Facsímil 03

Arquitecturas y modelos

Capítulo 06

Transfer learning, destilación y modelos abiertos

686f6c61 · 2026

Capítulo 06: Transfer learning, destilación y modelos abiertos

Casi nadie empieza desde cero

Entrenar un modelo grande desde cero es caro, lento y exige datos, infraestructura, evaluación y experiencia. La mayoría de equipos no empieza ahí. Empieza con un modelo que ya sabe mucho y lo adapta a una tarea concreta.

Ese cambio de mentalidad es enorme. En vez de preguntar «¿cómo entreno mi propio LLM desde cero?», preguntamos:

PREGUNTA PRÁCTICA	TÉCNICA QUE SUELE APARECER
¿Puedo reutilizar un modelo ya entrenado?	Transfer learning.
¿Puedo ajustarlo a mi dominio?	Fine-tuning o instruction tuning.
¿Puedo entrenar pocos parámetros?	LoRA, adapters, QLoRA.
¿Puedo hacerlo más pequeño?	Destilación y cuantización.
¿Puedo ejecutarlo y modificarlo legalmente?	Modelos abiertos, licencias y documentación.

Este capítulo une esas piezas. Es menos espectacular que hablar de billones de parámetros, pero mucho más cercano a la realidad profesional: elegir una base, adaptarla, medirla, comprimirla si hace falta y entender qué te permite su licencia.

Cuando aparezcan herramientas, catálogos, modelos abiertos o librerías de evaluación, toma la sección como estado observado a **10 de junio de 2026**. Las ideas estables son el mecanismo y el criterio de decisión; los nombres, versiones, licencias y capacidades concretas deben revisarse antes de usarlos en un proyecto real.

Qué no es adaptar un modelo

Adaptar un modelo no es «meterle documentos» como quien mete libros en una estantería. Si haces fine-tuning, cambias pesos. Si haces RAG, conectas recuperación externa. Si haces prompt engineering, cambias el contexto. Son mecanismos distintos.

Tampoco conviene pensar que fine-tuning arregla cualquier problema. Si el modelo falla porque no tiene el dato actualizado, puede que necesites recuperación. Si falla porque no sigue un formato, quizá necesitas mejores ejemplos, validación o una interfaz más estricta. Si falla porque la tarea requiere cálculo exacto, puede que necesites una herramienta.

Y un modelo abierto no siempre significa lo mismo. A veces puedes descargar pesos, pero no tienes datos de entrenamiento. A veces la licencia permite investigación pero limita uso comercial. A veces hay código, tokenizer y evaluación; otras veces solo un checkpoint. La palabra «abierto» exige mirar detalles.

Transfer learning: reutilizar una base

Transfer learning significa usar conocimiento aprendido en un contexto para mejorar otro. En NLP moderno, el patrón se volvió muy claro con modelos preentrenados: primero se entrena una base con grandes cantidades de texto; después se adapta a tareas concretas.[1] T5 lo llevó a una formulación texto-a-texto muy amplia.[2]

Una forma simple de verlo:

```
modelo base -> adaptación -> modelo útil para una tarea concreta
```

FASE	QUÉ APRENDE	EJEMPLO
Preentrenamiento	Patrones generales del lenguaje y del mundo textual.	Continuar texto, relacionar conceptos, estilo, idiomas.
Instruction tuning	Seguir instrucciones humanas en formato pregunta/respuesta.	«Resume esto», «extrae estos campos», «clasifica».
Fine-tuning de dominio	Patrones de un sector o tarea concreta.	Soporte técnico, documentos jurídicos, código interno.
Evaluación	Si la adaptación ayuda o daña.	Tests propios, casos reales, comparación contra base.

Instruction tuning se volvió central para convertir modelos base en asistentes útiles.[3] Pero no hay que confundirlo con una capacidad espontánea: es entrenamiento sobre ejemplos de comportamiento deseado.

Datasets: qué se entrena realmente

Cuando hablamos de entrenamiento, la palabra “datos” se queda corta. Un LLM no se entrena con “internet” como una masa uniforme. Se entrena con mezclas de documentos, código, conversaciones, instrucciones, comparaciones, problemas, imágenes, etiquetas, filtros y evaluaciones. Cada familia de datos empuja una capacidad distinta.

Un dataset no es solo una carpeta con archivos. Es una decisión de producto y de ingeniería:

PREGUNTA	POR QUÉ IMPORTA
¿De dónde sale?	Derechos, licencia, idioma, sesgo de dominio y calidad.
¿Qué formato tiene?	Texto libre, pares pregunta-respuesta, preferencias, imagen-texto, código, tablas.
¿Qué enseña?	Continuar texto, seguir instrucciones, razonar, preferir una respuesta, ver imágenes.
¿Qué no enseña?	Ningún dataset cubre todo; cada mezcla tiene huecos.
¿Cómo se filtra?	Duplicados, baja calidad, datos sensibles, idioma, toxicidad, contenido roto.
¿Cómo se mezcla?	La proporción entre web, código, matemáticas o diálogo cambia el modelo.
¿Cómo se evalúa?	Sin conjunto de evaluación separado, puedes entrenar para memorizar tus pruebas.

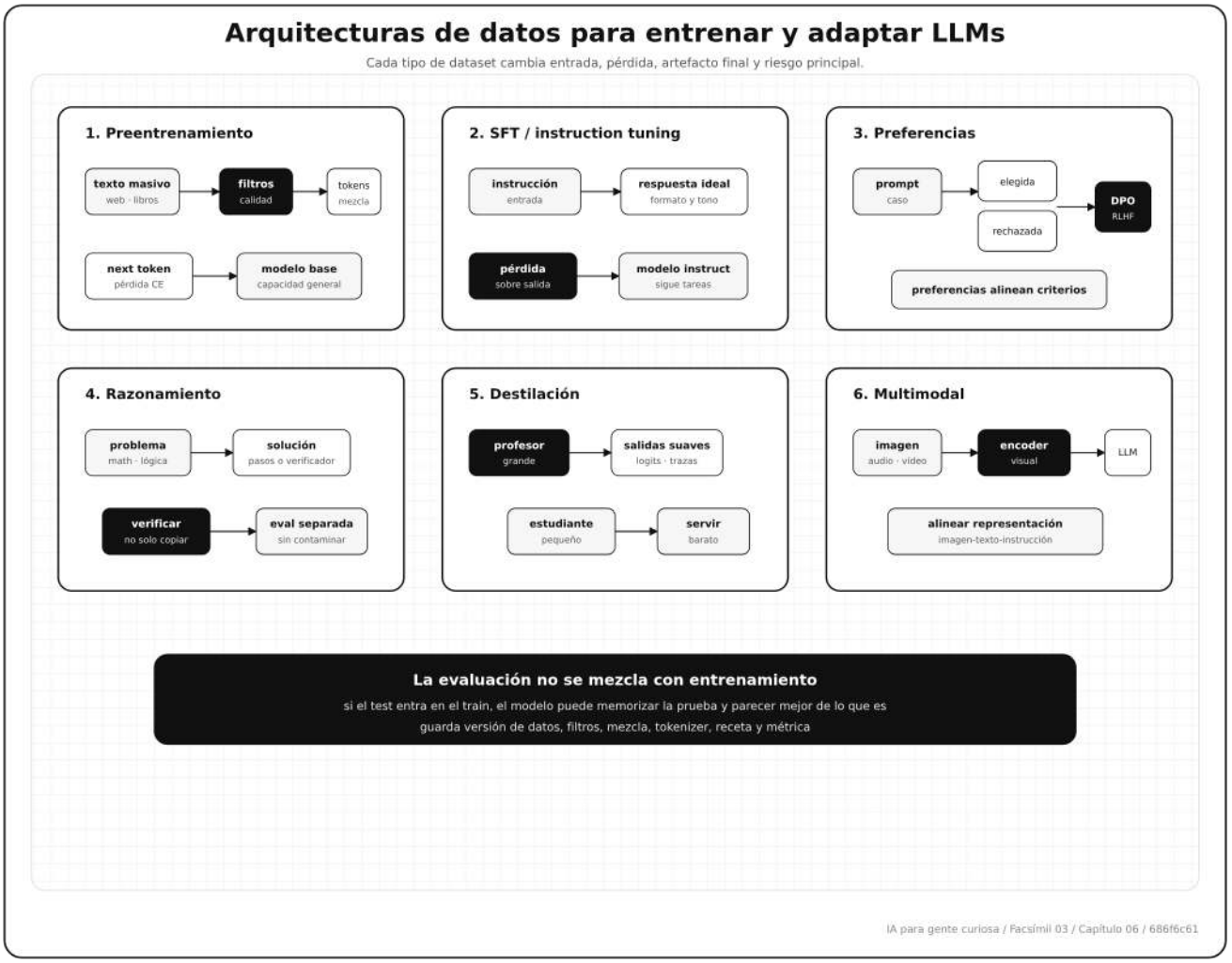
La arquitectura de entrenamiento cambia según el tipo de dataset.

TIPO DE DATASET	FORMA TÍPICA	QUÉ ENTRENA	EJEMPLOS CONOCIDOS
Preentrenamiento textual	Texto largo no etiquetado.	Predicción del siguiente token o reconstrucción.	C4/T5, The Pile.[4]
Código	Repositorios, archivos, funciones, comentarios.	Sintaxis, APIs, patrones de programación.	The Stack.[5]
SFT / instrucciones	`instrucción -> respuesta esperada`.	Seguir tareas, formato, tono, extracción, resumen.	FLAN, Dolly-15K.[6]
Preferencias	`prompt, respuesta A, respuesta B, preferida`.	Elegir entre salidas plausibles.	HH-RLHF, UltraFeedback.[7]
Razonamiento	Problema, solución, pasos o verificador.	Matemáticas, cadenas de inferencia, comprobación.	GSM8K.[8]
Multimodal	Imagen-texto, imagen-instrucción-respuesta, vídeo-texto.	Alinear señales visuales y lenguaje.	LAION-5B, LLaVA.[9]
Evaluación	Preguntas con respuesta esperada o rúbrica.	No entrena; mide.	MMLU, BEIR, MTEB.[10]

No confundas dataset de entrenamiento con dataset de evaluación. El primero mueve el modelo. El segundo mide si el modelo se comporta mejor. Si una pregunta de evaluación se cuela en entrenamiento, la nota puede subir sin que el modelo haya aprendido a generalizar. A eso se le suele llamar contaminación de datos.

Arquitecturas de datos para entrenar y adaptar

La palabra arquitectura no solo habla de capas del Transformer. También habla de cómo conectas datos, filtros, pérdidas, modelo base, evaluadores y artefactos finales. Aquí tienes el mapa de las arquitecturas de entrenamiento más comunes.



El SVG muestra seis arquitecturas porque no todas optimizan lo mismo. Preentrenar busca capacidad general. SFT enseña formato y seguimiento de instrucciones. Preferencias enseñan criterios comparativos. Razonamiento necesita problemas verificables. Destilación pasa señales de un profesor a un estudiante. Multimodalidad alinea señales no textuales con lenguaje.

Qué optimiza cada arquitectura

Para leer bien un dataset hay que mirar tres cosas a la vez: entrada, pérdida y artefacto final. Dos datasets pueden tener el mismo tamaño y servir para cosas muy distintas.

ARQUITECTURA	ENTRADA REAL	OBJETIVO MATEMÁTICO	ARTEFACTO QUE PRODUCE	QUÉ DEBES COMPROBAR
Preentrenamiento causal	Texto tokenizado $x_1, x_2, \ldots, x_T$.	Predecir el siguiente token: $\mathcal{L} = -\sum_t \log p(x_t \mid x_{<t})$.	Modelo base.	Mezcla, deduplicación, idiomas, licencias y benchmarks filtrados.
Preentrenamiento texto-a-texto / denoising	Texto con huecos o transformación entrada-salida.	Reconstruir partes ocultas o salida esperada.	Modelo base transferible.	Cómo se generan corrupciones, tareas y plantillas.
SFT / instruction tuning	Instrucción, contexto opcional y respuesta ideal.	Entropía cruzada sobre los tokens de respuesta.	Modelo que sigue formato y tarea.	Calidad de respuestas, diversidad de instrucciones y ejemplos contradictorios.
Preferencias	Prompt, respuesta elegida y respuesta descartada.	Aumentar la probabilidad relativa de la elegida; en DPO aparece una pérdida logística sobre diferencias de log-probabilidad.	Modelo o adaptador alineado con una rúbrica.	Quién decide la preferencia, con qué criterio y si hay sesgo de longitud o estilo.
Razonamiento verificable	Problema, solución, pasos y a veces verificador.	Premiar soluciones que llegan a una respuesta comprobable.	Modelo mejor en problemas estructurados.	Separar entrenamiento y test; comprobar que no aprende plantillas memorizadas.
Destilación	Entradas y salidas del profesor: logits, respuestas o trazas.	Acercar distribución del estudiante a la del profesor, por ejemplo con KL.	Modelo estudiante más barato.	Qué capacidades pierde el estudiante fuera de la tarea.
Multimodal	Pares imagen-texto, imagen-instrucción-respuesta, audio-texto o vídeo-texto.	Alinear representaciones y generar texto condicionado por otra señal.	Modelo capaz de razonar sobre varias modalidades.	Calidad de pares, resolución, licencias y si la descripción textual corresponde a la señal.

ARQUITECTURA	ENTRADA REAL	OBJETIVO MATEMÁTICO	ARTEFACTO QUE PRODUCE	QUÉ DEBES COMPROBAR
Evaluación	Preguntas, respuestas esperadas, qrels, rúbricas o casos no respondibles.	No debería optimizar el modelo; mide una versión.	Métrica, traza y decisión de publicación.	Que no se mezcle con entrenamiento y que represente uso real.

La mezcla de datos también es una arquitectura. Si tienes familias $D_1, D_2, \ldots, D_n$, no basta con juntarlas.

Ejemplo de fórmula. Para razonar de forma pedagógica sobre una mezcla, podemos escribir un conjunto de entrenamiento como suma ponderada de familias:

$$D_{\text{train}} = \lambda_1 D_1 + \lambda_2 D_2 + \cdots + \lambda_n D_n$$

Aquí D_i representa una familia de datos (web filtrada, código, instrucciones, preferencias, razonamiento, imagen-texto) y λ_i representa su peso relativo en la mezcla. Esta fórmula no describe por sí sola cómo se implementa un *data loader* de entrenamiento real: sirve para recordar que cambiar proporciones cambia capacidades, sesgos, coste y contaminación posible. En un sistema real habría que documentar muestreo, filtros, deduplicación, licencias, idioma, fechas, splits y evaluación.

SÍMBOLO	SIGNIFICADO
D_i	Familia de datos: web, código, matemáticas, conversación, imagen-texto...
λ_i	Peso o proporción de esa familia en la mezcla.
D_{train}	Conjunto efectivo que verá el entrenamiento.

Subir $\lambda_{\text{código}}$ puede mejorar programación y cambiar estilo de respuesta. Subir λ_{math} puede ayudar en problemas formales y no mover demasiado tareas conversacionales. Por eso los informes técnicos serios hablan de mezcla, filtros y etapas; no solo de “tamaño del dataset”.

Tipos de dataset con ejemplos

1. Preentrenamiento textual. Aquí el modelo aprende a predecir texto. No recibe una tarea humana explícita; recibe secuencias y aprende regularidades. C4 y The Pile son ejemplos conocidos de corpora grandes. Lo importante no es solo el tamaño: importan filtros, deduplicación, idiomas, licencias, dominios y contaminación de evaluación.

2. Código. Entrenar con código no es igual que entrenar con prosa. Un dataset como The Stack contiene repositorios y lenguajes de programación. Eso ayuda a aprender sintaxis, patrones de APIs, comentarios, tests y estructura de proyectos. También obliga a mirar licencias y duplicados con mucho cuidado.

3. SFT o instruction tuning. Aquí cada ejemplo suele tener una instrucción y una respuesta deseada. FLAN mezcla tareas y plantillas para enseñar a seguir instrucciones. Dolly-15K ilustra un dataset más pequeño y humano de instrucciones/respuestas. Esta familia enseña comportamiento, no conocimiento vivo.

4. Preferencias. En DPO/RLHF no basta con una respuesta correcta. El dataset dice cuál de dos respuestas se prefiere. HH-RLHF y UltraFeedback ilustran esta familia. La arquitectura cambia: el modelo aprende de comparaciones, no solo de una salida ideal.

5. Razonamiento y verificación. GSM8K contiene problemas matemáticos con soluciones. Sirve para entrenar o evaluar razonamiento paso a paso, pero hay que evitar que los ejemplos de evaluación entren en entrenamiento. En razonamiento, la solución no debe ser una decoración: debe poder comprobarse.

6. Multimodalidad. LAION-5B ilustra pares imagen-texto a gran escala; LLaVA usa instrucción visual para conectar imagen y diálogo. Aquí el dataset no solo enseña palabras. Enseña alineación entre una representación visual y lenguaje.

7. Evaluación. MMLU, BEIR o MTEB no deberían tratarse como entrenamiento de conveniencia. Son instrumentos de medida. Puedes inspirarte en ellos, pero tu producto necesita evaluación propia, como vimos en el capítulo 10 del facsímil 4.

Qué debe mirar un ingeniero en un dataset

La ficha de un dataset debería leerse como una ficha de arquitectura:

CAMPO	QUÉ PREGUNTAS HACER
Procedencia	¿Quién creó los datos? ¿Con qué permisos? ¿De qué dominios vienen?
Formato	¿Texto libre, instrucciones, preferencias, código, imagen-texto, tablas?
Tamaño	¿Número de ejemplos, tokens, documentos o pares?
Calidad	¿Hay filtros, deduplicación, revisión humana o heurísticas claras?
Licencia	¿Permite entrenamiento, redistribución, uso comercial o derivados?
Cobertura	¿Qué idiomas, dominios, estilos y tareas incluye?
Huecos	¿Qué casos no cubre o cubre mal?
Riesgos	¿Datos personales, duplicados, baja calidad, sesgos de fuente?
Contaminación	¿Se mezcló con benchmarks o tests que luego se reportan?
Trazabilidad	¿Puedo reconstruir versión, filtros y mezcla usada?

Una regla simple: un dataset pequeño y muy bueno puede ganar a un dataset enorme y sucio en una adaptación concreta. Para preentrenamiento, el volumen importa mucho; para SFT, preferencias o destilación, la calidad y la cobertura del caso real pesan muchísimo.

Software que ayuda a auditar datasets y evaluar modelos

Sí, existe software para ayudar, pero conviene no meterlo todo en el mismo saco. Una herramienta puede limpiar datos, otra puede detectar ejemplos sospechosos, otra puede ejecutar benchmarks y otra puede evaluar un RAG. “Evaluar el dataset” no es una única operación.

HERRAMIENTA	QUÉ EVALÚA O CURA	CUÁNDO USARLA	QUÉ NO DEBES PEDIRLE
DataTrove	Procesa, filtra, deduplica y calcula estadísticas sobre texto a gran escala.[11]	Antes de preentrenar o construir un corpus grande.	No decide por sí sola si una respuesta SFT es pedagógicamente buena.
NVIDIA NeMo Curator	Curation multimodal: texto, imagen, vídeo y audio; filtrado, deduplicación y calidad a escala.[12]	Cuando el volumen exige pipelines distribuidos o multimodales.	No sustituye tu criterio de dominio ni una licencia clara.
Cleanlab TLM / Studio	Detecta ejemplos problemáticos en pares instrucción-respuesta: baja calidad, prompts vagos, PII, lenguaje tóxico, gramática, etc.[13]	Para revisar SFT, datasets de preferencias o conjuntos de evaluación con respuestas.	No convierte automáticamente una mala rúbrica en una buena.
Deepchecks	Integridad de datos, distribuciones, splits, comparación de modelos y validación clásica de ML.[14]	En datasets tabulares, visión o ML tradicional; también para detectar problemas de splits.	No es el evaluador principal de un LLM conversacional.
Hugging Face Evaluate	Métricas, mediciones y comparaciones reproducibles para modelos y datasets.[15]	Cuando necesitas métricas conocidas y resultados reproducibles.	No entiende tu negocio si no defines bien la tarea.
LightEval	Evaluación de LLMs en múltiples backends, tareas existentes, tareas propias y resultados muestra a muestra.[16]	Después de adaptar un modelo o comparar variantes.	No audita por sí solo el dataset de entrenamiento.
lm-evaluation-harness	Benchmarks académicos y tareas reproducibles para modelos de lenguaje.[17]	Para comparar contra MMLU, GSM8K, BBH y otros benchmarks públicos.	No sustituye tu evaluación privada de producto.
HELM	Evaluación holística: escenarios, métricas, robustez, transparencia y comparación amplia.[18]	Como marco mental o benchmark amplio.	No te dice si tu asistente responde bien a tus usuarios concretos.

Un flujo profesional mínimo sería:

1. **Antes de entrenar:** perfilar, filtrar, deduplicar, detectar idioma, longitud, PII, duplicados y baja calidad con DataTrove, NeMo Curator, Cleanlab o checks propios.
2. **Antes de tocar pesos:** congelar splits `train`, `validation` y `test`, con hashes y versión de dataset.

3. **Después de adaptar:** evaluar con LightEval, lm-evaluation-harness, Hugging Face Evaluate o un harness propio.
4. **Antes de publicar:** pasar tu dataset privado de evaluación, no solo benchmarks públicos.
5. **Si es RAG:** usar evaluación por capas como la del facsímil 4, capítulo 10: retrieval, contexto, groundedness, citas y abstención.

La idea importante es esta: el software ayuda a encontrar problemas, pero no define por ti qué significa “bueno”. Esa definición sigue siendo del equipo: tarea, rúbrica, usuario, coste, licencia y riesgo de error.

Fine-tuning completo: mover todos los pesos

En fine-tuning completo, partimos de unos pesos θ y los actualizamos con nuevos datos:

$$\theta' = \theta - \eta \nabla_\theta \mathcal{L}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
θ	Pesos iniciales del modelo base.	Pesos de un LLM preentrenado.
θ'	Pesos después del ajuste.	Modelo especializado.
η	Tasa de aprendizaje.	2×10^{-5}.
$\nabla_\theta \mathcal{L}$	Gradiente de la pérdida respecto a los pesos.	Dirección para reducir error.
$\mathcal{L}$	Función de pérdida.	Error en respuestas esperadas.

Esto puede funcionar muy bien, pero tiene costes:

VENTAJA	COSTE
Gran capacidad de adaptación.	Mucha memoria y cómputo.
Puede modificar profundamente el comportamiento.	Riesgo de olvidar capacidades útiles si los datos son estrechos.
Produce un único modelo ajustado.	Guardar y servir muchas variantes puede ser caro.

En equipos pequeños, fine-tuning completo no suele ser la primera opción. Antes se prueba prompt, RAG, evaluación, datos mejores y métodos eficientes como LoRA.

El olvido catastrófico: adaptar puede borrar lo aprendido

Hay un riesgo concreto detrás de «el modelo puede olvidar capacidades útiles». Tiene nombre propio: olvido catastrófico. Cuando mueves muchos pesos con un conjunto de datos estrecho, el modelo mejora en esa tarea y a la vez degrada habilidades que ya tenía, porque reescribe parámetros que servían para otras cosas. Es un fenómeno conocido desde las primeras redes conexionistas.[19]

Imagina un asistente que respondía bien sobre muchos temas. Lo afinas con 2 000 ejemplos de un único formato jurídico. Después escribe contratos impecables, pero contesta peor preguntas generales que antes acertaba. No es magia ni mala suerte: los gradientes empujaron sus pesos hacia un rincón del espacio.

DEFENSA	IDEA	COSTE
Congelar la base	PEFT (LoRA, adapters, prompt tuning) deja W intacto y aprende una pieza aparte.	Menos capacidad de cambio profundo.
Mezclar datos	Incluir ejemplos generales junto a los del dominio (*data replay*).	Necesitas mantener un conjunto general.
Regularizar hacia el origen	Penalizar alejarse de los pesos base, como en *Elastic Weight Consolidation*.	Más configuración y ajuste.
Evaluar dentro y fuera	Medir la tarea nueva y también las capacidades viejas.	Exige un conjunto de regresión.

La regularización elástica es una de las defensas clásicas: protege los parámetros importantes para tareas previas mientras aprende la nueva.[20] Por eso PEFT no solo ahorra memoria: al congelar la base, reduce de raíz el riesgo de olvidar. Es una de las razones por las que LoRA se volvió la opción por defecto en muchos equipos.

La pérdida y los gradientes: qué cambia de verdad

Cuando decimos «ajustar pesos», estamos diciendo algo muy concreto: calcular una pérdida, derivarla y mover parámetros para reducirla.

Supongamos que queremos enseñar al modelo a continuar esta frase:

> El cliente quiere cambiar unos auriculares porque llegaron...

Y el token correcto en nuestro ejemplo es:

```
rotos
```

El modelo produce logits para tres candidatos:

TOKEN CANDIDATO	LOGIT
factura	2,0
rotos	1,0
saludo	0,0

Aplicamos softmax:

$$p_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

TOKEN	PROBABILIDAD P_I	ETIQUETA Y_I	GRADIENTE $P_I - Y_I$
factura	0,665	0	0,665
rotos	0,245	1	-0,755
saludo	0,090	0	0,090

La pérdida de entropía cruzada para el token correcto es:

$$\mathcal{L} = -\log(p_{\text{rotos}})$$

$$\mathcal{L} \approx -\log(0{,}245) \approx 1{,}408$$

La derivada respecto a cada logit, cuando usamos softmax y entropía cruzada, queda así:

$$\frac{\partial \mathcal{L}}{\partial z_i} = p_i - y_i$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
z_i	Logit del token candidato i.	$z_{\text{factura}} = 2{,}0$.
p_i	Probabilidad tras softmax.	$p_{\text{rotos}} = 0{,}245$.
y_i	Etiqueta real en formato one-hot.	Para «rotos», $y = 1$.
$p_i - y_i$	Gradiente respecto al logit.	Para «rotos», $-0{,}755$.

Lectura práctica:

TOKEN	QUÉ LE DICE EL GRADIENTE AL ENTRENAMIENTO
factura	«Te estás pasando con este token; baja su puntuación».
rotos	«Este era el bueno; sube su puntuación».
saludo	«También está algo alto; bájalo un poco».

Si la última capa calcula:

$$z = hW$$

entonces el gradiente llega a W así:

$$\frac{\partial \mathcal{L}}{\partial W} = h^T (p - y)$$

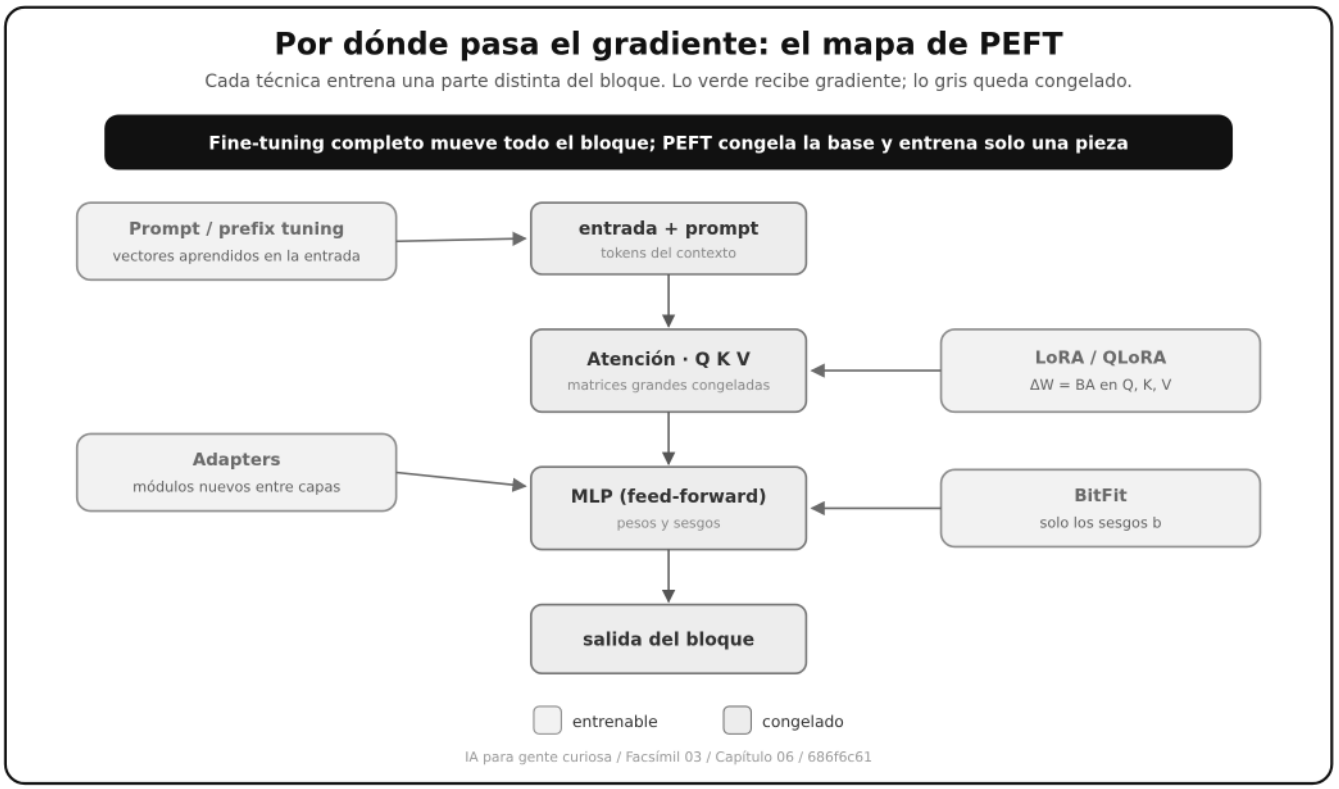
En la práctica no entrenamos con una frase, sino con lotes de ejemplos. Si tenemos B ejemplos, una pérdida media sería:

$$\mathcal{L}_{\text{lote}} = -\frac{1}{B} \sum_{b=1}^{B} \log p(y^{(b)} \mid x^{(b)})$$

El optimizador no ve «atención al cliente», «factura» o «tono educado». Ve números. Si el lote contiene muchos ejemplos donde una devolución debe terminar en un formato concreto, los gradientes empujan al modelo hacia ese patrón. Si los ejemplos son pobres, repetidos o demasiado estrechos, el modelo puede mejorar esa plantilla y empeorar fuera de ella.

TÉCNICA	POR DÓNDE DEJAMOS PASAR EL GRADIENTE	QUÉ QUEDA CONGELADO	LECTURA PRÁCTICA
Fine-tuning completo	Casi todo el modelo.	Poco o nada.	Máxima capacidad de cambio, máximo coste y más riesgo de tocar cosas que funcionaban.
LoRA / QLoRA	Matrices pequeñas A y B.	Matriz grande W y la mayor parte del modelo.	Adaptas el comportamiento con una pieza pequeña que se puede guardar aparte.
Adapters	Módulos añadidos entre capas.	La red base.	Útil si quieres muchas tareas sobre una base común.
Prompt tuning / prefix tuning	Vectores continuos aprendidos.	Los pesos del modelo.	Cambia la entrada interna, no el modelo base.
IA3	Vectores que escalan activaciones.	Casi todos los pesos.	Ajuste muy ligero: más fino, pero también menos expresivo.
DPO	Pesos o adaptadores a partir de preferencias.	Depende de la implementación.	Enseña qué respuesta preferimos cuando hay varias plausibles.
Destilación	El estudiante.	El profesor.	El grande enseña señales; el pequeño aprende a imitar lo suficiente.

Esa es la diferencia real entre técnicas: **por dónde dejamos pasar el gradiente**. Y por eso la pérdida no basta como única brújula. Puede bajar la pérdida en tus ejemplos y aun así empeorar una tarea real si el conjunto de evaluación no representa el uso diario.



No todo es LoRA: mapa de técnicas de adaptación

LoRA y QLoRA son importantes, pero no son las únicas opciones. Esta tabla sirve como brújula rápida. No hace falta memorizar todos los nombres; lo importante es saber qué parte se entrena, qué problema resuelve y qué puede salir mal.

TÉCNICA	QUÉ SE ENTRENA	CUÁNDO ENCAJA	CUIDADO
Prompting	Nada. Solo contexto escrito.	Primer intento, prototipos, tareas simples.	Si el contexto cambia mucho, puede volverse frágil.
RAG	Normalmente nada del modelo; se entrena o configura recuperación.	Conocimiento privado o cambiante.	Si recupera mal, el modelo contesta con mala base.
Fine-tuning completo	Todos o casi todos los pesos.	Cambio profundo de dominio o comportamiento.	Caro; riesgo de olvidar capacidades útiles.
SFT / instruction tuning	Pesos del modelo, con pares instrucción-respuesta.	Enseñar formato, estilo y seguimiento de tareas.	La calidad de ejemplos manda más que el nombre de la técnica.
Adapters	Módulos pequeños insertados en capas.	Muchas tareas con una base compartida.	Añaden algo de latencia y complejidad.[21]
Prompt tuning	Vectores de prompt continuos.	Modelos grandes, tareas concretas y bajo coste.	No es texto legible; es un prompt aprendido.[22]
Prefix tuning	Vectores de prefijo para condicionar capas.	Generación controlada con pocos parámetros.	Requiere saber dónde insertar el prefijo.[23]
P-tuning v2	Prompts continuos profundos, a menudo en varias capas.	Cuando quieres prompt tuning más expresivo en tareas de comprensión.	Sigue siendo menos interpretable que ejemplos escritos.[24]
BitFit	Principalmente sesgos del modelo.	Prueba barata cuando tienes pocos datos.	Puede quedarse corto si la tarea exige cambiar representaciones profundas.[25]
LoRA	Matrices pequeñas A y B.	Ajuste eficiente y práctico en LLMs.	Depende de rango, capas elegidas y datos.
QLoRA	LoRA sobre modelo base cuantizado.	Ajustar modelos grandes con poca memoria.	Cuantización y configuración importan mucho.
IA3	Vectores que escalan activaciones.	Ajuste muy ligero con pocos ejemplos.	Menos flexible que métodos con más parámetros.[26]
AdaLoRA	Bajo rango con presupuesto adaptativo.	Cuando no quieres repartir el mismo rango a todas las matrices.	Más configuración; no siempre merece la complejidad.[27]
DoRA	Magnitud y dirección del peso, usando bajo rango para la dirección.	Cuando LoRA se queda algo corto y quieres más capacidad.	Es una variante más especializada; conviene validar contra LoRA simple.[28]

TÉCNICA	QUÉ SE ENTRENA	CUÁNDO ENCAJA	CUIDADO
DPO / preferencias	Pesos o adaptadores usando pares preferido/rechazado.	Alinear estilo y preferencias tras SFT.	No sustituye datos correctos ni evaluación.[29]
Destilación	Un estudiante que imita al profesor.	Reducir tamaño o coste.	Puede perder capacidades fuera de la tarea.
Cuantización	A veces nada; a veces calibración.	Reducir memoria e inferencia.	Puede bajar calidad si se aprieta demasiado.

Una regla práctica: si no tienes evaluación, no elijas técnica todavía. Primero crea diez, cincuenta o cien casos reales que puedas revisar. Sin eso, fine-tuning, LoRA o DPO son formas sofisticadas de adivinar.

Más allá de DPO: alinear con preferencias hoy

DPO fue un punto de inflexión porque optimiza preferencias directamente, sin entrenar un modelo de recompensa aparte como en RLHF clásico. Pero no se quedó solo. A **10 de junio de 2026** conviene conocer la familia, no para usarlas todas, sino para no creer que «alinear» es una sola técnica.

MÉTODO	QUÉ NECESITA	QUÉ APORTA
RLHF	Modelo de recompensa entrenado más un bucle de refuerzo.	El esquema original; potente pero con más piezas que ajustar.
DPO	Pares preferido/rechazado.	Optimiza la preferencia directamente, sin modelo de recompensa.[30]
KTO	Solo una etiqueta buena/mala por respuesta, sin pares.	Útil cuando no tienes comparaciones, solo señales sueltas.[31]
ORPO	Pares de preferencia, pero sin modelo de referencia.	Une SFT y preferencias en un solo paso, más simple de operar.[32]
SimPO	Pares de preferencia, sin modelo de referencia.	Usa la probabilidad media de la secuencia como recompensa; más ligero en memoria.[33]

La lectura útil no es «cuál es la mejor», sino qué datos tienes. Si tienes pares claros, DPO u ORPO encajan. Si solo tienes señales sueltas de bueno o malo, KTO. Y, como siempre, la preferencia no arregla una base de datos pobre: ordena estilo y criterio sobre algo que ya funciona.

Casos cercanos: elegir sin ponerse solemne

La tienda online. El catálogo cambia cada día, pero el tono de atención al cliente debe ser siempre claro y amable. Para el catálogo, RAG: recuperar precio, stock, plazo y política vigente. Para el tono y el formato, quizá basten buenos ejemplos; si se repite mucho, LoRA o adapters pueden fijar esa forma de responder.

El departamento de compras. Recibe presupuestos en PDF y quiere extraer proveedor, importes, plazos y condiciones. Si el problema es leer documentos nuevos, empieza por extracción, validación y RAG. Si el problema es que el modelo devuelve JSON desordenado, SFT o LoRA con ejemplos muy buenos puede ayudar. Si las respuestas son largas y caras, destilar un modelo pequeño para esa tarea empieza a tener sentido.

La app que debe funcionar en un portátil normal. Aquí la pregunta no es solo «qué modelo sabe más», sino «qué modelo cabe, responde rápido y no quema presupuesto». Un modelo open weights cuantizado puede ser suficiente. Si además quieres una tarea muy concreta, QLoRA para adaptar y cuantización para servir son dos piezas que se hablan entre sí.

El equipo docente. Quiere un asistente para generar ejercicios de práctica con el estilo del curso. No necesita memorizar todos los materiales: necesita seguir una plantilla, respetar nivel y producir soluciones revisables. Puedes empezar con prompting y evaluación; si funciona pero se repite cada semana, una adaptación ligera puede ahorrar trabajo.

LoRA: adaptar sin tocar todo

LoRA, *Low-Rank Adaptation*, parte de una observación práctica: quizá no necesitamos actualizar toda una matriz grande para adaptar un modelo. Podemos congelar el peso original W y aprender una actualización de bajo rango.[34]

En una capa lineal:

$$y = Wx$$

LoRA usa:

$$y = Wx + \frac{\alpha}{r} BAx$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
W	Matriz original congelada.	No se entrena durante LoRA.
A	Matriz pequeña entrenable.	Reduce dimensión hacia rango r.
B	Matriz pequeña entrenable.	Vuelve a la dimensión original.
r	Rango de la adaptación.	$r = 8$, $r = 16$, $r = 64$.
α	Escala de la adaptación.	Controla fuerza de LoRA.
BA	Actualización de bajo rango.	Aproxima el cambio necesario en W.

Si W mide 4096×4096, tiene:

$$4096 \cdot 4096 = 16\,777\,216$$

parámetros.

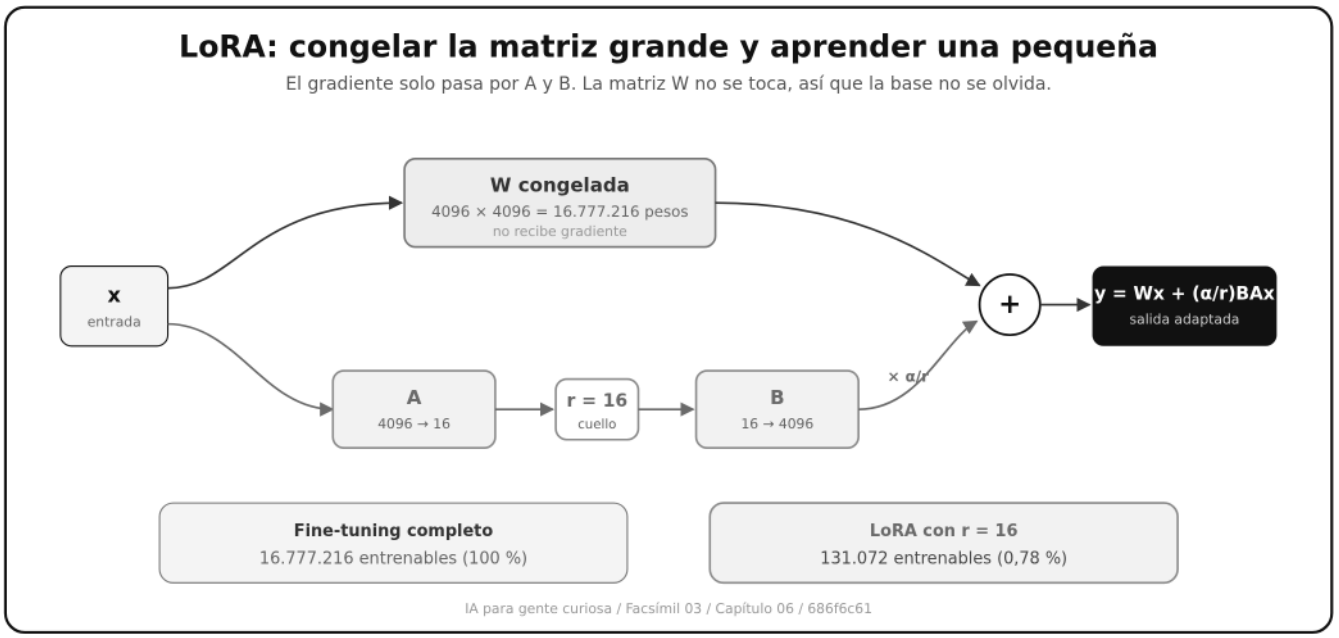
Con LoRA de rango $r = 16$:

$$16(4096 + 4096) = 131\,072$$

parámetros entrenables.

MÉTODO	PARÁMETROS ENTRENABLES EN LA MATRIZ DEL EJEMPLO
Fine-tuning completo	16 777 216
LoRA con $r = 16$	131 072

No es gratis ni siempre suficiente, pero cambia la escala del problema. Puedes adaptar un modelo grande entrenando una fracción pequeña de sus parámetros.



Una ventaja práctica aparece al servir el modelo. Como BA tiene la misma forma que W, puedes **fusionar** el adaptador sumándolo a los pesos base, $W' = W + \frac{\alpha}{r}BA$, y obtener un único modelo sin coste extra en inferencia. O puedes hacer lo contrario: mantener W compartida y **conmutar adaptadores** según la petición, sirviendo muchas variantes de tarea con una sola copia de la base en memoria. Esa flexibilidad, guardar un adaptador de pocos megabytes en lugar de un modelo entero, es parte de por qué LoRA se volvió tan común en producción.

QLoRA: adaptar con menos memoria

QLoRA combina cuantización y LoRA: mantiene el modelo base cuantizado y entrena adaptadores LoRA encima.[35]

La idea:

```
pesos base cuantizados y congelados + adaptadores LoRA entrenables
```

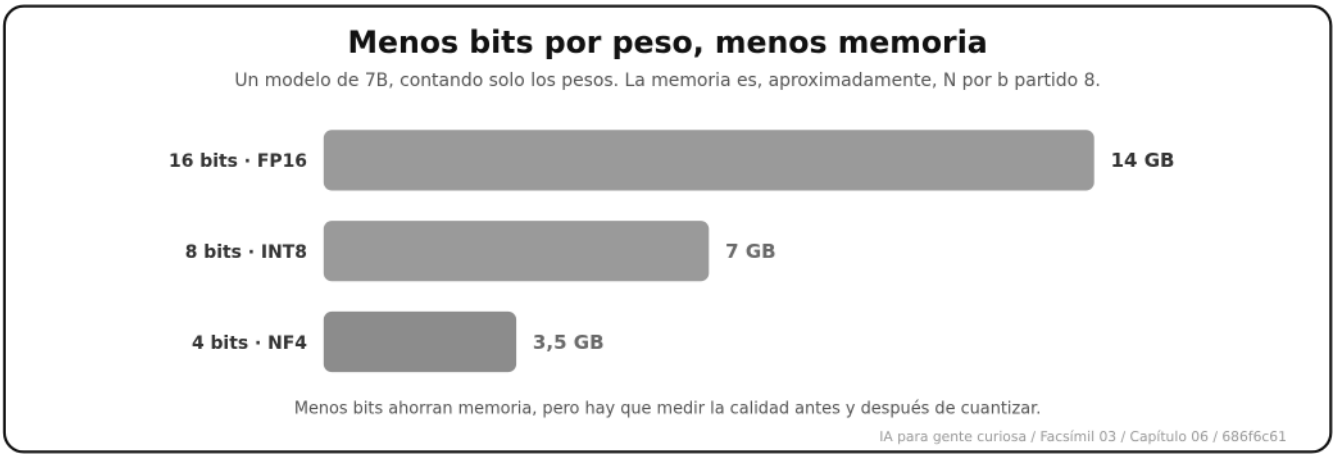
Si un modelo tiene N parámetros y cada parámetro usa b bits, la memoria aproximada solo para pesos es:

$$\text{memoria} \approx \frac{N \cdot b}{8}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
N	Número de parámetros.	7×10^9.
b	Bits por parámetro.	16, 8, 4.
8	Bits por byte.	Conversión a bytes.

Para un modelo de 7B:

PRECISIÓN	CÁLCULO APROXIMADO	MEMORIA APROXIMADA
16 bits	$7B \cdot 16/8$	14 GB
8 bits	$7B \cdot 8/8$	7 GB
4 bits	$7B \cdot 4/8$	3,5 GB



La realidad añade metadatos, activaciones, KV cache y sobrecostes del runtime, pero la intuición sirve: menos bits reducen memoria. En el capítulo 07 conectaremos esto con inferencia, hardware y ejecución local.

«Cuantizar a 4 bits» no es una sola receta. Hay métodos con nombre, y conviene reconocerlos porque tienen compromisos distintos. Tómalos como estado a **10 de junio de 2026**: los nombres y versiones cambian, pero la idea de cada familia es estable.

MÉTODO	IDEA CENTRAL	DÓNDE ENCAJA
GPTQ	Cuantización post-entrenamiento que ajusta peso a peso para minimizar el error de la capa.[36]	Servir modelos a 3-4 bits con poca caída de calidad.
AWQ	Protege los pesos más importantes mirando qué activaciones pesan más.[37]	Inferencia rápida a 4 bits en GPU.
NF4 (bitsandbytes)	Tipo de dato de 4 bits «normal-float» diseñado para pesos; es el que usa QLoRA.[38]	Entrenar adaptadores con poca memoria.
GGUF / llama.cpp	Formato de pesos cuantizados pensado para ejecutar en CPU y portátiles.	Edge, local y prototipos sin GPU.
FP8	Coma flotante de 8 bits con soporte en hardware reciente.	Entrenamiento e inferencia a gran escala.

Hay una diferencia importante que el capítulo 07 retomará: cuantizar **pesos** no es lo mismo que cuantizar **activaciones**. Algunos métodos tocan solo los pesos; otros, como SmoothQuant, reparten el rango entre pesos y activaciones para que el cálculo en enteros no se descontrole.[39] La regla práctica no cambia: cuantizar siempre se mide antes y después, porque apretar de más puede costar justo la capacidad que necesitabas.

Destilación: enseñar a un modelo más pequeño

Destilación significa entrenar un modelo estudiante para imitar señales de un modelo profesor. Hinton, Vinyals y Dean popularizaron la idea de transferir conocimiento usando las probabilidades suaves del profesor, no solo etiquetas duras.[40]

La intuición:

ETIQUETA DURA	DISTRIBUCIÓN SUAVE DEL PROFESOR
«La respuesta es París».	París 0,82; Lyon 0,10; Madrid 0,05; azul 0,03.

La segunda señal enseña más. No solo dice cuál es la respuesta más probable; también dice qué alternativas eran más plausibles.

Con temperatura T:

$$p_t^{(T)} = \mathrm{softmax}\left(\frac{z_t}{T}\right)$$

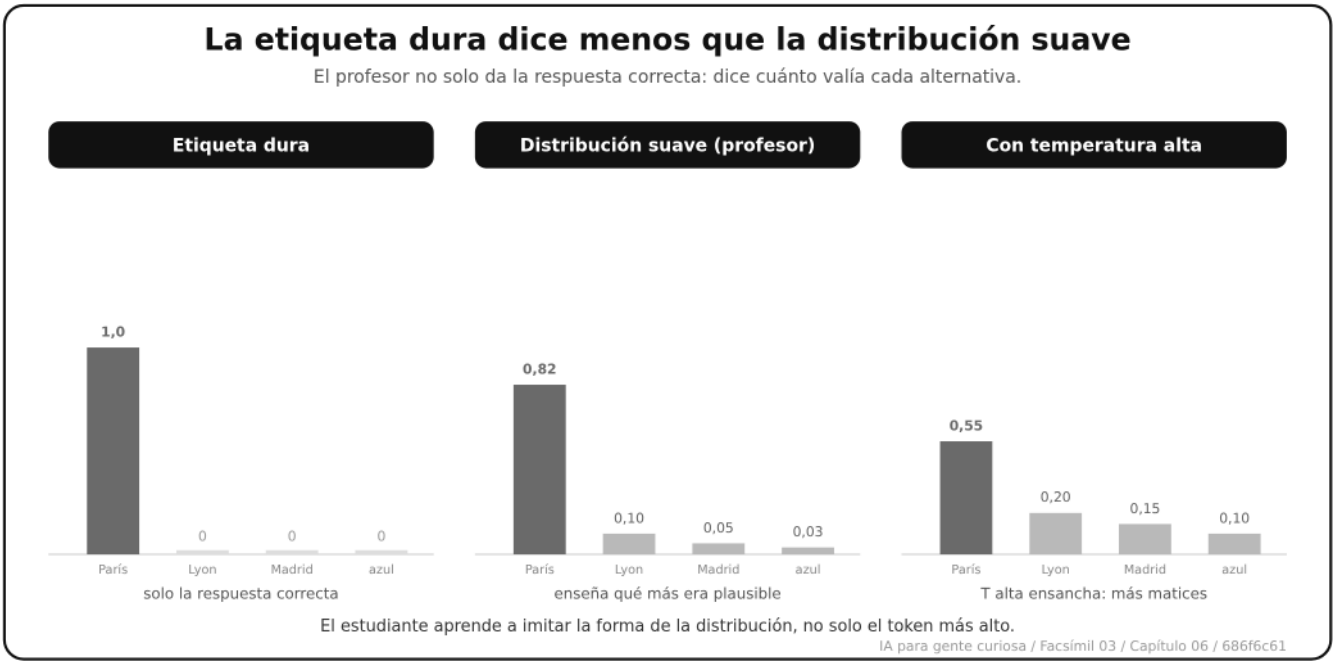
$$p_s^{(T)} = \mathrm{softmax}\left(\frac{z_s}{T}\right)$$

Una pérdida habitual combina etiqueta real y distilación:

$$\mathcal{L} = (1-\lambda)\,\mathrm{CE}(y, p_s) + \lambda T^2\,\mathrm{KL}(p_t^{(T)} \,\|\, p_s^{(T)})$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$p_t^{(T)}$	Distribución del profesor con temperatura.	Probabilidades suaves.
$p_s^{(T)}$	Distribución del estudiante con temperatura.	Lo que intenta imitar.
y	Etiqueta real o respuesta esperada.	Token correcto o clase correcta.
CE	Entropía cruzada.	Pérdida con etiqueta real.
KL	Divergencia KL.	Distancia entre distribuciones.
λ	Peso de la parte de distilación.	0,5, 0,7...
T	Temperatura.	Suaviza probabilidades.

DistilBERT es un ejemplo clásico: reduce tamaño y mejora velocidad manteniendo buena parte del rendimiento de BERT en tareas de comprensión.[41]



Un ejemplo entendible: la academia que prepara apuntes

Imagina una academia con una profesora experta y una ayudante nueva. La profesora corrige exámenes desde hace años. La ayudante todavía está aprendiendo.

Podemos enseñar a la ayudante de tres maneras:

MÉTODO	QUÉ RECIBE LA AYUDANTE	EQUIVALENTE EN MODELOS
Solo soluciones finales	«Esta respuesta está bien, está mal».	Etiquetas duras.
Explicaciones y matices	«Esta está bien; esta otra casi, pero confunde una fecha».	Distribuciones suaves o razonamiento.
Casos seleccionados	Ejercicios representativos del curso.	Datos curados para adaptación.

La destilación se parece al segundo caso. No obliga al estudiante a ser idéntico al profesor, pero intenta que aprenda su criterio de salida. En producto, esto puede servir para tener un modelo más barato o rápido que mantiene suficiente calidad en una tarea concreta.

Modo ingeniero: el presupuesto de adaptación en código

Las tres cuentas centrales del capítulo (parámetros entrenables de LoRA, gradiente de la pérdida y destilación con temperatura) caben en unas pocas líneas.

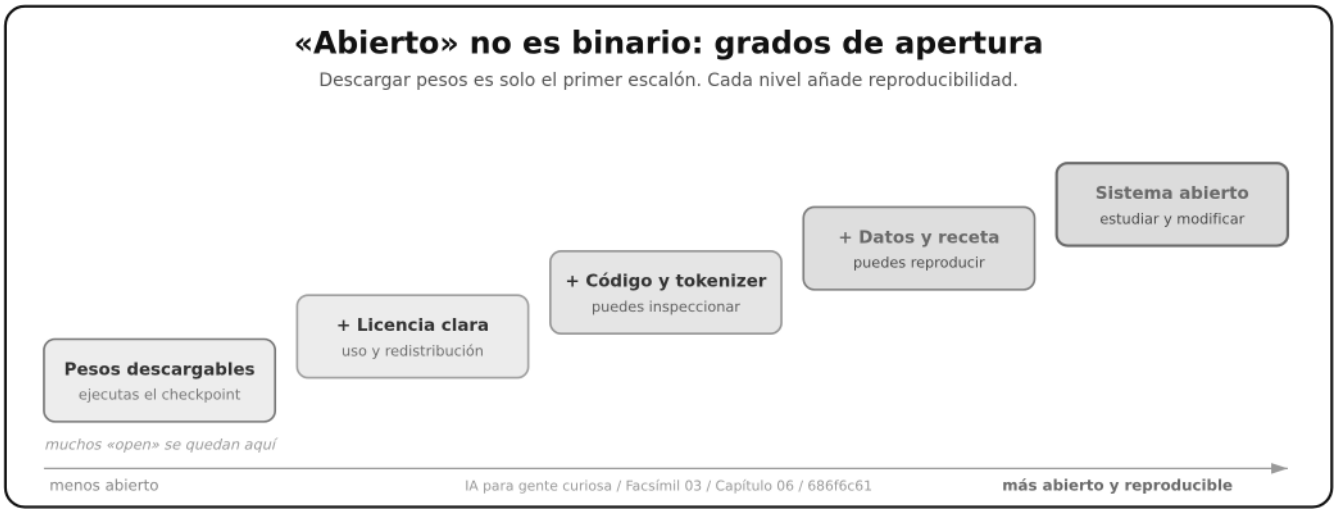
```
import numpy as np

def softmax(z):
    z = np.asarray(z, dtype=float)
    e = np.exp(z - z.max())
    return e / e.sum()

# 1) Parámetros entrenables: matriz 4096x4096
d_in = d_out = 4096
full = d_in * d_out                      # 16.777.216
lora = 16 * (d_in + d_out)               # r=16 -> 131.072
adapter = 2 * d_in * 64                  # cuello 64 -> 524.288
print(round(100 * lora / full, 3))       # 0.781  (%)
print(round(100 * adapter / full, 3))    # 3.125  (%)

# 2) Gradiente de la pérdida en los logits: p - y
logits = [2.0, 1.0, 0.0]                 # factura, rotos, saludo
target = 1                               # el token correcto es "rotos"
p = softmax(logits)                      # [0.665, 0.245, 0.090]
y = np.eye(len(logits))[target]
grad = p - y                             # [0.665, -0.755, 0.090]
loss = -np.log(p[target])                # 1.408
print(np.round(grad, 3), round(loss, 3))

# 3) Destilación: KL entre profesor y estudiante con temperatura T
T = 2.0
teacher = softmax(np.array([4.0, 2.0, 0.0]) / T)
student = softmax(np.array([3.0, 2.5, 0.5]) / T)
kl = float(np.sum(teacher * np.log(teacher / student)))
print(round(kl, 6))                      # 0.066603
```

El bloque deja ver tres ideas a la vez. LoRA entrena el 0,78 % de la matriz, no el 100 %. El gradiente `p - y` solo es negativo en el token correcto: «sube este, baja los demás». Y la destilación mide con una KL cuánto se aleja el estudiante de la distribución suave del profesor. Cambia el rango, los logits o la temperatura y verás moverse cada número.

Modelos abiertos: abrir qué, exactamente

La palabra «abierto» se usa demasiado deprisa. Conviene separar piezas:

PIEZA	PREGUNTA QUE DEBES HACER
Pesos	¿Puedo descargar y ejecutar el checkpoint?
Licencia	¿Puedo usarlo comercialmente? ¿Puedo redistribuir adaptaciones?
Código	¿Está el código de entrenamiento o inferencia?
Tokenizer	¿Está disponible y versionado?
Configuración	¿Conozco arquitectura, contexto, formato de chat y parámetros?
Datos	¿Sé qué datos o mezcla de datos se usaron?
Receta de entrenamiento	¿Puedo reproducir el proceso?
Evaluación	¿Hay benchmarks, limitaciones y resultados verificables?
Model card	¿Hay descripción de uso previsto, límites y supuestos?



La Open Source Initiative publicó en 2024 la versión 1.0 de su definición de Open Source AI, centrada en libertades para usar, estudiar, modificar y compartir sistemas de IA.[42] En la práctica, muchos modelos populares son **open weights**: los pesos están disponibles, pero no necesariamente todo el sistema cumple una definición estricta de open source.

Ejemplos de familias abiertas o de pesos disponibles, con matices:

FAMILIA	QUÉ ILUSTRA	MATIZ
Llama 3	Pesos y documentación técnica ampliamente usados.[43]	Licencia propia; no equivale automáticamente a open source completo.
Mistral 7B	Modelo eficiente de 7B con publicación técnica.[44]	Hay que mirar licencia y versión concreta.
Qwen2.5	Familia de modelos con informe técnico amplio.[45]	Cada tamaño y variante puede tener condiciones propias.

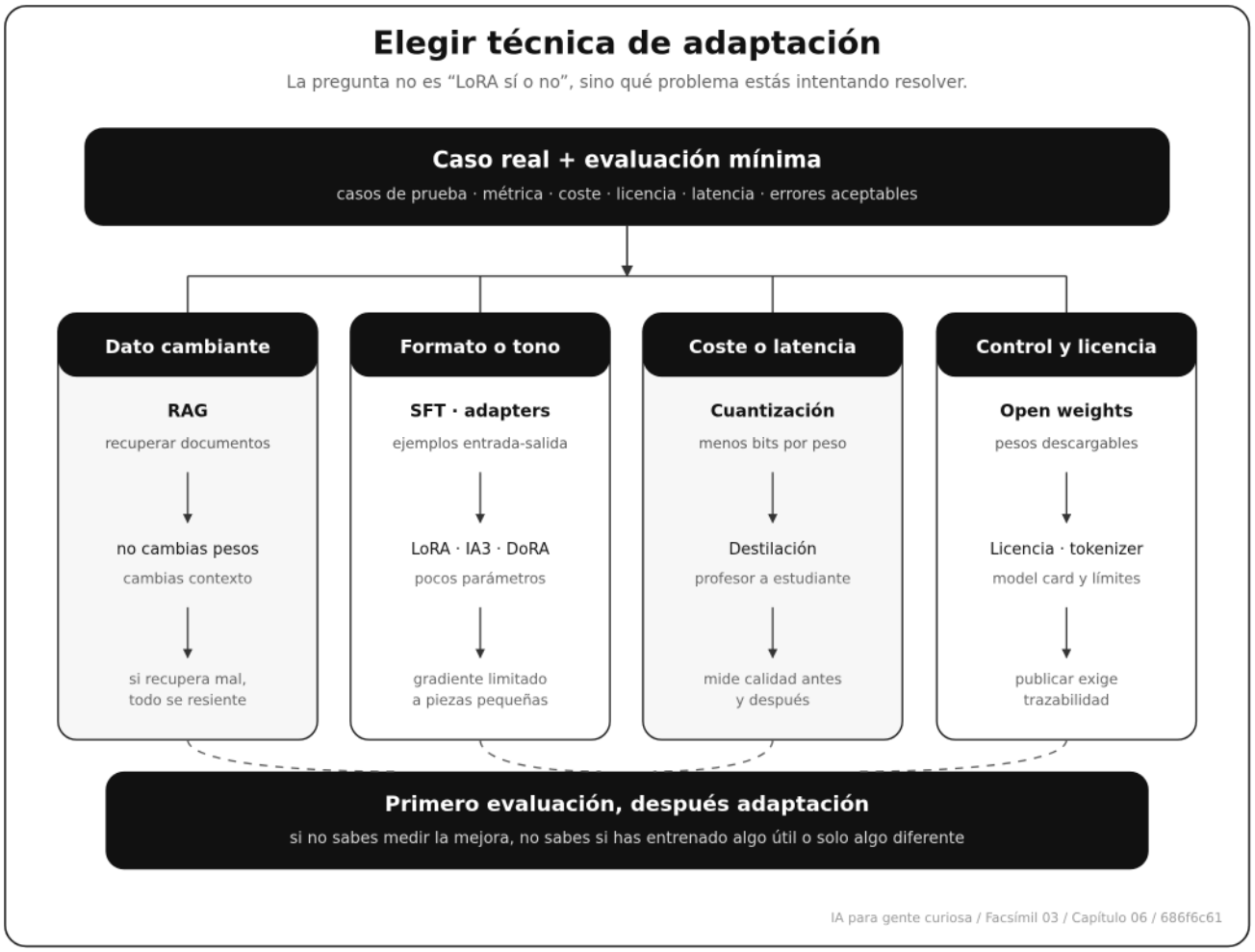
La conclusión importante: abierto no es binario. Hay grados de apertura y permisos concretos. Para uso profesional, mirar solo si «se puede descargar» es insuficiente.

Qué elegir según el caso

SITUACIÓN	PRIMERA OPCIÓN RAZONABLE	POR QUÉ
Necesitas datos actualizados o privados.	RAG antes que fine-tuning.	No quieres reentrenar para cada documento nuevo.
Necesitas formato muy específico.	Instruction tuning o ejemplos muy cuidados.	Enseñas patrón de salida.
Tienes pocos recursos de entrenamiento.	LoRA o QLoRA.	Entrenas pocos parámetros.
Necesitas baja latencia o edge.	Destilación, cuantización o modelo pequeño.	Reducen coste de inferencia.
Necesitas control local.	Modelo open weights con licencia compatible.	Puedes ejecutar y adaptar en tu infraestructura.
Necesitas máxima calidad general.	Modelo grande servido por API o modelo abierto fuerte bien evaluado.	A veces el coste compensa.

La decisión profesional no es «fine-tuning sí o no». Es una secuencia:

```
definir tarea -> crear evaluación -> probar base -> probar prompt/RAG -> adaptar si hace
falta -> comprimir si compensa -> revisar licencia
```



En el día a día

Un equipo de soporte. Tiene 3 000 respuestas históricas y quiere que el modelo use un tono concreto. Primero conviene crear evaluación y probar prompt/RAG. Si el problema es estilo y formato, LoRA puede bastar. Si el problema es conocimiento cambiante, RAG será más natural.

Un producto local. Necesita funcionar en una máquina pequeña. Aquí pesan cuantización y destilación. Tal vez un modelo de 7B cuantizado o un estudiante destilado sea más útil que un modelo enorme que no cabe en memoria.

Una empresa con restricciones de licencia. No basta con que el modelo esté en internet. Hay que revisar licencia, redistribución, uso comercial, obligación de atribución, datos, documentación y compatibilidad con el producto.

Un laboratorio universitario. Puede usar modelos abiertos para enseñar, reproducir experimentos y comparar adaptaciones. Pero debe enseñar también la diferencia entre pesos descargables y apertura completa.

Por qué debería importarte

Porque adaptar mal sale caro. Puedes gastar semanas afinando un modelo cuando bastaba con recuperar documentos. Puedes comprimir demasiado y perder justo la capacidad que necesitabas. Puedes usar un modelo con licencia incompatible con tu producto. O puedes entrenar LoRA sin evaluación y creer que ha mejorado porque responde con más confianza.

La buena noticia es que el mapa es manejable: primero define tarea y métricas; luego prueba una base; después decide si necesitas adaptación, destilación, cuantización o un modelo abierto específico. La arquitectura importa, pero el proceso de decisión importa igual.

También importa no enamorarse de una sigla. LoRA puede ser perfecta para un caso y excesiva para otro. Prompt tuning puede bastar si la tarea es estrecha. DPO puede ayudar cuando hay preferencias claras entre dos respuestas, pero no arregla una base de datos mala. Destilar puede ahorrar dinero, pero solo si el estudiante conserva lo que de verdad usas. La técnica correcta suele ser la más pequeña que resuelve el problema medido.

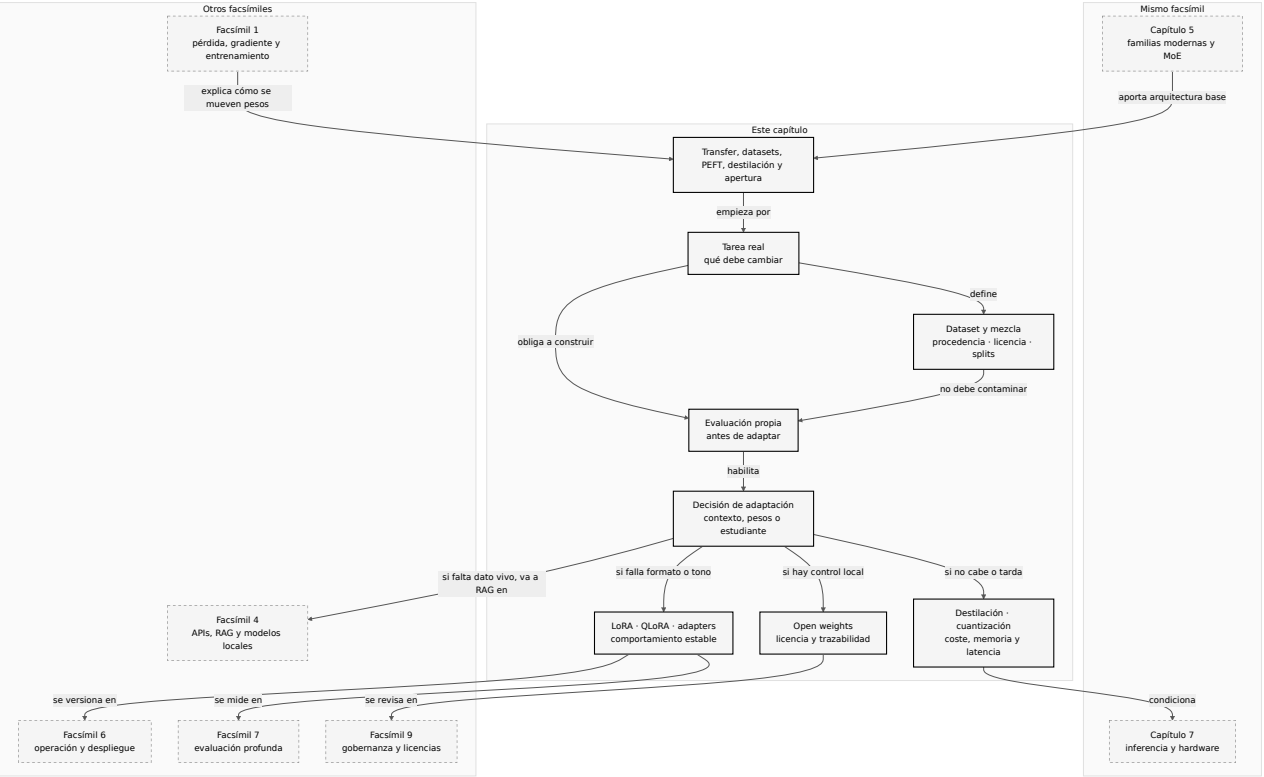
Dónde solía tropezar yo

ERROR	POR QUÉ ES UN ERROR	ANTÍDOTO
Tratar un dataset como una lista de ejemplos	El dataset define formato, pérdida, cobertura, sesgos, permisos y trazabilidad. Es parte de la arquitectura.	Léelo como leerías una model card: procedencia, estructura, licencia, mezcla y versión.
Mezclar entrenamiento y evaluación	Si el modelo ve tus preguntas de test, puede mejorar la nota sin mejorar la capacidad real.	Mantén splits separados, guarda hashes y crea conjuntos de regresión aparte.
Usar fine-tuning para meter conocimiento cambiante	Si los datos cambian cada semana, reentrenar no es la forma más cómoda de mantenerlos vivos.	Pregunta primero si el problema es conocimiento, formato, estilo o razonamiento.
Confundir LoRA con modelo completo	LoRA suele ser una adaptación que necesita el modelo base compatible.	Guarda siempre base, adaptador, tokenizer, configuración y versión.
Pensar que destilar conserva todo	Un estudiante pequeño puede perder capacidades fuera del dominio usado para destilar.	Evalúa dentro y fuera de la tarea principal.
Decir open source cuando solo hay pesos	Descargar pesos no implica conocer datos, receta, licencia completa ni reproducibilidad.	Usa categorías: pesos abiertos, código abierto, datos abiertos, sistema abierto.
Cuantizar sin medir	Menos bits ahorran memoria, pero pueden afectar calidad en tareas sensibles.	Compara respuestas, latencia, memoria y errores antes y después.

Cómo encaja todo

Este mapa conecta adaptación con todo el temario. El capítulo 6 no vive aislado: usa pérdidas y gradientes del facsímil 1, familias de arquitectura del capítulo 5 y prepara inferencia, operación, evaluación y gobernanza.

La idea que debe quedar es operativa: antes de tocar pesos hay que definir tarea, dataset, evaluación y coste. Después se decide qué parte se mueve: contexto, recuperador, adaptador, estudiante, precisión o modelo base.



Vocabulario aprendido

TÉRMINO	DEFINICIÓN
Transfer learning	Reutilizar un modelo entrenado como punto de partida para otra tarea.
Fine-tuning	Ajustar pesos de un modelo base con datos específicos.
PEFT	Familia de técnicas de ajuste eficiente de parámetros.
Gradiente de pérdida	Señal numérica que indica cómo cambiar parámetros para reducir el error.
Instruction tuning	Ajuste con ejemplos de instrucciones y respuestas deseadas.
Adapter	Módulos pequeños añadidos al modelo para adaptar tareas sin tocar toda la base.
Prompt tuning	Vectores de prompt continuos aprendidos que condicionan el modelo congelado.
Prefix tuning	Vectores de prefijo aprendidos para condicionar la generación sin actualizar todos los pesos.
P-tuning v2	Prompt tuning profundo con prompts continuos en varias capas.
BitFit	Ajuste de sesgos del modelo como técnica mínima de adaptación.
LoRA	Adaptación de bajo rango que entrena matrices pequeñas con el modelo base congelado.
QLoRA	LoRA sobre pesos base cuantizados para reducir memoria de entrenamiento.
IA3	Ajuste con vectores que escalan activaciones internas.
AdaLoRA	Variante que reparte el presupuesto de bajo rango según la importancia de cada matriz.
DoRA	Variante de LoRA que separa magnitud y dirección del peso para mejorar la adaptación.
DPO	Ajuste con pares de respuestas preferidas y no preferidas.
Destilación	Entrenar un modelo pequeño usando señales de un modelo mayor.
Cuantización	Reducir bits por peso o activación para ahorrar memoria y coste.
Open weights	Pesos descargables, no necesariamente apertura completa.

TÉRMINO	DEFINICIÓN
Modelo abierto	Modelo con permisos, documentación y artefactos suficientes para uso, estudio y adaptación.
Dataset de preentrenamiento	Corpus grande usado para aprender patrones generales antes de adaptar.
Dataset SFT	Pares de instrucción-respuesta para enseñar formato, tono y seguimiento de tareas.
Dataset de preferencias	Comparaciones entre respuestas para enseñar criterios de elección.
Dataset de razonamiento	Problemas con solución, pasos o verificador para entrenar o medir resolución estructurada.
Dataset multimodal	Datos que combinan texto con imágenes, audio, vídeo u otras señales.
Data mixture	Proporción de familias de datos en entrenamiento.
Data contamination	Presencia de ejemplos de evaluación dentro del entrenamiento.
Olvido catastrófico	Pérdida de capacidades previas cuando se adapta un modelo con datos estrechos.
Fusión de adaptadores	Sumar un adaptador LoRA a los pesos base para servir un único modelo sin coste extra en inferencia.
Cuantización post-entrenamiento	Reducir bits de un modelo ya entrenado con métodos como GPTQ, AWQ o NF4.

Antes de pasar página

- [ ] ¿Puedo explicar la diferencia entre prompt, RAG y fine-tuning? (Si no, vuelve a «Qué no es adaptar un modelo».)
- [ ] ¿Puedo explicar qué dataset usaría para preentrenamiento, SFT, preferencias, razonamiento, código y multimodalidad? (Si no, vuelve a «Datasets».)
- [ ] ¿Sé distinguir dataset de entrenamiento y dataset de evaluación? (Si no, vuelve a «Tipos de dataset con ejemplos».)
- [ ] ¿Sé leer un dataset como una decisión de arquitectura: procedencia, licencia, formato, filtros, mezcla y trazabilidad? (Si no, vuelve a «Qué debe mirar un ingeniero».)
- [ ] ¿Entiendo qué significa $p_i - y_i$ en el gradiente de una pérdida? (Si no, vuelve a «La pérdida y los gradientes».)

- [ ] ¿Puedo distinguir LoRA, QLoRA, adapters, prompt tuning, IA3, AdaLoRA, DoRA y DPO sin quedarme solo con siglas? (Si no, vuelve a «No todo es LoRA: mapa de técnicas de adaptación».)
- [ ] ¿Sé cuándo probar LoRA antes que fine-tuning completo? (Si no, vuelve a «LoRA».)
- [ ] ¿Puedo calcular cuántos parámetros entrena LoRA en una matriz pequeña? (Si no, vuelve a «LoRA: adaptar sin tocar todo».)
- [ ] ¿Entiendo por qué QLoRA reduce memoria? (Si no, vuelve a «QLoRA».)
- [ ] ¿Sé qué es el olvido catastrófico y por qué congelar la base con PEFT lo reduce? (Si no, vuelve a «El olvido catastrófico».)
- [ ] ¿Distingo GPTQ, AWQ, NF4 y GGUF y para qué sirve cada uno? (Si no, vuelve a «Cuantización por dentro».)
- [ ] ¿Conozco alternativas a DPO como KTO, ORPO o SimPO según los datos que tengo? (Si no, vuelve a «Más allá de DPO».)
- [ ] ¿He reproducido en código los parámetros de LoRA, el gradiente y la KL de destilación? (Si no, vuelve a «Modo ingeniero».)
- [ ] ¿Puedo explicar qué aprende un estudiante en destilación? (Si no, vuelve a «Destilación».)
- [ ] ¿Sé distinguir open weights de open source completo? (Si no, vuelve a «Modelos abiertos».)
- [ ] ¿Sé qué cambia al variar el rango de LoRA, los bits de cuantización o la temperatura de destilación? (Si no, vuelve a «LoRA» y «Destilación».)
- [ ] ¿Puedo conectar adaptación, compresión y apertura con inferencia y evaluación? (Si no, vuelve a «Cómo encaja todo».)

En resumen

IDEA FUERZA	DETALLE
Casi nadie entrena un LLM grande desde cero.	Lo habitual es reutilizar una base y adaptarla con datos, instrucciones o herramientas.
Los datasets son arquitectura.	Preentrenamiento, SFT, preferencias, razonamiento, código y multimodalidad tienen formatos, pérdidas y riesgos distintos.
La pérdida baja moviendo gradientes.	Cada técnica decide qué partes reciben esa señal: todo el modelo, adaptadores, matrices LoRA, prompts continuos o un estudiante.
LoRA y QLoRA cambian la escala del ajuste, pero no están solas.	Adapters, prompt tuning, prefix tuning, BitFit, IA3, AdaLoRA, DoRA y DPO son piezas distintas del mismo mapa.
Destilación y cuantización comprimen, pero hay que medir.	Ahorran coste y memoria, aunque pueden perder capacidades.
Abierto no significa solo descargable.	Pesos, licencia, tokenizer, datos, receta, evaluación y model card importan.

Para saber más

Dettmers, T., Pagnoni, A., Holtzman, A. y Zettlemoyer, L. (2023). QLoRA: Efficient finetuning of quantized LLMs. *Advances in Neural Information Processing Systems 36*. https://arxiv.org/abs/2305.14314

Devlin, J., Chang, M.-W., Lee, K. y Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. *Proceedings of NAACL-HLT*, 4171-4186. https://doi.org/10.18653/v1/N19-1423

Gao, L. y otros (2020). *The Pile: An 800GB Dataset of Diverse Text for Language Modeling*. https://arxiv.org/abs/2101.00027

Longpre, S. y otros (2023). *The Flan Collection: Designing Data and Methods for Effective Instruction Tuning*. https://arxiv.org/abs/2301.13688

Databricks. (2023). *databricks-dolly-15k*. https://huggingface.co/datasets/databricks/databricks-dolly-15k

Bai, Y. y otros (2022). *Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback*. https://arxiv.org/abs/2204.05862

Cui, G. y otros (2023). *UltraFeedback: Boosting Language Models with High-quality Feedback*. https://arxiv.org/abs/2310.01377

Cobbe, K. y otros (2021). *Training Verifiers to Solve Math Word Problems*. https://arxiv.org/abs/2110.14168

Kocetkov, D. y otros (2022). *The Stack: 3 TB of permissively licensed source code*. https://arxiv.org/abs/2211.15533

Schuhmann, C. y otros (2022). *LAION-5B: An open large-scale dataset for training next generation image-text models*. https://arxiv.org/abs/2210.08402

Liu, H. y otros (2023). *Visual Instruction Tuning*. https://arxiv.org/abs/2304.08485

Hendrycks, D. y otros (2021). *Measuring Massive Multitask Language Understanding*. https://arxiv.org/abs/2009.03300

Thakur, N. y otros (2021). *BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models*. https://arxiv.org/abs/2104.08663

Muennighoff, N. y otros (2023). *MTEB: Massive Text Embedding Benchmark*. https://arxiv.org/abs/2210.07316

Dubey, A. y otros (2024). *The Llama 3 herd of models*. https://arxiv.org/abs/2407.21783

Hinton, G., Vinyals, O. y Dean, J. (2015). *Distilling the knowledge in a neural network*. https://arxiv.org/abs/1503.02531

Houlsby, N. y otros (2019). Parameter-efficient transfer learning for NLP. *International Conference on Machine Learning*. https://arxiv.org/abs/1902.00751

Hu, E. J. y otros (2022). LoRA: Low-rank adaptation of large language models. *International Conference on Learning Representations*. https://arxiv.org/abs/2106.09685

Jacob, B. y otros (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2704-2713. https://doi.org/10.1109/CVPR.2018.00286

Jiang, A. Q. y otros (2023). *Mistral 7B*. https://arxiv.org/abs/2310.06825

Lester, B., Al-Rfou, R. y Constant, N. (2021). The power of scale for parameter-efficient prompt tuning. *Proceedings of EMNLP*, 3045-3059. https://doi.org/10.18653/v1/2021.emnlp-main.243

Li, X. L. y Liang, P. (2021). Prefix-tuning: Optimizing continuous prompts for generation. *Proceedings of ACL*, 4582-4597. https://doi.org/10.18653/v1/2021.acl-long.353

Liu, H. y otros (2022). *Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning*. https://arxiv.org/abs/2205.05638

Liu, S.-Y. y otros (2024). DoRA: Weight-decomposed low-rank adaptation. *International Conference on Machine Learning*. https://arxiv.org/abs/2402.09353

Liu, X. y otros (2022). P-Tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *Proceedings of ACL*. https://arxiv.org/abs/2110.07602

Open Source Initiative. (2024). *The Open Source AI Definition -- 1.0*. https://opensource.org/ai/open-source-ai-definition

Ouyang, L. y otros (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems 35*, 27730-27744. https://arxiv.org/abs/2203.02155

Rafailov, R. y otros (2023). Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems 36*. https://arxiv.org/abs/2305.18290

Raffel, C. y otros (2020). Exploring the limits of transfer learning with a unified text-to-text Transformer. *Journal of Machine Learning Research, 21*(140), 1-67. https://www.jmlr.org/papers/v21/20-074.html

Sanh, V., Debut, L., Chaumond, J. y Wolf, T. (2019). *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. https://arxiv.org/abs/1910.01108

Yang, A. y otros (2024). *Qwen2.5 technical report*. https://arxiv.org/abs/2412.15115

Ben-Zaken, E., Ravfogel, S. y Goldberg, Y. (2022). BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *Proceedings of ACL*. https://arxiv.org/abs/2106.10199

Zhang, Q. y otros (2023). AdaLoRA: Adaptive budget allocation for parameter-efficient fine-tuning. *International Conference on Learning Representations*. https://arxiv.org/abs/2303.10512

Notas

1. Devlin, J., Chang, M.-W., Lee, K. y Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. *Proceedings of NAACL-HLT*, 4171-4186. https://doi.org/10.18653/v1/N19-1423. BERT popularizó el esquema de preentrenar representaciones de lenguaje y ajustarlas en tareas posteriores. ↩

2. Raffel, C. y otros (2020). Exploring the limits of transfer learning with a unified text-to-text Transformer. *Journal of Machine Learning Research, 21*(140), 1-67. https://www.jmlr.org/papers/v21/20-074.html. T5 estudia transferencia en un marco unificado donde muchas tareas se expresan como entrada de texto y salida de texto. ↩

3. Ouyang, L. y otros (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems 35*, 27730-27744. https://arxiv.org/abs/2203.02155. El trabajo muestra cómo el ajuste con instrucciones y preferencias humanas mejora la capacidad de seguir peticiones. ↩

4. Raffel y otros (2020) crearon C4 para T5 como corpus web filtrado. Gao, L. y otros (2020). *The Pile: An 800GB Dataset of Diverse Text for Language Modeling*. https://arxiv.org/abs/2101.00027. ↩

5. Kocetkov, D. y otros (2022). *The Stack: 3 TB of permissively licensed source code*. https://arxiv.org/abs/2211.15533. ↩

6. Longpre, S. y otros (2023). *The Flan Collection*. https://arxiv.org/abs/2301.13688. Databricks. (2023). *databricks-dolly-15k*. https://huggingface.co/datasets/databricks/databricks-dolly-15k. ↩

7. Bai, Y. y otros (2022). *Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback*. https://arxiv.org/abs/2204.05862. Cui, G. y otros (2023). *UltraFeedback*. https://arxiv.org/abs/2310.01377. ↩

8. Cobbe, K. y otros (2021). *Training Verifiers to Solve Math Word Problems*. https://arxiv.org/abs/2110.14168. ↩

9. Schuhmann, C. y otros (2022). *LAION-5B*. https://arxiv.org/abs/2210.08402. Liu, H. y otros (2023). *Visual Instruction Tuning*. https://arxiv.org/abs/2304.08485. ↩

.0. Hendrycks, D. y otros (2021). *Measuring Massive Multitask Language Understanding*. https://arxiv.org/abs/2009.03300. BEIR y MTEB aparecen en los capítulos de embeddings y evaluación. ↩

.1. Hugging Face. (2026). *DataTrove*. https://github.com/huggingface/datatrove. La documentación lo presenta como una librería para procesar, filtrar y deduplicar texto a gran escala, con bloques de estadísticas, filtros, tokens y deduplicación. ↩

.2. NVIDIA. (2026). *Overview of NeMo Curator*. https://docs.nvidia.com/nemo/curator/latest/about. La documentación describe una plataforma abierta para curation de datos escalable y consciente de privacidad para datasets de entrenamiento de IA generativa. ↩

.3. Cleanlab. (2026). *Discover Bad Data/Responses in any LLM/Human-Written Dataset*. https://help.cleanlab.ai/tlm/use-cases/instruction_tuning_data/. El tutorial muestra cómo puntuar pares prompt-respuesta y localizar ejemplos de baja calidad en datasets de instruction tuning o evals. ↩

.4. Deepchecks. (2026). *Welcome to Deepchecks*. https://docs.deepchecks.com/0.8/getting-started/welcome.html. La documentación lo define como herramienta para validar datos y modelos: integridad, distribuciones, splits y comparación entre modelos. ↩

.5. Hugging Face. (2026). *Evaluate*. https://huggingface.co/docs/evaluate/index. La documentación lo presenta como librería para evaluar modelos y datasets con métodos de evaluación de NLP, visión, RL y otros dominios. ↩

.6. Hugging Face. (2026). *Lighteval*. https://huggingface.co/docs/lighteval/index. La documentación lo describe como toolkit para evaluar LLMs con distintos backends, tareas, métricas y resultados detallados. ↩

.7. EleutherAI. (2026). *Language Model Evaluation Harness*. https://github.com/EleutherAI/lm-evaluation-harness. El repositorio lo describe como framework unificado para probar modelos generativos en muchas tareas, con benchmarks públicos y soporte para modelos locales, APIs y adaptadores. ↩

.8. Liang, P. y otros (2022). *Holistic Evaluation of Language Models*. https://arxiv.org/abs/2211.09110. HELM propone evaluar modelos con múltiples escenarios y métricas para evitar una lectura estrecha del rendimiento. ↩

.9. McCloskey, M. y Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation, 24*, 109-165. El trabajo describe cómo entrenar una red en una tarea nueva puede borrar lo aprendido en una anterior. ↩

0. Kirkpatrick, J. y otros (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences, 114*(13), 3521-3526. https://doi.org/10.1073/pnas.1611835114. El método estima la importancia de cada peso y penaliza moverlo si era relevante para tareas anteriores. ↩

1. Houlsby, N. y otros (2019). Parameter-efficient transfer learning for NLP. *International Conference on Machine Learning*. https://arxiv.org/abs/1902.00751. El trabajo propone adaptar modelos de lenguaje con módulos pequeños manteniendo la mayoría de parámetros congelados. ↩

2. Lester, B., Al-Rfou, R. y Constant, N. (2021). The power of scale for parameter-efficient prompt tuning. *Proceedings of EMNLP*, 3045-3059. https://doi.org/10.18653/v1/2021.emnlp-main.243. El método aprende soft prompts manteniendo congelado el modelo base. ↩

3. Li, X. L. y Liang, P. (2021). Prefix-tuning: Optimizing continuous prompts for generation. *Proceedings of ACL*, 4582-4597. https://doi.org/10.18653/v1/2021.acl-long.353. Prefix tuning optimiza vectores continuos de prefijo con el modelo base congelado. ↩

4. Liu, X. y otros (2022). P-Tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *Proceedings of ACL*. https://arxiv.org/abs/2110.07602. P-tuning v2 muestra que prompts continuos bien optimizados pueden acercarse al fine-tuning con pocos parámetros. ↩

25. Ben-Zaken, E., Ravfogel, S. y Goldberg, Y. (2022). BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *Proceedings of ACL*. https://arxiv.org/abs/2106.10199. BitFit ajusta solo términos de sesgo y aun así puede competir en algunos escenarios. ↩

26. Liu, H. y otros (2022). *Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning*. https://arxiv.org/abs/2205.05638. El trabajo introduce IA3, que aprende vectores de escala para adaptar activaciones con pocos parámetros. ↩

27. Zhang, Q. y otros (2023). AdaLoRA: Adaptive budget allocation for parameter-efficient fine-tuning. *International Conference on Learning Representations*. https://arxiv.org/abs/2303.10512. AdaLoRA reparte el presupuesto entrenable según la importancia estimada de cada matriz. ↩

28. Liu, S.-Y. y otros (2024). DoRA: Weight-decomposed low-rank adaptation. *International Conference on Machine Learning*. https://arxiv.org/abs/2402.09353. DoRA separa magnitud y dirección para acercarse más al fine-tuning completo con pocos parámetros. ↩

29. Rafailov, R. y otros (2023). Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems 36*. https://arxiv.org/abs/2305.18290. DPO optimiza preferencias directamente sin entrenar un modelo de recompensa separado. ↩

30. Rafailov, R. y otros (2023). Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems 36*. https://arxiv.org/abs/2305.18290. ↩

31. Ethayarajh, K. y otros (2024). *KTO: Model alignment as prospect theoretic optimization*. https://arxiv.org/abs/2402.01306. ↩

32. Hong, J., Lee, N. y Thorne, J. (2024). *ORPO: Monolithic preference optimization without reference model*. https://arxiv.org/abs/2403.07691. ↩

33. Meng, Y., Xia, M. y Chen, D. (2024). *SimPO: Simple preference optimization with a reference-free reward*. https://arxiv.org/abs/2405.14734. ↩

34. Hu, E. J. y otros (2022). LoRA: Low-rank adaptation of large language models. *International Conference on Learning Representations*. https://arxiv.org/abs/2106.09685. LoRA congela los pesos preentrenados e introduce matrices entrenables de bajo rango para adaptar modelos grandes con muchos menos parámetros. ↩

35. Dettmers, T., Pagnoni, A., Holtzman, A. y Zettlemoyer, L. (2023). QLoRA: Efficient finetuning of quantized LLMs. *Advances in Neural Information Processing Systems 36*. https://arxiv.org/abs/2305.14314. QLoRA reduce memoria al ajustar modelos grandes cuantizados en 4 bits mientras entrena adaptadores de bajo rango. ↩

36. Frantar, E., Ashkboos, S., Hoefler, T. y Alistarh, D. (2022). *GPTQ: Accurate post-training quantization for generative pre-trained transformers*. https://arxiv.org/abs/2210.17323. El método cuantiza con información de segundo orden para reducir la pérdida de calidad a 3-4 bits. ↩

37. Lin, J. y otros (2023). *AWQ: Activation-aware weight quantization for LLM compression and acceleration*. https://arxiv.org/abs/2306.00978. AWQ preserva un pequeño porcentaje de pesos salientes según las activaciones para mantener calidad. ↩

38. Dettmers, T. y otros (2023). *QLoRA: Efficient finetuning of quantized LLMs*. https://arxiv.org/abs/2305.14314. Introduce NF4 y la doble cuantización para entrenar adaptadores sobre una base de 4 bits. ↩

39. Xiao, G. y otros (2023). *SmoothQuant: Accurate and efficient post-training quantization for large language models*. https://arxiv.org/abs/2211.10438. Desplaza dificultad de las activaciones a los pesos para habilitar cuantización entera de ambos. ↩

40. Hinton, G., Vinyals, O. y Dean, J. (2015). *Distilling the knowledge in a neural network*. https://arxiv.org/abs/1503.02531. El trabajo propone entrenar modelos más pequeños usando la distribución de salida de modelos más grandes o conjuntos de modelos. ↩

41. Sanh, V., Debut, L., Chaumond, J. y Wolf, T. (2019). *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. https://arxiv.org/abs/1910.01108. DistilBERT usa destilación durante preentrenamiento para obtener un modelo más pequeño y rápido que conserva gran parte de las capacidades de BERT. ↩

42. Open Source Initiative. (2024). *The Open Source AI Definition -- 1.0*. https://opensource.org/ai/open-source-ai-definition. La definición busca aclarar qué condiciones debería cumplir un sistema de IA para considerarse open source. ↩

43. Dubey, A. y otros (2024). *The Llama 3 herd of models*. https://arxiv.org/abs/2407.21783. El informe presenta modelos Llama 3 preentrenados y postentrenados, incluyendo tamaños grandes y variantes para distintos usos. ↩

44. Jiang, A. Q. y otros (2023). *Mistral 7B*. https://arxiv.org/abs/2310.06825. El trabajo presenta un modelo de 7B con decisiones de eficiencia como sliding window attention y grouped-query attention. ↩

45. Yang, A. y otros (2024). *Qwen2.5 technical report*. https://arxiv.org/abs/2412.15115. El informe describe la familia Qwen2.5 y resultados en múltiples tareas. ↩