Reporte: Fine-Tuning

1. Resumen

El objetivo principal fue realizar un **Fine-Tuning** sobre la arquitectura **Llama 3 8B** para especializarlo como tutor de Python, utilizando un dataset personalizado.

El desafío central consistió en ejecutar este flujo de trabajo en una sola GPU RTX 3060 y 16GB de RAM, superando limitaciones físicas de memoria mediante cuantización (**QLoRA**) y gestión de memoria virtual (**Swap/Offloading**). El resultado final es un modelo portable en formato **GGUF** capaz de ejecutarse localmente en una laptop.

2. Especificaciones Técnicas y Stack Tecnológico

2.1 Hardware de Entrenamiento

- **Procesador:** AMD Ryzen 7 5700G
- **GPU:** NVIDIA GeForce RTX 3060 (12GB VRAM)
- **RAM:** 16GB DDR4
- **Sistema Operativo:** Windows 11 con subsistema WSL.

2.2 Stack de Software

- **Framework de Entrenamiento: Axolotl**. Seleccionado por su capacidad de orquestar configuraciones complejas de entrenamiento mediante archivos YAML y su soporte nativo para QLoRA.
- **Motor de Inferencia: Ollama**. Utilizado para desplegar el modelo final.
- **Librerías Críticas:**
    - PyTorch: Motor de cálculo tensorial.
    - bitsandbytes: Para la cuantización de 4-bits.
    - PEFT (Parameter-Efficient Fine-Tuning): Gestión de adaptadores LoRA.
    - llama.cpp: Conversión de tensores y cuantización final a GGUF.

2.3 Dataset

- **Formato:** JSONL estilo "Alpaca".
- **Estructura:** Tripletas de {instruction, input, output}.
- **Contenido:** Ejemplos didácticos de programación en Python, diseñados para enseñar conceptos fundamentales con brevedad.

3. Metodología Implementada: QLoRA

Dado que el modelo base (Llama 3 8B) requiere aproximadamente 16GB de VRAM solo para cargarse en precisión media (FP16), y mucho más para entrenarse, se utilizó la técnica **QLoRA**.

1. **Cuantización 4-bit:** El modelo base se carga congelado en precisión de 4 bits, reduciendo su huella de memoria a aprox. 6GB.
2. **Low-Rank Adapters (LoRA):** Se inyectan pequeñas matrices entrenables en las capas de atención del modelo. Solo se actualizan los pesos de estas matrices (menos del 1% de los parámetros totales), permitiendo que el entrenamiento quepa en los 12GB de la RTX 3060.

4. Desafíos Técnicos y Soluciones

Durante el ciclo de desarrollo, se encontraron y resolvieron cuellos de botella críticos relacionados con la gestión de recursos y compatibilidad de software.

4.1 "Dependency Hell" (Conflicto de Versiones)

- **Problema:** Incompatibilidad entre `torchvision` (requería PyTorch 2.9.1) y `Axolotl` (optimizado para versiones anteriores o específicas de CUDA). Esto impedía el inicio del entrenamiento.
- **Solución:** Reconstrucción del entorno virtual (`venv`), forzando la instalación sincronizada de `torch`, `torchvision` y `torchaudio` antes de instalar las dependencias de Axolotl, y desactivación de `Flash Attention 2` en favor de SDPA nativo.

4.2 Saturación de VRAM (OOM - Out of Memory)

- **Problema:** Al iniciar el entrenamiento, el consumo de VRAM excedía los 12GB físicos, provocando el uso de memoria compartida (lenta) o fallos directos.
- **Solución:** Optimización de hiperparámetros en `config.yml`:
    - Reducción de `micro_batch_size` a 1.
    - Aumento de `gradient_accumulation_steps` a 8 (para mantener la estabilidad matemática).
    - Exclusión de capas densas (`embed_tokens`, `lm_head`) del entrenamiento LoRA.

4.3 Colapso de RAM durante la Fusión

- **Problema Crítico:** La fusión de los adaptadores LoRA con el modelo base requiere cargar el modelo completo en FP16 (~15GB). Con solo 16GB de RAM física y el sistema operativo corriendo, el proceso era eliminado por el kernel de Linux (`Killed`) por falta de memoria.
- **Solución:**
    1. Creación de un archivo **Swap de 16GB** en Linux para extender la memoria RAM usando el disco duro.
    2. Desarrollo de un script personalizado que implementa **CPU Offloading**, forzando la carga secuencial de capas entre el disco y la RAM para evitar el desbordamiento.

5. Resultados y Optimización Final

5.1 Entrenamiento

El modelo completó 50 épocas de entrenamiento. La curva de pérdida (**Training Loss**) mostró una reducción consistente desde valores iniciales de ~3.5 hasta converger por debajo de 0.8, indicando un aprendizaje efectivo de los patrones del dataset.

5.2 Optimización para Despliegue (GGUF)

Para permitir la ejecución en una laptop sin GPU dedicada potente, se realizó una conversión final:

1. **Conversión:** De tensores PyTorch FP16 a formato GGUF.
2. **Cuantización:** Se aplicó el método **`q4_k_m`**.
    - *Tamaño original:* ~15 GB.
    - *Tamaño final:* ~5.7 GB.

5.3 Validación

Se realizaron pruebas A/B comparando Llama 3 Base vs. Llama 3 Fine-Tuned con temperatura 0.0. El modelo ajustado demostró:

- Adherencia estricta al formato de respuesta breve solicitado.

6. Conclusión

Este proyecto demuestra la viabilidad de entrenar y desplegar modelos de lenguaje de última generación utilizando recursos limitados. A través de una combinación de técnicas de eficiencia algorítmica (QLoRA) y gestión de sistemas (Swap, Offloading), se logró crear un asistente especializado funcional, portátil y eficiente..

Anexos: Axolotl YAML

```yaml
base_model: NousResearch/Meta-Llama-3-8B
adapter: qlora
load_in_4bit: true
sequence_len: 1024
sample_packing: true
lora_r: 32
lora_alpha: 16
lora_target_linear: true
micro_batch_size: 1
gradient_accumulation_steps: 8
num_epochs: 50
optimizer: adamw_bnb_8bit
learning_rate: 0.0002
```