

    # Create pairs of (line, reply)
    if len(conversation_texts) > 1:
        for i in range(len(conversation_texts) - 1):
            input_text = conversation_texts[i]
            response_text = conversation_texts[i+1]

            # Add the pair to our training data
            entry = {'conversation': [input_text, response_text]}
            training_data.append(entry)

# Save the final, formatted dataset
with open(FORMATTED_DATASET_FILE, 'w') as f:
    for entry in training_data:
        f.write(json.dumps(entry) + '\n')

print(f"✅ Dataset prepared with {len(training_data)} conversational pairs.")
print(f"   Formatted data saved to '{FORMATTED_DATASET_FILE}'.")
print("-" * 40)
```

```
--- PART 1: Preparing Dataset from utterances.jsonl ---
✅ Found 3107 conversations.
✅ Dataset prepared with 58211 conversational pairs.
   Formatted data saved to 'friends_final_conversations.jsonl'.
----------------------------------------
```

```
# ==================================================================
# PART 2: FINE-TUNE THE MODEL
# This part loads the new file and starts training.
# ==================================================================
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer, Trainer, TrainingArguments
from datasets import load_dataset

print("\n--- PART 2: Starting Fine-Tuning ---")

# --- Configuration ---
MODEL_NAME = "microsoft/DialoGPT-medium"
OUTPUT_MODEL_PATH = "./friends-chatbot-final"

# --- Load Tokenizer and Model ---
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
model = AutoModelForCausalLM.from_pretrained(MODEL_NAME)
tokenizer.pad_token = tokenizer.eos_token

# --- Load the dataset WE JUST CREATED ---
raw_datasets = load_dataset('json', data_files=FORMATTED_DATASET_FILE, split='train')

# --- Tokenize the data ---
def tokenize_function(examples):
    conversations = [''.join([turn + tokenizer.eos_token for turn in convo]) for convo in examples["conversation"]]
    tokenized_outputs = tokenizer(
        conversations,
        truncation=True,
        padding="max_length",
        max_length=256
    )
    tokenized_outputs["labels"] = tokenized_outputs["input_ids"].copy()
    return tokenized_outputs

tokenized_datasets = raw_datasets.map(tokenize_function, batched=True, remove_columns=raw_datasets.column_names)

# --- Configure and Run Training ---
training_args = TrainingArguments(
    output_dir="./friends-bot-results",
    num_train_epochs=2,
    per_device_train_batch_size=4,
    gradient_accumulation_steps=2,
    learning_rate=5e-5,
    fp16=True,
    logging_steps=100,
    save_strategy="epoch",
    report_to="none",
)
```

```
--- PART 2: Starting Fine-Tuning ---
Map: 100%                                                    58211/58211 [00:29<00:00, 2384.55 examples/s]
```

```
trainer = Trainer(model=model, args=training_args, train_dataset=tokenized_datasets)

# --- Start Training ---
print("🚀 Training is starting now...")
trainer.train()
print("🎉 Training complete!")

# --- Save the Final Model ---
trainer.save_model(OUTPUT_MODEL_PATH)
tokenizer.save_pretrained(OUTPUT_MODEL_PATH)
```

```
print(f"✅ Final model saved to '{OUTPUT_MODEL_PATH}'")
```

Show hidden output

```
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch

# Load your fine-tuned model
model_path = "./friends-chatbot-final"
tokenizer = AutoTokenizer.from_pretrained(model_path)
model = AutoModelForCausalLM.from_pretrained(model_path)

# Let's chat!
print("Your 'Friends' chatbot is ready. Type 'quit' to exit.")
chat_history_ids = None

for step in range(10): # Chat for 10 turns
    user_input = input(">> You: ")
    if user_input.lower() == 'quit':
        break

    new_user_input_ids = tokenizer.encode(user_input + tokenizer.eos_token, return_tensors='pt')
    bot_input_ids = torch.cat([chat_history_ids, new_user_input_ids], dim=-1) if step > 0 else new_user_input_ids

    chat_history_ids = model.generate(
        bot_input_ids,
        max_length=1000,
        pad_token_id=tokenizer.eos_token_id,
        do_sample=True,
        top_k=50,
        top_p=0.95,
        temperature=0.8
    )

    response = tokenizer.decode(chat_history_ids[:, bot_input_ids.shape[-1]:][0], skip_special_tokens=True)
    print(f"ChatBot: {response}")
```

```
Your 'Friends' chatbot is ready. Type 'quit' to exit.
>> You: hey
ChatBot: Hey!
>> You: how are you?
ChatBot:
>> You: quit
```

Start coding or generate with AI.