IACADEMY

DE FUNDAMENTOS A ARQUITECTURA DE IA

MÓDULO 25

Fine-tuning: cuando y como

Avanzado

iacedemy.com — 2026

MÓDULO 25

Fine-tuning: cuando y como

Nivel: Avanzado

Autor: David Moya

Publicación: Mayo 2026

Plataforma: iacedemy.com

Que es fine-tuning realmente

Fine-tuning es el proceso de tomar un modelo de lenguaje pre-entrenado y ajustar sus pesos con datos especificos de tu dominio. No es entrenar desde cero (eso cuesta millones). Es adaptar un modelo existente para que sea mejor en una tarea concreta.

Piensa en ello como la diferencia entre contratar a un generalista y formar a alguien en tu empresa. El modelo base ya sabe escribir, razonar y seguir instrucciones. Con fine-tuning le ensenas tu estilo, tu terminologia, tu formato de salida exacto.

El resultado es un modelo que responde como quieres sin necesitar prompts largos con 20 ejemplos few-shot. Eso reduce latencia, reduce costes de tokens y mejora la consistencia. Pero no es magia: necesitas datos de calidad y un proceso sistematico.

Tipos de fine-tuning

Existen tres enfoques principales:

- **Supervised Fine-Tuning (SFT):** Le das pares input/output y el modelo aprende a replicar ese patron. Es el mas comun.
- **RLHF (Reinforcement Learning from Human Feedback):** Usas preferencias humanas para alinear el modelo. Complejo y caro. Solo para equipos grandes.
- **DPO (Direct Preference Optimization):** Alternativa a RLHF sin necesidad de un reward model separado. Mas accesible pero aun avanzado.

Para el 95% de los casos practicos, SFT con datos de calidad es todo lo que necesitas. Ese es nuestro foco en este modulo.

El arbol de decision: prompting vs RAG vs fine-tuning

El error mas comun es saltar directamente a fine-tuning cuando un buen prompt o RAG resolveria el problema. Sigue este orden:

Paso 1: Intenta con prompting

Antes de cualquier otra cosa, prueba un system prompt bien estructurado con few-shot examples. En el 70% de los casos esto es suficiente. Si el modelo base (Claude, GPT-4, Qwen) ya puede hacer la tarea con un prompt de 500 tokens, no necesitas fine-tuning.

Paso 2: Intenta con RAG

Si el problema es que el modelo no tiene acceso a informacion especifica (documentos internos, productos, politicas), RAG es la solucion. No necesitas cambiar el modelo, solo darle contexto relevante en cada consulta.

Paso 3: Fine-tuning como ultimo recurso

Fine-tuning tiene sentido cuando:

- Necesitas un formato de salida muy especifico y consistente (JSON schema exacto, estilo de escritura particular)
- Quieres reducir latencia eliminando prompts largos con ejemplos
- El modelo base no entiende tu dominio incluso con contexto (jerga medica, legal, tecnica muy especializada)
- Necesitas un modelo pequeno que funcione como uno grande en tu tarea especifica
- Volumen alto: procesas miles de requests al dia y los tokens de few-shot cuestan mas que el fine-tune

> **Regla de oro**
>
> Si puedes resolver el problema con un prompt de menos de 1000 tokens y few-shot examples, NO hagas fine-tuning. El coste de mantenimiento del dataset y el pipeline no compensa.

Preparar tu dataset (ChatML format)

La calidad de tu dataset determina el 80% del resultado. No es cuestion de cantidad: 200 ejemplos excelentes superan a 10.000 mediocres. El formato estandar es ChatML, compatible con la mayoria de frameworks de fine-tuning.

Estructura ChatML

Cada ejemplo es una conversacion con roles system, user y assistant:

```
{
  "conversations": [
    {
      "role": "system",
      "content": "Eres un analista SOC que clasifica alertas en formato JSON."
    },
    {
      "role": "user",
      "content": "Alerta: Conexion SSH desde IP 185.220.101.42 al servidor prod-db-01 a las 03
    },
    {
      "role": "assistant",
      "content": "{\"severity\": \"high\", \"category\": \"unauthorized_access\", \"tactic\":
    }
  ]
}
```

Cuantos ejemplos necesitas

COMPLEJIDAD DE LA TAREA	MINIMO	IDEAL	NOTAS
Clasificacion simple	50	200-300	Etiquetas claras, poco ambiguas
Generacion estructurada	100	300-500	JSON, markdown, formatos exactos
Estilo/tono especifico	200	500-1000	El modelo necesita captar matices
Razonamiento de dominio	500	1000-5000	Necesita aprender logica nueva

Reglas para un buen dataset

- **Diversidad:** Cubre todos los edge cases que esperas en produccion. No repitas el mismo patron 50 veces.
- **Consistencia:** Las respuestas del assistant deben seguir el mismo formato siempre. Si mezclas JSON con texto libre, el modelo se confunde.
- **Calidad humana:** Cada ejemplo debe ser una respuesta que un experto daria. Si metes basura, sale basura.
- **Balance:** Si es clasificacion, intenta que las clases esten equilibradas. Si tienes 90% clase A y 10% clase B, el modelo ignorara B.

Script para validar tu dataset

```
import json
from pathlib import Path

def validate_dataset(path: str) -> dict:
    """Valida un dataset ChatML y reporta problemas."""
    data = json.loads(Path(path).read_text())
    issues = []
    stats = {"total": len(data), "valid": 0, "issues": []}

    for i, example in enumerate(data):
        convs = example.get("conversations", [])
        roles = [c["role"] for c in convs]

        # Verificar estructura basica
        if "system" not in roles:
            issues.append(f"Ejemplo {i}: sin system message")
        if "assistant" not in roles:
            issues.append(f"Ejemplo {i}: sin assistant response")
        if roles[-1] != "assistant":
            issues.append(f"Ejemplo {i}: no termina en assistant")

        # Verificar longitud
        for conv in convs:
            if len(conv["content"]) < 10:
                issues.append(f"Ejemplo {i}: contenido muy corto ({conv['role']})")

        if not any(f"Ejemplo {i}" in iss for iss in issues):
            stats["valid"] += 1

    stats["issues"] = issues
    print(f"Total: {stats['total']} | Validos: {stats['valid']} | Problemas: {len(issues)}")
    return stats

validate_dataset("dataset.json")
```

LoRA vs full fine-tuning

Full fine-tuning actualiza todos los pesos del modelo. Para un modelo de 27B parametros, necesitas ~108GB de VRAM solo para los pesos en FP32, mas gradientes y estados del optimizador. Estamos hablando de 4-8 GPUs A100 de 80GB. Inviable para la mayoria.

LoRA (Low-Rank Adaptation) resuelve esto de forma elegante: en vez de actualizar todos los pesos, inserta matrices pequenas (adapters) en capas clave del modelo. Solo entrenas esas matrices, que representan el 0.1-1% del total de parametros.

Como funciona LoRA

La idea matematica es simple. En vez de actualizar una matriz de pesos W (de dimension d x d), descompones el update en dos matrices pequenas: A (d x r) y B (r x d), donde r es el "rank" y es mucho menor que d (tipicamente 8, 16, 32 o 64).

```
# Conceptualmente:
# W_nuevo = W_original + A * B
# Donde A es (d x r) y B es (r x d)
# Si d=4096 y r=16:
#   Parametros originales: 4096 x 4096 = 16.7M
#   Parametros LoRA:       4096 x 16 + 16 x 4096 = 131K
#   Reduccion: 99.2%
```

Comparativa practica

ASPECTO	FULL FINE-TUNING	LORA (R=16)	QLORA (4-BIT)
VRAM (7B model)	~60GB	~16GB	~6GB
VRAM (27B model)	~220GB	~48GB	~18GB
Tiempo entrenamiento	Horas-dias	30min-2h	30min-2h
Calidad resultado	Optima	95-99% de full	90-97% de full
Tamano adapter	N/A (modelo completo)	10-100MB	10-100MB
Hardware minimo	4x A100 80GB	1x A100 40GB	1x RTX 4090 24GB

Para casi todos los casos, QLoRA con rank 16-32 es el sweet spot: calidad excelente, cabe en una sola GPU consumer, y entrena en menos de 2 horas.

Ejecutar un fine-tune con Unsloth

Unsloth es la herramienta de referencia para fine-tuning eficiente en 2026. Es 2-5x mas rapido que el stack clasico de HuggingFace + PEFT, usa menos memoria, y tiene una API limpia. Soporta Qwen, Llama, Mistral, Phi y la mayoria de modelos populares.

Instalacion

```
# En un entorno con CUDA instalado
pip install unsloth
# O para la version bleeding-edge:
pip install "unsloth[cu121-ampere-torch250] @ git+https://github.com/unslothai/unsloth.git"
```

Script completo de fine-tuning

```
from unsloth import FastLanguageModel
from trl import SFTTrainer
from transformers import TrainingArguments
from datasets import load_dataset

# 1. Cargar modelo base
model, tokenizer = FastLanguageModel.from_pretrained(
    model_name="unsloth/Qwen2.5-7B-Instruct",  # Qwen 3.5 27B si tienes VRAM
    max_seq_length=2048,
    dtype=None,  # auto-detect
    load_in_4bit=True,  # QLoRA
)

# 2. Configurar LoRA
model = FastLanguageModel.get_peft_model(
    model,
    r=16,                # Rank del adapter
    target_modules=["q_proj", "k_proj", "v_proj", "o_proj",
                    "gate_proj", "up_proj", "down_proj"],
    lora_alpha=16,
    lora_dropout=0,      # 0 es optimo segun papers
    bias="none",
    use_gradient_checkpointing="unsloth",  # 30% menos VRAM
)

# 3. Cargar dataset
dataset = load_dataset("json", data_files="dataset.json", split="train")

# 4. Formatear para ChatML
def format_chatml(example):
    convs = example["conversations"]
    text = tokenizer.apply_chat_template(convs, tokenize=False)
    return {"text": text}

dataset = dataset.map(format_chatml)

# 5. Entrenar
trainer = SFTTrainer(
    model=model,
    tokenizer=tokenizer,
    train_dataset=dataset,
    dataset_text_field="text",
    max_seq_length=2048,
    args=TrainingArguments(
        per_device_train_batch_size=2,
        gradient_accumulation_steps=4,
        warmup_steps=5,
        num_train_epochs=3,
        learning_rate=2e-4,
        fp16=True,
        logging_steps=10,
        output_dir="outputs",
```

```
        optim="adamw_8bit",
        seed=42,
    ),
)

trainer.train()

# 6. Guardar adapter
model.save_pretrained("my-finetuned-adapter")
tokenizer.save_pretrained("my-finetuned-adapter")

# 7. (Opcional) Exportar a GGUF para Ollama
model.save_pretrained_gguf("my-model-gguf", tokenizer, quantization_method="q4_k_m")
```

Con 200 ejemplos y Qwen 7B en QLoRA, esto tarda unos 15-20 minutos en una RTX 4090. Con el modelo 27B, 45-90 minutos.

Evaluar resultados

Entrenar sin evaluar es como cocinar sin probar. Necesitas un proceso sistematico para saber si tu fine-tune realmente mejora sobre el modelo base.

Holdout set obligatorio

Siempre reserva un 15-20% de tus datos para evaluacion. Nunca evalues con datos que el modelo ya vio durante entrenamiento. Divide antes de empezar:

```
from sklearn.model_selection import train_test_split

train_data, eval_data = train_test_split(
    full_dataset,
    test_size=0.15,
    random_state=42
)
# Guarda eval_data aparte, no lo toques hasta el final
```

Metricas automatizadas

- **Exact match:** Para tareas de clasificacion o extraccion estructurada. El output coincide exactamente con la referencia.
- **ROUGE/BLEU:** Para generacion de texto. Miden overlap con respuestas de referencia.
- **JSON validity:** Si esperas JSON, verifica que el 100% de outputs sean JSON valido.
- **Perplexity:** Metrica del propio modelo. Menor = mas confianza. Util para comparar checkpoints.

Evaluacion humana

Las metricas automatizadas no captan todo. Haz una evaluacion manual de al menos 50 ejemplos del holdout set. Para cada uno, evalua:

- Correctitud: la respuesta es factualmente correcta?
- Formato: sigue el formato esperado?
- Completitud: incluye toda la informacion necesaria?
- Regresiones: algo que el modelo base hacia bien y ahora hace mal?

Comparativa A/B

```
# Evaluar modelo base vs fine-tuned en los mismos inputs
results = []
for example in eval_data:
    base_output = generate(base_model, example["input"])
    ft_output = generate(finetuned_model, example["input"])
    results.append({
        "input": example["input"],
        "expected": example["expected"],
        "base": base_output,
        "finetuned": ft_output,
        "base_correct": evaluate(base_output, example["expected"]),
        "ft_correct": evaluate(ft_output, example["expected"]),
    })

# Resumen
base_acc = sum(r["base_correct"] for r in results) / len(results)
ft_acc = sum(r["ft_correct"] for r in results) / len(results)
print(f"Base: {base_acc:.1%} | Fine-tuned: {ft_acc:.1%} | Delta: {ft_acc-base_acc:+.1%}")
```

Si tu fine-tune no mejora al menos un 5-10% sobre el modelo base con un buen prompt, probablemente no merece la pena el esfuerzo de mantenimiento.

Costes y hardware

El fine-tuning tiene costes directos (hardware, electricidad) e indirectos (tiempo de preparacion de datos, iteracion, mantenimiento). Vamos a ser concretos.

Hardware propio

GPU	VRAM	PRECIO	MODELO MAX (QLORA)	TIEMPO 500 EJEMPLOS
RTX 4090	24GB	~1.800 EUR	27B (4-bit)	45-90 min
RTX 4080	16GB	~1.200 EUR	13B (4-bit)	30-60 min
A100 40GB	40GB	~8.000 EUR (usado)	70B (4-bit)	20-45 min
Hetzner GEX44	L40S 48GB	~170 EUR/mes	70B (4-bit)	20-45 min

Cloud on-demand

PROVEEDOR	GPU	COSTE/HORA	COSTE POR FINE-TUNE (500 EJ, 7B)
RunPod	A100 80GB	~1.90 EUR/h	~1 EUR
Lambda Labs	A100 80GB	~2.20 EUR/h	~1.10 EUR
Vast.ai	RTX 4090	~0.40 EUR/h	~0.50 EUR
Google Colab Pro	A100 40GB	~10 EUR/mes flat	Incluido

El fine-tuning en si es barato. Lo caro es el tiempo humano preparando datos y iterando. Un dataset de 500 ejemplos de calidad puede llevar 20-40 horas de trabajo humano. Ese es el coste real.

Cuando NO hacer fine-tuning

Hay situaciones donde fine-tuning es la respuesta equivocada, por mucho que parezca la solucion obvia:

- **Pocos datos:** Si tienes menos de 50 ejemplos de calidad, el modelo sobreajustara. Usa few-shot prompting.
- **Datos que cambian:** Si tu base de conocimiento se actualiza frecuentemente (precios, inventario, noticias), usa RAG. Un fine-tune queda obsoleto al dia siguiente.
- **Multiples tareas:** Si necesitas un modelo que haga 15 cosas diferentes, mejor usa un modelo general con prompts especializados. Fine-tuning tiende a especializar (y degradar en otras tareas).
- **Sin pipeline de evaluacion:** Si no puedes medir sistematicamente si el fine-tune mejora algo, no hagas fine-tuning. Volaras a ciegas.

- **El modelo base ya es bueno:** Si Claude o GPT-4 ya resuelve tu tarea con un prompt simple al 95% de accuracy, el fine-tune no va a ganar mucho. Invierte ese esfuerzo en otra cosa.
- **Produccion sin GPU:** Si tu entorno de produccion no puede servir el modelo fine-tuned (necesitas GPU para inferencia), el fine-tune es inutil. Evalua si puedes usar vLLM o si dependes de APIs.

> **Senal de alerta**
>
> Si tu motivacion para fine-tuning es "los modelos no entienden mi negocio", en el 90% de los casos la solucion real es un mejor prompt con contexto (RAG) y no un fine-tune. Prueba eso primero durante al menos 2 semanas antes de considerar fine-tuning.

Ejercicio practico

Ejercicio: Prepara un mini-dataset y ejecuta un fine-tune

Objetivo: Crear un dataset de 50 ejemplos para clasificacion de tickets de soporte y ejecutar un fine-tune con Unsloth.

Paso 1: Crear el dataset

Crea un archivo `dataset.json` con 50 conversaciones ChatML. Cada ticket de soporte debe clasificarse en una de estas categorias: billing, technical, account, feature_request.

```
# Genera 50 ejemplos variados. Aqui van 3 como referencia:
[
  {
    "conversations": [
      {"role": "system", "content": "Clasifica tickets de soporte en: billing, technical,
      {"role": "user", "content": "No puedo acceder a mi cuenta desde ayer, me dice passw
      {"role": "assistant", "content": "account"}
    ]
  },
  {
    "conversations": [
      {"role": "system", "content": "Clasifica tickets de soporte en: billing, technical,
      {"role": "user", "content": "Me han cobrado dos veces la suscripcion de este mes"},
      {"role": "assistant", "content": "billing"}
    ]
  },
  {
    "conversations": [
      {"role": "system", "content": "Clasifica tickets de soporte en: billing, technical,
      {"role": "user", "content": "La API devuelve 500 cuando envio un payload mayor a 1M
      {"role": "assistant", "content": "technical"}
    ]
  }
]
```

Paso 2: Validar

Usa el script de validacion de la seccion anterior. Verifica que todos los ejemplos tienen la estructura correcta.

Paso 3: Ejecutar fine-tune

Si tienes acceso a GPU (local o cloud), ejecuta el script de Unsloth con tu dataset. Si no, usa Google Colab con el notebook gratuito de Unsloth.

Paso 4: Evaluar

Reserva 10 de tus 50 ejemplos como holdout. Compara accuracy del modelo base (con prompt) vs fine-tuned.

Entregable: Un documento con: dataset (50 ejemplos), accuracy base vs fine-tuned, y una conclusion sobre si el fine-tune valio la pena para esta tarea.

Conclusiones clave

Key takeaways del M25

1. Fine-tuning es el ULTIMO recurso, no el primero. Siempre intenta prompting y RAG antes.
2. La calidad del dataset importa mas que la cantidad. 200 ejemplos perfectos superan a 10.000 mediocres.
3. ChatML es el formato estandar. System + User + Assistant, sin excepciones.
4. QLoRA con rank 16 es el sweet spot: cabe en una RTX 4090 y da 95%+ de la calidad de full fine-tuning.
5. Unsloth es la herramienta mas eficiente en 2026. 2-5x mas rapido que HuggingFace vanilla.
6. SIEMPRE evalua con un holdout set. Si no mejora un 5-10% sobre el base con prompt, no vale la pena.
7. El coste real no es la GPU (eso es barato). Es el tiempo humano preparando datos y manteniendo el pipeline.