AGENTIC AI LABORATORY

Presented by:
Yashwi Pandey

Under the supervision of
Mr. Ayush Kumar Singh

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

SHARDA SCHOOL OF ENGINEERING AND TECHNOLOGY

GREATER NOIDA

Fine-Tuning a Small Language Model (SLM) Using LoRA and 4-bit Quantization

1. Introduction

This project demonstrates how to fine-tune a Small Language Model (SLM) efficiently using modern parameter-efficient fine-tuning techniques. The base model used is TinyLlama-1.1B-Chat, and it is fine-tuned on a dataset of English quotes.

To reduce memory usage and computational cost, the project applies:

- **4-bit quantization (QLoRA)**
- **LoRA (Low-Rank Adaptation)**
- **Supervised Fine-Tuning (SFT) using the TRL library**

The final model is evaluated using both qualitative text generation and quantitative metrics such as loss and perplexity.

2. Library Installation

This cell installs all required libraries:

- **PyTorch – Core deep learning framework**
- **Transformers – Pretrained models and tokenizers**
- **Datasets – Loading and processing datasets**
- **PEFT – Parameter-Efficient Fine-Tuning (LoRA)**
- **BitsAndBytes – 4-bit and 8-bit quantization**
- **TRL – Training utilities for language models**

These libraries together enable efficient fine-tuning on limited hardware.

3. Imports and Configuration:

```
import torch
from datasets import load_dataset
from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig
from peft import LoraConfig
from trl import SFTTrainer
```

Model and Dataset Selection:

```
model_name = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"
dataset_name = "Abirate/english_quotes"
new_model = "TinyLlama-1.1B-Quotes-FinteTuned"
```

Base Model: TinyLlama (1.1 billion parameters)

Dataset: English quotes dataset

Output Model: Name used to save the fine-tuned model

4. Dataset Loading

```
dataset = load_dataset(dataset_name, split="train")
```

The dataset is loaded using the Hugging Face datasets library.
Each training sample contains a quote, which is used as the text for supervised fine-tuning.

5. 4-bit Quantization Configuration

Purpose of Quantization

- **Reduces memory usage dramatically**
- **Enables training large models on consumer GPUs**

Key Settings

- **NF4: Normal Float 4-bit quantization**
- **float16 computation: Improves speed and stability**
- **QLoRA approach: Quantized base model + trainable LoRA adapters**

6. Loading the Base Model

```
model = AutoModelForCausalLM.from_pretrained(
   model_name,
   quantization_config=bnb_config,
   device_map={"": 0}
)
```

Loads the pretrained TinyLlama model

Applies 4-bit quantization

Automatically places the model on GPU

Additional configuration:

```
model.config.use_cache = False
```

model.config.pretraining_tp = 1
Disabling cache avoids training issues

Ensures compatibility with LoRA fine-tuning

7. Tokenizer Setup

- **Uses the same tokenizer as the base model**
- **Sets padding token to EOS token (required for causal LM training)**
- **Right padding is optimal for transformer models**

8. LoRA (Low-Rank Adaptation) Configuration

Why LoRA?

- **Only a small number of parameters are trained**
- **Base model weights remain frozen**
- **Faster training and lower memory usage**

Key Parameters

- **r = 64: Rank of LoRA matrices**
- **alpha = 16: Scaling factor**
- **dropout = 0.1: Regularization**

9. Supervised Fine-Tuning (SFT) Configuration

Training Strategy

- **SFT (Supervised Fine-Tuning): Model learns directly from labeled text**
- **Single epoch: Sufficient for demonstration purposes**
- **Batch size of 4: Balanced for memory and speed**

Additional settings control:

- **Logging frequency**
- **Optimizer (`paged_adamw_32bit`)**
- **Gradient stability**
- **Learning rate scheduling**

10. Trainer Initialization and Training

The `SFTTrainer`:

- **Handles tokenization**
- **Applies LoRA adapters**
- **Manages training loop**

11. Saving the Fine-Tuned Model

The fine-tuned LoRA-adapted model is saved locally for future inference or deployment.

12. Qualitative Evaluation (Inference)

The model is tested using a natural language prompt to evaluate:

- **Coherence**
- **Relevance**
- **Fluency of generated quotes**

This provides a human-interpretable evaluation of training quality.

13. Quantitative Evaluation: Loss and Perplexity

Loss Extraction

python

```python
log_history = trainer.state.log_history
```

- Training logs are parsed to extract loss values
- Final loss represents model convergence

Perplexity Calculation

python

```python
perplexity = exp(final_loss)
```

- Lower perplexity indicates better language modeling
- Measures how "surprised" the model is by the text

14. Training Loss Visualization

A loss curve is plotted over training steps
Helps verify stable learning and convergence

15. Final Text Generation Test

The final generation test confirms:

- **The model has learned stylistic patterns of quotes**
- **The output is meaningful and contextually appropriate**

16. Conclusion

This project successfully demonstrates:

- **Fine-tuning a Small Language Model using LoRA**
- **Efficient training with 4-bit quantization**
- **Practical evaluation using loss, perplexity, and text generation**

The approach is scalable, memory-efficient, and suitable for real-world NLP applications where computational resources are limited.