Task 1: Build and Deploy a Domain-Specific Chatbot

1) Generated 60 pairs of human-assistant conversational data.
2) Transform the pair into JSON format and stored it into JSON file.
3) Performed fine tunning on pre-trained model – "google/gemma-3-270m"
4) About Model :
   It is a 270 million parameter AI model designed for efficient, task-specific fine-tunning and strong instruction-following capabilities.
   The model is been selected because it is light weight and can easily be compute on free GPU available in google collab.
5) Fine-Tuning Approach for Causal Language Model (LoRA + 4-bit Quantization)

   1. Quantization and Efficiency Setup

      - Used BitsAndBytes 4-bit quantization (bnb_config) for memory-efficient model loading.
      - Configured compute dtype (float16 or bfloat16) based on GPU capability to optimize training efficiency.
      - Optional nested quantization enabled for improved performance with reduced memory footprint.
      - GPU capability checked to leverage bfloat16 (bf16) if supported (Ampere GPUs and above).

   2. Base Model Loading

      - Loaded a pretrained causal language model (AutoModelForCausalLM) with quantization applied.
      - Device mapping (device_map) allows multi-GPU or single-GPU deployment.
      - Disabled model caching (use_cache=False) to save memory during fine-tuning.

   3. LoRA (Low-Rank Adaptation) Configuration

      - Applied LoRA to efficiently fine-tune large models with fewer trainable parameters.
      - Configured parameters:
        - r: Rank of LoRA matrices.
        - lora_alpha: Scaling factor for LoRA updates.
        - lora_dropout: Dropout rate to improve generalization.
      - Task type set to CAUSAL_LM for language modeling.
      - Bias set to "none" to reduce overhead.

   4. Training Setup (Supervised Fine-Tuning)

      - Used Hugging Face TrainingArguments to define:

        - Number of epochs, batch size, and gradient accumulation steps.
        - Optimizer choice and learning rate scheduling (lr_scheduler_type).
        - Mixed precision (fp16) and optional bf16 support for speed and memory efficiency.
        - Gradient clipping (max_grad_norm) to stabilize training.
        - Warmup ratio and weight decay for optimizer regularization.
        - TensorBoard logging for monitoring metrics.
    - Training checkpoints saved at regular intervals (save_steps).

5. Trainer Initialization
    - SFTTrainer used to manage supervised fine-tuning.
    - LoRA configuration (peft_config) integrated into trainer for low-rank updates.
    - Training dataset supplied (dataset['train']) in Hugging Face Dataset format.
6. Training Execution
    - Fine-tuning initiated via trainer.train().
    - Efficient on-device memory usage due to 4-bit quantization + LoRA.
    - Model parameters mostly frozen except LoRA layers, reducing computational cost.
7. Advantages of This Approach
    - Allows fine-tuning large models on limited GPU memory.
    - LoRA reduces trainable parameters, making fine-tuning faster and cheaper.
    - Mixed precision and quantization enhance training efficiency without significant loss in performance.
    - Flexible to different GPUs and scales easily for larger datasets.
8. Save the Model for future use.

6) Training Loss

```
TrainOutput(global_step=15, training_loss=2.58641357421875, metrics={'train_runtime': 12.0721, 'train_samples_per_second': 4.97,
'train_steps_per_second': 1.243, 'total_flos': 1539832277760.0, 'train_loss': 2.58641357421875, 'entropy': 2.634469292561213, 'num_tokens':
2485.0, 'mean_token_accuracy': 0.5077415764331817, 'epoch': 1.0})
```