🔹 Cell 1: Install Required Libraries

```
!pip install -q bitsandbytes accelerate datasets loralib peft transformers trl

!pip install datasets
```

📌 **Explanation**:
Installs all the required libraries to run and fine-tune a QLoRA model using TinyLlama:

- bitsandbytes for 4-bit quantization.
- accelerate, transformers, trl for Hugging Face model management and training.
- loralib, peft for applying Low-Rank Adaptation (LoRA).
- datasets to load the training dataset from Hugging Face Hub.

🔹 Cell 2: Import Libraries & Load Base Model

```
import torch

from datasets import Dataset

from transformers import AutoTokenizer, AutoModelForCausalLM, TrainingArguments, Trainer, BitsAndBytesConfig

from peft import LoraConfig, get_peft_model, prepare_model_for_kbit_training
```

📌 **Explanation**:
Imports necessary modules to:

- Load the model & tokenizer
- Configure LoRA and quantization
- Setup training arguments

🔹 Cell 3: Configure 4-bit Quantization (QLoRA)

```
model_name = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"
```

```
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
)
```

📌 **Explanation**:
Loads the TinyLlama model with **4-bit quantization** using BitsAndBytesConfig, which reduces memory and speeds up training (QLoRA technique).

🔹 Cell 4: Load Pretrained Model with Quantization Config

```
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    quantization_config=bnb_config,
    device_map="auto"
)
```

📌 **Explanation**:
Downloads the pretrained TinyLlama model and applies the quantization configuration. device_map="auto" places the model across GPUs/CPU as needed.

🔹 Cell 5: Load Tokenizer & Prepare Model for LoRA

```
tokenizer = AutoTokenizer.from_pretrained(model_name, use_fast=True)
tokenizer.pad_token = tokenizer.eos_token
model.config.use_cache = False

model.gradient_checkpointing_enable()
```

```
model = prepare_model_for_kbit_training(model)
```

📌 **Explanation**:

- Loads tokenizer and adjusts padding token.
- Enables gradient_checkpointing to reduce memory.
- Prepares the model for training with LoRA + quantization.

🔹 Cell 6: Configure & Apply LoRA

```
lora_config = LoraConfig(
    r=8,
    lora_alpha=32,
    target_modules=["q_proj", "v_proj"],
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM"
)


model = get_peft_model(model, lora_config)
```

📌 **Explanation**:
Defines LoRA config targeting the attention projections (q_proj, v_proj) and applies LoRA to the model using get_peft_model.

🔹 Cell 7: Load Dataset

```
from datasets import load_dataset


dataset = load_dataset("MakTek/Customer_support_faqs_dataset",
split="train")
print("Dataset columns:", dataset.column_names)
```

📌 **Explanation**:
Loads the MakTek/Customer_support_faqs_dataset from Hugging Face, which contains support-related questions and answers.

🔹 Cell 8: Format Instructions as Prompt-Response Pairs

```
def format_instruction(example):
    return f"### Instruction:\n{example['question']}\n\n### Response:\n{example['answer']}"


dataset = dataset.map(lambda x: {"text": format_instruction(x)})
```

📌 **Explanation**:
Reformats each row of the dataset into an instruction-tuned format (helpful for chat-based models):

```
### Instruction:
<user_question>


### Response:
<model_answer>
```

🔹 Cell 9: Tokenize the Dataset

```
def tokenize_function(example):
    tokenized = tokenizer(example["text"], truncation=True, padding="max_length", max_length=512)
    tokenized["labels"] = tokenized["input_ids"].copy()
    return tokenized


tokenized_dataset = dataset.map(tokenize_function, batched=True)
```

📌 **Explanation**:
Converts text into tokens for model training, with labels matching input_ids so that it can learn to generate the full sequence.

🔹 Cell 10: Define Training Arguments

```
training_args = TrainingArguments(
    output_dir="./tinyllama-qlora-support-bot",
    per_device_train_batch_size=2,
    gradient_accumulation_steps=4,
    learning_rate=2e-4,
    logging_dir="./logs",
    num_train_epochs=3,
    logging_steps=10,
    save_total_limit=2,
    save_strategy="epoch",
    bf16=True,
    optim="paged_adamw_8bit"
)
```

📌 **Explanation**:
Sets training config:

- 3 epochs
- 2 batch size
- Gradient accumulation for effective larger batch size
- 8-bit optimizer (paged_adamw_8bit)
- BF16 support if available (saves memory)

🔹 Cell 11: Start Training

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset,
    tokenizer=tokenizer
)

trainer.train()
```

📌 **Explanation**:
Creates a Trainer object and begins model fine-tuning on the tokenized dataset.

🔹 Cell 12: Save the Fine-Tuned Model

```
model.save_pretrained("tinyllama-qlora-support-bot")
tokenizer.save_pretrained("tinyllama-qlora-support-bot")
```

📌 **Explanation**:
Stores your trained model and tokenizer locally for future use or deployment.

🔹 Cell 13: Test the Model Locally

```
from transformers import pipeline

pipe = pipeline("text-generation", model=model, tokenizer=tokenizer)

instruction = "how can i request refund?"
prompt = f"### Instruction:\n{instruction}\n\n### Response:\n"

output = pipe(prompt, max_new_tokens=100)
print(output[0]['generated_text'])
```

📌 **Explanation**:
Creates a test pipeline to generate responses for a sample instruction using the fine-tuned model.

🔹 Cell 14: Deploy as Gradio Chat UI

```
import gradio as gr


def generate_response(instruction):
    prompt = f"### Instruction:\n{instruction}\n\n### Response:\n"
    output = pipe(prompt, max_new_tokens=100, do_sample=True, temperature=0.7)
    return output[0]['generated_text'].replace(prompt, "").strip()


gr.Interface(
    fn=generate_response,
    inputs=gr.Textbox(lines=3, placeholder="Ask your customer support question here..."),
    outputs=gr.Textbox(lines=6),
    title="🛠️ Customer Support Chatbot (TinyLlama + QLoRA)",
    description="Ask any support question. Model trained on MakTek/Customer_support_faqs_dataset using TinyLlama 1.1B."
).launch()
```