Crear LLM

Hecho por: Roberto Monedero Alonso

Parte 1: Teoría Fundamental (Actualizada)

Antes de ejecutar código, entendamos qué estamos haciendo basándonos en tu documentación y el estado del arte actual.

1. ¿Qué es "Crear" un LLM localmente?

No estamos "creando" un cerebro desde cero (Pre-entrenamiento), lo cual requeriría miles de GPUs. Lo que haremos es **Fine-Tuning (Ajuste Fino)**.

- **Concepto:** Tomas un modelo "base" que ya sabe hablar (como Llama 3 o Qwen) y lo especializas en una tarea concreta.
- **Analogía:** Si el modelo base es un estudiante universitario graduado, el Fine-Tuning es enviarlo a hacer un máster en una especialidad (ej. Cocina o Ingeniería Espacial).

2. La Técnica: LoRA (Low-Rank Adaptation)

Entrenar todos los parámetros de un modelo es costoso. Usaremos **LoRA** (o QLoRA si cuantizamos).

- **Cómo funciona:** En lugar de modificar todo el "cerebro" del modelo, congelamos el modelo original y entrenamos solo unas pequeñas capas de "adaptadores" encima.
- **Ventaja:** Es mucho más rápido, consume menos memoria y el resultado pesa megabytes en lugar de gigabytes.

3. El Flujo de Datos (Pipeline)

El proceso sigue estos 4 pasos críticos:

1. **Dataset (Datos):** La calidad es reina ("Garbage in, garbage out"). Necesitamos pares de Instrucción -> Respuesta.
2. **Entrenamiento (Training):** El Mac leerá los datos y ajustará los adaptadores LoRA para minimizar el error en sus respuestas.
3. **Fusión (Fusing):** Unimos los adaptadores entrenados con el modelo base para tener un solo archivo funcional.
4. **Cuantización (GGUF):** Convertimos el modelo a formato GGUF (formato universal para CPUs y Mac) y reducimos su precisión (ej. a 4 bits) para que ocupe menos espacio sin perder mucha inteligencia.

Parte 2: Práctica (El Script Maestro)

He consolidado todos los pasos manuales de los PDFs en un único script de Python moderno. Este script detectará tu entorno, preparará los datos, entrenará el modelo usando la librería mlx, lo fusionará y lo convertirá a GGUF para LM Studio.

Requisitos Previos

Abre tu terminal en el Mac/Linux/Windows y asegúrate de tener lo siguiente:

1. **Python instalado.**
2. **Instala las librerías necesarias:**

python3 -m venv venv; source venv/bin/activate; pip install mlx mlx-lm huggingface_hub → Mac
pip install torch transformers peft trl bitsandbytes accelerate → Windows/Linux

(Nota 1: No necesitamos instalar llama.cpp vía pip, el script clonará y compilará la última versión automáticamente para asegurar compatibilidad con GGUF).

(Nota 2: Windows es más exigente con las versiones. Si el script te da error al instalar `bitsandbytes`, usa este comando específico antes de correr el script: `pip install https://github.com/jllllll/bitsandbytes-windows-webui/releases/download/wheels/bitsandbytes-0.41.1-py3-none-win_amd64.whl` (Esta es una versión precompilada para Windows muy popular).)

El Script: crear_llm_mac.py

nano crear_llm_mac.py

Copia este código, guárdalo como crear_llm_mac.py en una carpeta vacía y ejecútalo:

```
import os
import json
import subprocess
import sys
import platform
import shutil
from pathlib import Path
# Importamos esto para la auto-reparación del diccionario al final
from huggingface_hub import snapshot_download

# ==========================================
# ⚙️ CONFIGURACIÓN (Puedes editar esto)
# ==========================================

# Nombre de tu proyecto (así se llamarán los archivos finales)
NOMBRE_PROYECTO = "Mi-IA-Universal"

# Modelo base: Qwen 2.5 (3B) es excelente para Mac: rápido y muy inteligente.
MODELO_ID = "Qwen/Qwen2.5-3B-Instruct"

# System Prompt: La "personalidad" base de tu IA.
SYSTEM_PROMPT = "Eres un asistente inteligente y conciso."

# TUS DATOS: Aquí es donde la IA aprende.
```

```
# Para el ejemplo usamos pocos, pero para una IA real, pon aquí 50-100 ejemplos.
EJEMPLOS_ENTRENAMIENTO = [
    {"user": "¿Capital de España?", "assistant": "Madrid."},
    {"user": "Explica la gravedad.", "assistant": "Es la fuerza que atrae objetos con masa."},
    {"user": "Python: print hola", "assistant": "print('hola')"},
    {"user": "¿Quién eres?", "assistant": "Soy una IA personalizada entrenada en tu Mac."}
]

# Configuración técnica
TRAIN_EPOCHS = 100        # Iteraciones. 100 es rápido para probar. Pon 600+ para
aprender de verdad.
BATCH_SIZE = 1            # Dejar en 1 para no saturar la memoria del Mac.
TIPO_CUANTIZACION = "q4_k_m" # El formato de compresión equilibrado.

# Rutas del sistema (No tocar)
WORK_DIR = Path(os.getcwd())
LLAMA_DIR = WORK_DIR / "llama.cpp"

# ==========================================
# 🛠️ FUNCIONES DEL PROCESO
# ==========================================

def detectar_entorno():
    """Verifica que estás en un Mac."""
    print(f"\n🖥️  Detectando sistema...")
    if platform.system() != "Darwin":
        print("❌ Este script está diseñado específicamente para Mac (Apple Silicon).")
        sys.exit(1)
    print("✅ Sistema Mac detectado. Usando aceleración MLX (GPU/Neural Engine).")

def preparar_datos():
    """
    Convierte tus ejemplos de arriba en archivos .jsonl que MLX entiende.
    Genera train.jsonl (para estudiar) y valid.jsonl (para examinarse).
    """
    print(f"\n📂 [1/6] Preparando los datos...")
    os.makedirs(WORK_DIR / "data", exist_ok=True)

    datos_formateados = []
    for ej in EJEMPLOS_ENTRENAMIENTO:
        # Formato ChatML (System -> User -> Assistant)
        msgs = [
            {"role": "system", "content": SYSTEM_PROMPT},
            {"role": "user", "content": ej["user"]},
            {"role": "assistant", "content": ej["assistant"]}
        ]
        datos_formateados.append({"messages": msgs})
```

```
    # MLX es estricto: necesita validación. Duplicamos los datos para este ejemplo simple.
    for archivo in ["train.jsonl", "valid.jsonl"]:
        ruta = WORK_DIR / "data" / archivo
        with open(ruta, "w", encoding="utf-8") as f:
            for linea in datos_formateados:
                f.write(json.dumps(linea, ensure_ascii=False) + "\n")

    print(f"✅ Datos listos en la carpeta ./data")

def preparar_herramientas():
    """
    Descarga y compila llama.cpp, la herramienta necesaria para convertir el modelo.
    Usa 'cmake' porque el sistema antiguo 'make' da problemas en versiones nuevas.
    """
    print(f"\n🛠️  [2/6] Preparando herramientas (llama.cpp)...")

    # 1. Clonar si no existe
    if not LLAMA_DIR.exists():
        print("   ⬇️  Clonando repositorio...")
        subprocess.run(["git", "clone", "https://github.com/ggerganov/llama.cpp.git"],
check=True)

    # 2. Instalar dependencias de Python
    print("   📦 Instalando librerías auxiliares...")
    try:
        subprocess.run([sys.executable, "-m", "pip", "install", "-r", "requirements.txt"],
cwd=LLAMA_DIR, check=True)
    except:
        # Si falla requirements.txt (por versiones estrictas), instalamos lo básico manual
        print("   ⚠️  Instalando dependencias manualmente (modo compatible)...")
        subprocess.run([sys.executable, "-m", "pip", "install", "numpy", "sentencepiece", "gguf",
"protobuf"], check=True)

    # 3. Compilar con CMake (El paso crítico que fallaba antes)
    binario_quantize = LLAMA_DIR / "build" / "bin" / "llama-quantize"

    if not binario_quantize.exists():
        print("   🔨 Compilando binarios con CMake (esto puede tardar unos minutos)...")
        try:
            # Crear carpeta de construcción
            (LLAMA_DIR / "build").mkdir(exist_ok=True)
            # Configurar y Construir
            subprocess.run(["cmake", "-B", "build"], cwd=LLAMA_DIR, check=True)
            subprocess.run(["cmake", "--build", "build", "--config", "Release", "-j"],
cwd=LLAMA_DIR, check=True)
        except Exception as e:
            print(f"❌ Error compilando: {e}")
            print("💡 Pista: Asegúrate de tener cmake instalado ('brew install cmake')")
```

```
            sys.exit(1)

    return LLAMA_DIR

def entrenar_y_fusionar():
    """
    Usa la librería MLX de Apple para entrenar los adaptadores LoRA y luego fusionarlos.
    """
    print(f"\n🧠 [3/6] Entrenando modelo (Fine-Tuning)...")
    adapter_path = WORK_DIR / "adapters"

    # Comando de entrenamiento
    # Usamos 'python -m mlx_lm.lora'
    cmd_train = [
        sys.executable, "-m", "mlx_lm.lora",
        "--model", MODELO_ID,
        "--train",
        "--data", str(WORK_DIR / "data"),
        "--iters", str(TRAIN_EPOCHS),
        "--batch-size", str(BATCH_SIZE),
        "--adapter-path", str(adapter_path)
    ]

    # Si ya existen adaptadores, no re-entrenamos para ahorrar tiempo (bórralos si quieres
empezar de cero)
    if not adapter_path.exists():
        subprocess.run(cmd_train, check=True)
    else:
        print("   ⏩ Adaptadores encontrados. Saltando entrenamiento.")

    print(f"\n🔗 [4/6] Fusionando cerebro nuevo con el base...")
    model_fused = WORK_DIR / f"{NOMBRE_PROYECTO}-Fused"

    cmd_fuse = [
        sys.executable, "-m", "mlx_lm.fuse",
        "--model", MODELO_ID,
        "--adapter-path", str(adapter_path),
        "--save-path", str(model_fused)
    ]
    subprocess.run(cmd_fuse, check=True)

    return model_fused

def reparar_diccionario(ruta_modelo_fusionado):
    """
    EL TRASPLANTE: Descarga los archivos de configuración originales y sanos
    para sobrescribir los que MLX genera a veces corruptos.
    Esto evita el error 'list object has no attribute keys'.
```

```
    """
    print(f"\n🚑 [5/6] Auto-reparación del Tokenizer...")
    print("   Descargando configuración original sana desde HuggingFace...")

    try:
        snapshot_download(
            repo_id=MODELO_ID,
            allow_patterns=["tokenizer*", "vocab*", "merges*", "special_tokens*"],
            local_dir=ruta_modelo_fusionado,
            local_dir_use_symlinks=False
        )
        print("✅ Reparación completada. El modelo está listo para convertirse.")
    except Exception as e:
        print(f"⚠️ Error en la reparación: {e}")

def convertir_y_comprimir(ruta_modelo_fusionado, llama_dir):
    """
    Convierte a GGUF y comprime el archivo final.
    """
    print(f"\n📦 [6/6] Exportando a GGUF final...")

    script_convert = llama_dir / "convert_hf_to_gguf.py"
    archivo_f16 = WORK_DIR / f"{NOMBRE_PROYECTO}.gguf"
    archivo_final = WORK_DIR / f"{NOMBRE_PROYECTO}-{TIPO_CUANTIZACION}.gguf"

    # Paso A: Conversión a GGUF crudo (F16)
    print("   A) Convirtiendo estructura (F16)...")
    subprocess.run([sys.executable, str(script_convert), str(ruta_modelo_fusionado),
"--outfile", str(archivo_f16), "--outtype", "f16"], check=True)

    # Paso B: Cuantización (Compresión) usando el binario compilado
    print(f"   B) Comprimiendo a {TIPO_CUANTIZACION}...")
    binario_quantize = llama_dir / "build" / "bin" / "llama-quantize"

    subprocess.run([str(binario_quantize), str(archivo_f16), str(archivo_final),
TIPO_CUANTIZACION], check=True)

    # Limpieza: Borrar el archivo intermedio pesado
    if archivo_f16.exists():
        os.remove(archivo_f16)

    return archivo_final

# ==========================================
# 🏁 PUNTO DE ENTRADA
# ==========================================
if __name__ == "__main__":
    try:
```

```
    detectar_entorno()
    preparar_datos()
    llama_path = preparar_herramientas()

    # Entrenamos y obtenemos la ruta de la carpeta fusionada
    ruta_fused = entrenar_y_fusionar()

    # Aplicamos el parche de reparación ANTES de convertir
    reparar_diccionario(ruta_fused)

    # Convertimos
    ruta_final = convertir_y_comprimir(ruta_fused, llama_path)

    print("\n" + "="*60)
    print(f"🎉 ¡ÉXITO TOTAL! Tu IA está lista.")
    print(f"📄 Archivo: {ruta_final}")
    print("👉 Arrástralo a LM Studio y pruébalo.")
    print("="*60)

except Exception as e:
    print(f"\n❌ Ocurrió un error: {e}")
    print("Revisa los mensajes anteriores para ver qué falló.")
```

- Si estaba en Windows:

El script descarga un ZIP oficial de `llama.cpp`, extrae el archivo `llama-quantize.exe` y lo colócalo en la carpeta.

Ahora toca ejecutar el archivo: python3 crear_llm_mac.py

Parte 3: Explicación Detallada de los Comandos y Opciones

Aquí explico qué hace cada sección del script para que entiendas la "magia" detrás, alineado con tu petición.

1. Configuración y Dataset

- MODELO_BASE = "mlx-community/Llama-3.2-3B...": He elegido la versión **Llama 3.2** de 3 billones de parámetros.
  - *Por qué:* Es un modelo muy moderno (2024), pequeño (ideal para empezar), y la versión mlx-community ya viene preparada para Apple Silicon.
- crear_dataset(): Genera el archivo train.jsonl.
  - *Formato:* Usa el estándar de mensajes (role: system/user/assistant) que es el formato moderno de chat. Esto es crucial para que el modelo entienda que es una conversación.

2. Entrenamiento (mlx_lm.lora)

El script ejecuta este comando internamente. Desglosemos las opciones importantes:

- --model: El modelo base que descargará de Hugging Face.
- --train: Activa el modo entrenamiento.
- --iters 100: **Iteraciones**. Es el número de veces que el modelo verá ejemplos.
    - *Nota:* Para pruebas rápidas, 100 está bien. Para un modelo "inteligente" real, sube esto a 600 o 1000 en el script.
- --batch-size 1: Cuántos ejemplos procesa a la vez. En Mac, mantenerlo en 1 o 2 ahorra memoria RAM unificada.
- --lora-layers 16: Cuántas capas del cerebro estamos adaptando. 16 es un estándar equilibrado entre calidad y velocidad.
- --adapter-path: Dónde se guardan los archivos temporales (.safetensors) con el nuevo aprendizaje.

3. Fusión (mlx_lm.fuse)

- Este paso toma el modelo original (que pesa varios GBs) y le "pega" matemáticamente los cambios que aprendió (los adaptadores). El resultado es una carpeta con un modelo completo de PyTorch/Safetensors.

4. Cuantización y GGUF (llama.cpp)

Aquí es donde hacemos la magia para LM Studio.

- **Clonación y Make:** El script descarga llama.cpp (la herramienta estándar para correr LLMs en local) y la compila usando make. Al estar en Mac, detectará automáticamente Metal (la tecnología gráfica de Apple) para acelerar el proceso.
- convert_hf_to_gguf.py: Es el script de Python dentro de llama.cpp que transforma el modelo.
- --outtype q4_k_m: **Esto es clave.**
    - Significa "Quantization 4-bit K-Medium".
    - Reduce el peso del modelo drásticamente (de 16 bits a 4 bits por "neurona") manteniendo casi toda la inteligencia. Es el formato recomendado por LM Studio.

Parte 4: Integración en LM Studio

Una vez que el script termine y veas el mensaje de "¡ÉXITO!", sigue estos pasos finales mencionados en la documentación :

1. Abre **LM Studio**.
2. Ve a la pestaña de búsqueda (lupa) o a "My Models".
3. Arrastra el archivo generado (Chef-Bot-Pro-Q4_K_M.gguf) directamente a la interfaz de LM Studio, o colócalo en la carpeta de modelos del programa.
4. Selecciona el modelo en la barra superior.
5. **System Prompt:** Configura el prompt del sistema igual que en el entrenamiento para activar su personalidad:
   "Eres un chef experto. Tu objetivo es dar recetas precisas basadas en ingredientes."
6. ¡Pregúntale por una receta con ingredientes que no estaban en el dataset para ver cómo generaliza!