

INSTITUTO TECNOLÓGICO DE MORELIA DIVISIÓN DE ESTUDIOS PROFESIONALES

INGENIERÍA EN SISTEMAS COMPUTACIONALES

“Reporte proyecto 4: Tutor por fine tuning”

Presenta:

Morales Cazares Alberik Said

Nombre del Profesor:

Alcaraz Chávez Jesús Eduardo

19 de diciembre del 2025

Introducción

En el presente trabajo se trabajara con fine tuning un ajuste fino, que se implementa sobre el modelo de LLM “Llama3.2:3b” para actuar no como un generador de código genérico, sino como un tutor pedagógico estructurado. La justificación reside en la capacidad del modelo adaptado para replicar una metodología de enseñanza específica: desglosar problemas complejos mediante analogías del mundo real, explicaciones paso a paso y análisis de complejidad asintótica, ofreciendo una interacción personalizada y paciente que los recursos estáticos no pueden proveer.

El presente proyecto es debido a una barrera para los nuevos juniors que se unen a estudiar programación, y siente difícil cuando empiezan a querer programar. La enseñanza tradicional a menudo carece de la granularidad necesaria para explicar la "intuición" detrás de una solución antes de presentar el código, lo que resulta en una memorización sintáctica sin comprensión lógica

Alcance

Este sistema está pensado únicamente como una ayuda para aprender algoritmos. No es una herramienta para programar rápido, sino para entender bien lo que se está haciendo. Funciona como un tutor que acompaña el proceso de razonamiento: parte de la idea inicial del problema, explica cómo se llega a una solución y muestra cómo llevarla a código en Python de forma clara.

Planteamiento del problema

Aprender a programar representa un problema para gran parte de los de estudiantes. Los estudiantes novatos frecuentemente intentan memorizar la sintaxis de una solución sin comprender la lógica ni los patrones, como se evidencia en la necesidad de desglosar problemas complejos mediante analogías y pasos explícitos, lo que resulta en que aprender a programar resulte de manera superficial, y no logren adaptar códigos a alcance de lo exigido.

La implementación de un tutor que utilice fine-tuning, surge como algo viable para dar mentoría y guiar a los nevos de manera paciente sin tener que recurrir a los profesores cuando no cuentan con el tiempo. A diferencia de los modelos genéricos que priorizan la entrega inmediata de código, este sistema se entrena específicamente para emular un rol docente: primero establece una intuición clara, luego explica el razonamiento paso a paso, y finalmente presenta el código junto con su análisis de complejidad

Objetivos

Desarrollar un asistente inteligente especializado en la tutoría de algoritmos y estructuras de datos, mediante fine-tuning del modelo de lenguaje Llama 3.2:3B especializándolo para generar explicaciones de programación en algoritmos.

Objetivos específicos:

Diseñar y obtener un dataset para entrenar el modelo

- Realizar un dataset con preguntas y sus respectivas respuestas guiadas paso a paso
- Dar formato al dataset en formato JSONL

Configuración del Entorno de Adaptación

- Implementar la arquitectura de entrenamiento con Unsloth.
- Configurar adaptadores LoRA

Procesamiento de datos

- Normalizar el dataset
- Dar el formato a las instrucciones para que lo comprenda la maquina.

Ejecución del Ajuste Fino

- Entrenar el modelo mediante Supervised Fine-Tuning con 3 épocas.
- Brindar una personalidad al modelo, para s¿que se asemeje a un tutor.
- Aplicar técnicas de optimización (LoRA, cuantización)
- Convertir el modelo ajustado a formato GGUF
- Integrar el modelo final en Ollama mediante un Modelfile

Marco Teorico

Modelo LLM y Fine Tuning

Los modelos mas grandes de Lenguajes (LLM), como la que usaremos Llama 3.2, funcionan como redactores probabilísticos, que se entrenan con una gran cantidad de corpus. Pero en casos más especializados es necesario realizar fine-tuning que ajusta los pesos del modelo utilizando pares de "instrucción-respuesta", condicionando al sistema para seguir especificaciones en lugar de simplemente completar texto. Aquí es donde entran los adaptadores de bajo rango como LoRA que permite una adaptación eficiente sin los costos computacionales del reentrenamiento total.

Tutor inteligente y aprendizaje guiado

Busca replicar la interacción personalizada, como si se tratase de un docente humano, la configuración del modelo no solo para entregar información correcta, sino para hacerlo con un tono que faciliten el aprendizaje, tal como se define en el prompt que se le pasa.

Metodología

Alcances

El alcance del tutor de limita a la resolución de y explicación de problemas de algoritmos como arrays, strings, arboles, etc. Durante este se priorizó la comprensión lógica por eso se seleccionó Llama 3.2 en su variante de 3 billones de parámetros. Lo que a pesar de ser algo pesada, está en un equilibrio entre el razonamiento y eficiencia, más que nada eficiencia permitiendo el entrenamiento en entornos con recursos limitados.

Tecnologías utilizadas

Modelos usados:

- Llama 3.2 con 3B de parámetros
- En la cuantización utiliza 4-bit para reducir el consumo de memoria.

Librerías usadas y formatos:

- Unsloth: Sirve para acelerar el fine tuning del modelo.
- Pytorch
- Hugging Face Ecosystem: Para la carga y formateo de los datos, como gestión de los procesos.
- Ollama
- LoRA

Diseño del Dataset Educativo

Se construyó un corpus de entrenamiento especializado siguiendo un esquema de diálogo estructurado. Cada entrada del dataset fue diseñada manualmente para incluir:

- **Rol del sistema:** darle una personalidad al tutor, en este caso un tutor de algoritmos que es paciente.
- **Consulta del usuario:** Preguntas que tal vez queda preguntar.
- **Respuesta del modelo:** una estructura fija que explica, brinda código y dice la complejidad del algoritmo.

Procesamiento de los datos

Los datos se encuentran en un formato JSONL, fueron normalizados para garantizar la estructura y compatibilidad del pipeline de entrenamiento, se implementó una función que aplanaba la estructura de lista de mensajes en un formato de instrucción lineal compatible con modelos de LLM.

Pipeline

Se ejecutó utilizando la técnica de LoRA para modificar los pesos del modelo sin reentrenar la totalidad de la red. La configuración del pipeline incluyó:

- Rango: 64, para asegurar suficiente capacidad de aprendizaje
- Escalado: 128, para amplificar la influencia de los nuevos pesos.
- Módulos objetivos: Aplicar los adaptadores en este caso son q_proj, k_proj, v_proj, o_proj.
- Epocas: 3 ciclos.
- Longitud de la secuencia: Se configuró con un máximo de 2028 tokens para explicaciones extensas.

Aquí podemos ver la configuración, donde se configuran ciertos parámetros

```
trainer = SFTTrainer(  
    model=model,  
    tokenizer=tokenizer,  
    train_dataset=dataset,  
    dataset_text_field="text",  
    max_seq_length=max_seq_length,  
    dataset_num_proc=2,  
    args=TrainingArguments(  
        per_device_train_batch_size=2,  
        gradient_accumulation_steps=4, # Effective batch size = 8  
        warmup_steps=10,  
        num_train_epochs=3,  
        learning_rate=2e-4,  
        fp16=not torch.cuda.is_bf16_supported(),  
        bf16=torch.cuda.is_bf16_supported(),  
        logging_steps=25,  
        optim="adamw_8bit",  
        weight_decay=0.01,  
        lr_scheduler_type="linear",  
        seed=3407,  
        output_dir="outputs",  
        save_strategy="epoch",  
        save_total_limit=2,  
        dataloader_pin_memory=False,  
        report_to="none", # Disable Weights & Biases logging  
    ),  
)
```

Aquí la configuración del Modelfile

```
FROM llama3.2:2b  
ADAPTER ./Llama-3.2-3B.Q4_K_M.gguf  
PARAMETER temperature 0.7  
SYSTEM "Eres un Tutor Experto en Algoritmos. Explica paso a paso"
```

Exportar GGUF

Después del entrenamiento, los adaptadores LORA se fusionan con el modelo base, dando como resultado una exportación a GGUF para ejecutarlo localmente, y finalmente se aplicó el Modelfile para que pueda interactuar con Ollama, estableciendo una temperatura de **0.7** para balancear creatividad y precisión, además en el Modelfile se fija el prompt para mantener las respuestas a manera de tutorial.

#Exportacion a GGUF

```
model.save_pretrained_gguf("gguf_model", tokenizer, quantization_method="q4_k_m")
```

Diseño del dataset

La construcción del dataset no consiste solo en la acumulación de ejercicios y respuesta a conceptos, sino en la capacidad de ingresar capacidades de razonamiento en el modelo. El corpus se compuso de diversos algoritmos para asegurar que pueda generalizar conceptos a través de diferentes niveles de abstracción.

Datos y Estructura

El dataset se compone de pares de instrucciones y su respuesta. Los datos se clasifican en:

- **Consulta:** La solicitud sobre el ejercicio o algoritmo.
- **Respuestas estructuradas:** Salidas generadas que siguen estrictamente el formato: Título -> Intuición -> Explicación -> Código -> Complejidad.

El dataset incluye ejemplos para abordar varios problemas de pensamiento computacional, un ejemplo de estos es el siguiente:

```
{"messages": [  
  {  
    "role": "system",  
    "content": "Eres un tutor de algoritmos, paciente y que explicas por pasos."  
  },  
  
  {  
    "role": "user",  
    "content": "Necesito ayuda para entender el algoritmo Smallest Value After Replacing With  
Sum of Prime Factors"},  
  {  
    "role": "assistant",
```



```

        "content": "***Intuición:***\nImagina que tienes un número grande y  
quieres encontrar sus factores primos ..."  
    }  
]  
}  
.
```

Resultado y Evaluación del Modelo

El modelo fue entrenado con aproximadamente 700 ejemplos cuidadosamente seleccionados. La evaluación no se limitó a verificar la correcta ejecución del proceso de fine-tuning, sino que se centró en determinar si el sistema cumple efectivamente su función como tutor de algoritmos. Para ello, se analizó la forma en que el modelo estructura sus explicaciones, mantiene un orden lógico en las respuestas y alinea sus resultados con los objetivos pedagógicos definidos en el dataset.

Para evaluar esto, se usaron criterios simples pero claros: que el modelo siga siempre la misma estructura al responder, que sus explicaciones ayuden a entender el problema sin confundir, y que el código y su análisis sean coherentes con la solución.

Para evaluar el modelo, se revisaron las respuestas generadas y se verificó que estas fueran claras, ordenadas y relacionadas directamente con el problema planteado, así como que el código presentado correspondiera a la explicación dada.

Dentro de la captura se hace la pregunta: “Necesito ayuda para entender el algoritmo Smallest Value After Replacing With Sum of Prime Factors” la cual es la primera pregunta del dataset, responde correctamente alineado con la respuesta, el orden de ejecución es correcto, al principio te dice un pequeño texto a manera de explicación, para después pasar enseñarte el código en lenguaje natural paso a paso, después empezó a generar el código correctamente con algunos errores, ya mas a manera de lógica o escritura en algún punto o similar, pero coincide casi en su totalidad con lo esperado, finalmente termina diciendo la complejidad del algoritmo, como su diciendo que la respuesta que pediste ya te la explico arriba.

En resumen, la respuesta si es correcta, elabora todos los pasos correctamente, la lógica si es la correcta alineado con el fine tuning realizado, la manera en que concluye es corta, sensata y fácil de entender claramente con conocimiento básico. La manera de explicar si es paso a paso incluso hasta con pseudocódigo y ejemplos.

Implementación del tutor

El tutor entrenado fue integrado en una interfaz sencilla de tipo conversacional, similar a un chat, cuyo objetivo principal es permitir al estudiante realizar preguntas sobre algoritmos y recibir explicaciones guiadas. La interfaz no prioriza aspectos gráficos avanzados, sino la claridad de las respuestas y la facilidad de interacción.

El tutor fue integrado mediante la terminal, aunque si ingresamos a ollama el modelo nos debe aparecer a la hora de buscarlo, en caso de encontrarlo en las opciones, debemos seguir los pasos, una vez tengamos el archivo GGUF:

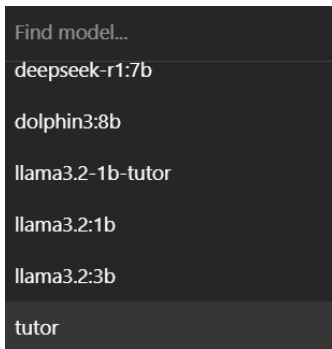
Dentro de la terminal debemos situarlo en la ubicación de nuestro Modelfile

Ollama create <nombre modelo nuevo> -f Modelfile

Una vez cargado podemos llamarlo si escribimos o bien ingresando a ollama y cargando el modelo que creamos

Ollama run <nombre del modelo>

En ollama se mostrará de esta manera, solo debemos seleccionarlo



El estudiante puede escribir preguntas de forma libre, ya sea solicitando la explicación de un algoritmo, la resolución de un ejercicio o el análisis de un fragmento de código. El tutor responde siguiendo una estructura progresiva, iniciando con una explicación conceptual y avanzando hacia ejemplos y pseudocódigo cuando el problema lo requiere. Para un mejor aprovechamiento del sistema, se recomienda que las preguntas sean claras y específicas, por ejemplo, solicitando la explicación paso a paso de un algoritmo

```

PS D:\sai\Desk\InteligenciaArtificial\Class-Repo\tarea22 - PROYECTO - finetuning> ollama run tutor "Necesito ayuda para entender el algoritmo Smallest Value After Replacing With Sum of Prime Factors"
**Intuición:** Imagine una caja con números escritos sobre ella, como si estuvieras ordenando un libro. Quieres saber cuántos números puedes obtener sumándole uno a uno los números más grandes que hayas encontrado hasta ahora.
**Explicación:**
El algoritmo 'smallest_value_after_replacing' utiliza una técnica llamada "reemplazo con suma" para encontrar el valor mínimo después de reemplazar cada número por la suma de sus primos factores. La idea es comparar este nuevo valor con el anterior valor mínimo encontrado hasta ahora.

1. **Precondición:** El arreglo 'nums' debe estar lleno y no está vacío.
2. **Procedimiento:**
   * Si la longitud del arreglo es cero, significa que hemos terminado reemplazar todos los números en 'nums', por lo que devolvemos el valor mínimo encontrado hasta ahora.
   * Recorremos cada número en el arreglo 'nums'. Para cada uno de ellos:
     + Calculamos su suma de primos factores ('sum_of_primes') utilizando la fórmula  $(2^{**k}) - 1$  donde 'k = log2(n)' es el factor logarítmico del número 'n'.
     + Si la longitud actual del arreglo ('prev_len') es menor que el largo de este paso (cuyo valor se obtiene mediante llamada a 'len(nums) - sum_of_primes'), entonces actualizamos el valor previo 'prev_sum' con la suma total de los números en el arreglo hasta ahora. Esto lo hace posible porque si un número se reemplaza, todos los números posteriores al mismo ya están incluidos.
     + Luego, calculamos el nuevo valor mínimo ('new_min') como 'sum_of_primes - 1' (esto es porque la suma de primos factores es siempre positiva).
     + Devolvemos este nuevo valor menor o igual a 'prev_sum'.

**Código:**
```python
def encontrar_valor_mínimo(nums):
 # Precondición: El arreglo 'nums' debe estar lleno y no está vacío.

 # Comprueba si el arreglo es vacío, lo cual significa que hemos terminado reemplazar todos los números en nums,
 if not nums:
 return 0

 # Recorrer cada número del arreglo
 for num in nums:
 # Calcula la suma de primos factores (sum_of_primes) utilizando fórmula $(2^{**k}) - 1$.
 sum_of_primes = ((2**len(str(num))) - 1)

```

```

 # Calcula la suma de primos factores (sum_of_primes) utilizando fórmula $(2^{**k}) - 1$.
 sum_of_primes = ((2**len(str(num))) - 1)

 # Si la longitud actual del arreglo es menor que el largo de este paso,
 # entonces actualizamos prev_sum con suma total de los números en el arreglo hasta ahora.
 if len(nums) < sum_of_primes:
 prev_len = nums[0:len(nums)]

 for i in range(len(prev_len)):
 prev_sum += int(prev_len[i])

 return sum_of_primes - 1

 # Actualiza prev_sum con suma total de los números en el arreglo hasta ahora.
 prev_sum = sum(int(num) for num in nums if int(num) != 0)

 # Calcula nuevo valor mínimo como (sum_of_primes) - 1.
 new_min = sum_of_primes - 1

 return min(prev_len, prev_sum)

Ejemplo de uso:
print(encontrar_valor_mínimo([1,2,3]))
```

**Complejidad:** La complejidad del algoritmo es  $O(n^2)$ , donde n es la longitud del arreglo 'nums'. Esto se debe a que recorremos cada número en el arreglo dos veces: una vez para calcular la suma de primos factores y otra para actualizar prev_sum con la suma total de los números hasta ahora.

**Nota:** El código anterior utiliza una solución técnica llamada "reemplazo con suma" para encontrar el valor mínimo después de reemplazar cada número por la suma de sus primos factores. Si estás interesado en aprender más sobre este algoritmo o otras técnicas similares, ¡no dudes en preguntar! ¡Sé su amigo experto y tutor inteligente!<reserved_special_token_176>иначасуер

**Tutor Responsable:** El tutor responsable es un profesional experto en algoritmos que se especializa en la solución de problemas técnica. Si tienes alguna pregunta o necesitas más ayuda, no dudes en preguntar.дивидувājiciassistant

```

Discusión

Durante el desarrollo del proyecto se observaron resultados positivos en la capacidad del modelo para explicar algoritmos de forma estructurada y comprensible, especialmente en problemas bien definidos. El fine-tuning permitió mejorar la coherencia de las explicaciones.

Sin embargo, el sistema presenta limitaciones. En algunos casos, el modelo genera errores menores en el código o explicaciones incompletas cuando el problema se vuelve más complejo o se desvía del contenido presente en el dataset. Esto no indica que las respuestas dependen en gran medida de que el tema abarcado se encuentre dentro del dataset.

Hay riesgos que se encuentra la posibilidad de que el modelo genere respuestas incorrectas con un tono convincente, lo que podría inducir a errores si el estudiante no verifica la información. además, el modelo puede mostrar resultados mas dirigidos hacia los tipos de problemas más representados en el dataset.

Conclusiones

Durante el desarrollo del proyecto, nos permitió entrenar un modelo de lenguaje, esta vez con la capacidad de responder preguntas de programación, como elaborar código sobre algoritmos, en base a un dataset que fue introducido en un modelo mediante fine tuning. Los objetivos planteados se cumplieron en términos de diseño del dataset, entrenamiento del modelo y evaluación de su comportamiento como tutor.

Este proyecto confirma que, incluso con un conjunto de datos limitado, es posible adaptar un modelo de lenguaje para cumplir funciones educativas específicas, siempre que el proceso esté guiado por criterios claros.

Trabajo futuro

Como trabajo futuro se planea incluir una cantidad mayor de algoritmos y estructuras de datos. Esto nos permitiría mejorar los temas que no se han cubierto o no están muy bien integrados por falta de datos.

Asimismo, podría explorarse la adaptación del tutor a distintos estilos de enseñanza o niveles de dificultad, así como su integración en plataformas educativas más completas. Estas extensiones permitirían evaluar el potencial del sistema en contextos reales de enseñanza y aprendizaje.

Referencias

Bhargava, A. Y. (2016). *Grokking algorithms: An illustrated guide for programmers and other curious people*. Manning Publications.

Miller, B. N., & Ranum, D. L. (2011). *Problem solving with algorithms and data structures using Python* (2.^a ed.). Franklin, Beedle & Associates.