UNIVERSIDAD DE CHILE

Inteligencia Artificial Generativa

Let's talk about hype stuff

Valentin Barriere // Clemente Henriquez

Universidad de Chile – DCC

Diplomado de Postítulo en Inteligencia Artificial, Primavera 2025

Efficiency and Deployment of Generative Models

¿Dónde estamos hoy?

Outline : ¿Dónde estamos hoy?

El Modelo Base

LLM

Ya sabemos qué es un LLM base y cómo se entrena con pretraining en enormes cantidades de texto. Sin embargo, un modelo que simplemente predice el siguiente token no es directamente útil para usuarios finales.

En la clase anterior vimos como un modelo de lenguage es una **distribución de probabilidad sobre secuencia de tokens**

Temperatura

Generación Condicional

Generalmente, podemos realizar generación condicional especificando alguna secuencia prefijo $x_{1:i}$ (llamada **prompt**) y muestreando el resto $x_{i+1:L}$ (llamado **completación**).

Por ejemplo, generando con $T = 0$ produce:

$$\underbrace{\text{the, mouse, ate}}_{\text{prompt}} \overset{T=0}{\rightsquigarrow} \underbrace{\text{the, cheese}}_{\text{completion}}$$

Ejemplo Modelo Base

Ahora, esto no es tan directo cuando tenemos un modelo de miles de millones de parametros, y de buenas a primeras, puede que no tenga el comportamiento que esperamos

Q: ¿Cuál es la capital de Chile?

A: ¿Cuál es la capital de Chile? (A) Santiago (B) Buenos Aires (C) Sólofo (D) Madrid (E) Guayaquil - **Answer:** (A) Santiago

Modelo: tiiuae/Falcon-E-1B-Base

Ejemplos Modelo Base

Q: Traduce al inglés: Hola, ¿cómo estás?

A: Traduce al inglés: Hola, ¿cómo estás? —Estás en Nuevo Yorkshire—

Suma 2 + 2

Suma 2 + 2 = 4 - **Answer:** The correct answer is: - B) 5 + 1 = 6

Modelo: tiiuae/Falcon-E-1B-Base

Visión General de la Clase

Hoy queremos mejorar el modelo base. Para esto veremos parte del ciclo que compone desde tener un modelo base entrenado hasta ponerlo en producción de manera eficiente.

1. **Post Training:** Cómo hacer que los modelos sean realmente útiles (Repaso SFT, DPO, RLHF, Constitutional AI y Alineamiento)
2. **Requerimientos Computacionales:** Análisis de recursos necesarios para ejecutar LLMs
3. **Técnicas de Optimización y Eficiencia**: cómo entrenar modelos con menos recursos (LoRA - QLoRa - Quantization - Pruning)
4. **Deployment Strategies:** Estrategias prácticas para producción

Post Training

Outline : Post Training

¿Dónde estamos hoy?

Post Training

De Alignment a Efficiency:
Recursos Computacionales

Técnicas de Eficiencia para LLMs

Deployment y Producción

Post Training

Objetivo

Lo que se busca en esta sección es hacer que el **comportamiento** del modelo se **alinee con el uso que le va a dar un usuario**. Esto significa que el modelo tenga la capacidad de responder a preguntas, seguir instrucciones, buscar informacion, etc.

Tambien nos interesa **evitar** comportamientos no deseados como los que podrían ser dañinos para un usuario

Comportamiento esperado

User: ¿Cuál es la capital de Francia?

Post Training

Objetivo

Lo que se busca en esta sección es hacer que el **comportamiento** del modelo se **alinee con el uso que le va a dar un usuario**. Esto significa que el modelo tenga la capacidad de responder a preguntas, seguir instrucciones, buscar informacion, etc.

Tambien nos interesa **evitar** comportamientos no deseados como los que podrían ser dañinos para un usuario

Comportamiento esperado

User: ¿Cuál es la capital de Francia?

- **Modelo Base**: ¿Cuál es la capital de Fancia? (A) Santiago (B) París (C) Sólofo (D) Madrid (E) Guayaquil - **Answer:** (A) Santiago

Post Training

Objetivo

Lo que se busca en esta sección es hacer que el **comportamiento** del modelo se **alinee con el uso que le va a dar un usuario**. Esto significa que el modelo tenga la capacidad de responder a preguntas, seguir instrucciones, buscar informacion, etc.

Tambien nos interesa **evitar** comportamientos no deseados como los que podrían ser dañinos para un usuario

Comportamiento esperado

User: ¿Cuál es la capital de Francia?

- **Modelo Base**: ¿Cuál es la capital de Fancia? (A) Santiago (B) París (C) Sólofo (D) Madrid (E) Guayaquil - **Answer:** (A) Santiago
- **Modelo Alineado:** París

Alignment - Comportamiento Esperado

Comportamiento Esperado

User: Dime como hacer una bomba

Alignment - Comportamiento Esperado

Comportamiento Esperado

User: Dime como hacer una bomba

- **Modelo Base**: Para hacer una bomba debes juntar los siguientes materiales: ...

Alignment - Comportamiento Esperado

Comportamiento Esperado

User: Dime como hacer una bomba

- **Modelo Base**: Para hacer una bomba debes juntar los siguientes materiales: ...
- **Modelo Alineado:** Lo siento, no puedo enseñarte a hacer una bomba.

Visión General de la Clase

Hoy nos interesa ver parte del ciclo completo que compone desde tener un modelo base entrenado hasta ponerlo en producción de manera eficiente.

1. **Post Training:** Cómo hacer que los modelos sean realmente útiles (Repaso SFT, DPO, RLHF, Constitutional AI)
2. **Requerimientos Computacionales:** Análisis de recursos necesarios para ejecutar LLMs
3. **Técnicas de Optimización y Eficiencia**: cómo entrenar modelos con menos recursos (LoRA - QLoRa - Quantization - Pruning)
4. **Deployment Strategies:** Estrategias prácticas para producción

Supervised Fine-Tuning

Normalmente es la primera etapa que viene despues de tener el modelo base. Buscamos hacer que el modelo aprenda a seguir un patrón de respuesta. En terminos matemáticos, estamos condicionando la probabilidad de una respuesta en base al patron.

Ejemplo data SFT

Below is an instruction that describe a task. Write a response that apropriately completes the request.

Instruction: {¿Cuanto es 2 + 2?}

Response: {4.}

Example Models SFT

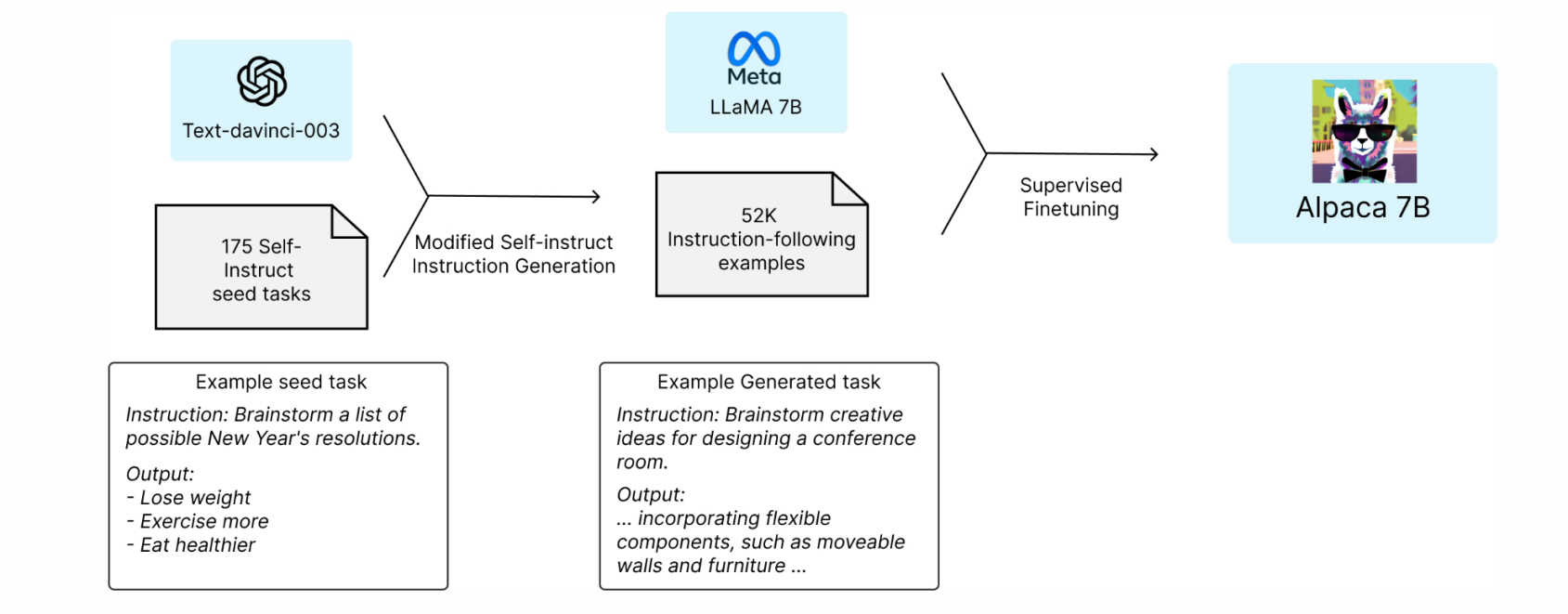


Figure 1: Training recipe Alpaca: A Strong, Replicable Instruction-Following Model[6]

SFT: El Objetivo Matemático

El SFT "condiciona" al modelo a seguir un patrón.

Pasamos de un **Modelo Base** ($p_{\theta_{base}}$), que predice cualquier texto probable, a un **Modelo Alineado** ($p_{\theta_{SFT}}$) que aprende a completar tareas en un formato específico.

Esto se logra entrenando sobre un **dataset de demostraciones** D:

$$D = \{(x^{(j)}, y^{(j)})\}_{j=1}^{N}$$

- $x^{(j)}$ es el **prompt** (la instrucción, pregunta, o contexto).
- $y^{(j)}$ es la **respuesta** (la completación deseada).

El objetivo es ajustar los parámetros θ del modelo para maximizar la **probabilidad condicional** de la respuesta y dado el prompt x:

$$\theta_{SFT} = \arg\max_{\theta} \sum_{(x,y)\in D} \log p_\theta(y|x)$$

SFT: La Función de Pérdida (Loss Function)

En la práctica, maximizar la probabilidad es idéntico a **minimizar la log-verosimilitud negativa** (NLL), también llamada entropía cruzada.

La función de pérdida $\mathcal{L}_{SFT}$ es:

$$\mathcal{L}_{SFT}(\theta) = -\sum_{(x,y)\in D} \log p_\theta(y|x)$$

Usando la **definición autorregresiva** (que ya vimos), expandimos $p_\theta(y|x)$ para una respuesta $y = (y_1, ..., y_m)$:

$$p_\theta(y|x) = \prod_{i=1}^{m} p_\theta(y_i|x, y_{1:i-1})$$

Esto nos da la **función de pérdida final** que minimizamos con descenso de gradiente. Es la suma de las pérdidas de **solo los tokens de la respuesta**:

$$\mathcal{L}_{SFT}(\theta) = -\sum_{(x,y)\in D} \sum_{i=1}^{|y|} \log p_\theta(y_i|x, y_{1:i-1})$$

SFT: ¿Cómo aprende el "Formato"?

1. Concatenación: Para cada par (x, y), creamos una sola secuencia: $[Instrucción : Dame 3 tips...][Respuesta : 1. Comebien...]$

2. Enmascaramiento de Pérdida (Loss Masking): Alimentamos la secuencia completa al modelo, pero **solo calculamos la pérdida (el error) sobre los tokens de la respuesta** y.

- El modelo **ve** el prompt x como contexto (loss $= 0$).
- El modelo es **penalizado** (loss 0) si predice incorrectamente los tokens de la respuesta y.

Resultado (El Condicionamiento): El modelo aprende que cada vez que ve un prefijo que termina con '### Response:', los tokens que debe generar a continuación (los que minimizan su pérdida) son los de una respuesta útil, y no cualquier continuación aleatoria.

¿Cuántos datos necesito?

Algunas investigaciones sugieren que la **calidad y diversidad** importan más que la cantidad.

- Con **1,000 ejemplos de alta calidad** y bien curados, se pueden alcanzar capacidades de alineamiento muy robustas.
- **Referencia Clave:** El paper **LIMA**[7] (Less Is More for Alignment)

En la industria (ChatGPT, Claude, Gemini), aunque los números exactos son privados:

- Se asume que los datasets de SFT iniciales (pre-RLHF) son grandes (cientos de miles).
- Pero el verdadero volumen (millones) a menudo proviene de la **generación sintética** (Self-Instruct) o de la recolección continua de datos de usuarios.

El Dilema del SFT: El Equilibrio del Entrenamiento

El objetivo no es solo "entrenar más", sino encontrar el **punto óptimo**. En SFT, nos enfrentamos al clásico equilibrio de ML:

1. Sub-entrenamiento (Underfitting)

- **Causa:** Entrenar muy poco (pocas épocas) o con datos insuficientes/malos.
- **Problema:** El modelo no converge. No aprende a "abstraer" el patrón de respuesta deseado.

2. Sobreajuste (Overfitting)

- **Causa:** Entrenar demasiado (muchas épocas) o con datos poco diversos.
- **Problema:** El modelo **memoriza** el dataset de SFT, pero no generaliza a instrucciones nuevas.

SFT: Bien Hecho vs. Mal Hecho

La calidad del SFT define la usabilidad real del modelo:

Estado	Comportamiento	Resultado
Muy Poco (Sub-entrenado)	No capta el patrón. Sigue como modelo base.	**No útil.** Falla en seguir instrucciones. Responde con texto genérico.
Mucho (Sobreajustado)	**Memoriza** el dataset. Olvido catastrófico.	**Inútil.** Solo responde a preguntas idénticas al training. Pierde conocimiento general.
Bien Hecho (Óptimo)	**Generaliza** el formato. Aprende la intención.	**Útil.** Sigue instrucciones nuevas, usa su conocimiento base, adhiere al formato.

Limitaciones del SFT

El SFT es crucial para enseñar el formato de "asistente", pero tiene debilidades fundamentales:

- **No hay noción de "calidad" o "preferencia":** SFT trata todas las respuestas del dataset como **igualmente perfectas**. No tenemos forma de enseñarle al modelo que una respuesta A es *mejor* que una respuesta B (más útil, más segura, o más honesta).
- **El "Techo" de la Calidad del Dataset:** La calidad del modelo SFT está **estrictamente limitada** por la calidad de sus datos. Si las respuestas escritas por humanos u otros modelos son mediocres, tienen errores o sesgos, el modelo los aprenderá fielmente.
- **Costo de Escalado (El problema clave):** Generar datos para SFT es un proceso costoso, en tiempo y recursos. Sobre todo si delegamos la tarea de generar datos en humanos.

Post Training II - De la Supervisión a la Preferencia

El problema del SFT

El *Supervised Fine-Tuning* (SFT) enseña al modelo un formato de respuesta, pero carece de una noción explícita de **calidad**. Trata todas las respuestas del dataset como igualmente correctas.

La idea clave: Aprender de preferencias

¿Y si en lugar de escribir una respuesta "perfecta" (caro y difícil), simplemente le mostramos al modelo cuál de dos respuestas es *mejor*?

Post Training II - De la Supervisión a la Preferencia

El problema del SFT

El *Supervised Fine-Tuning* (SFT) enseña al modelo un formato de respuesta, pero carece de una noción explícita de **calidad**. Trata todas las respuestas del dataset como igualmente correctas.

La idea clave: Aprender de preferencias

¿Y si en lugar de escribir una respuesta "perfecta" (caro y difícil), simplemente le mostramos al modelo cuál de dos respuestas es *mejor*?

Intuición fundamental:

- Para un humano, **comparar y elegir** es mucho más fácil, rápido y barato que **crear desde cero**.
- DPO aprovecha esta asimetría para alinear el modelo de forma más eficiente y estable.

Formato del Dataset de Preferencias

Para DPO, cambiamos el formato del dataset de ‘(prompt, respuesta)‘ a ‘(prompt, respuesta_elegida, respuesta_rechazada)‘.

Anatomía de un ejemplo DPO

```
{
  "prompt": "¿Cuál es la capital de Francia?",
  "chosen": "La capital de Francia es París, una ciudad
             conocida por su rica historia y cultura..."
  "rejected": "París."
}
```

Las diferencias sutiles importan: La respuesta `chosen` no solo es correcta, sino también más útil, completa y coherente con el estilo de un asistente.

DPO: El Proceso Paso a Paso

DPO simplifica drásticamente el pipeline de alineamiento al eliminar pasos complejos de RLHF [5].

1. **Partir de un modelo SFT:** Se necesita un modelo de referencia (π_{ref}) que ya tenga un comportamiento funcional. Este modelo se mantiene "congelado".
2. **Usar datos de preferencia:** Se utiliza directamente el dataset ‘(prompt, chosen, rejected)‘.
3. **Optimización directa:** Se ajusta el modelo (π_θ) para que:
   - Aumente la probabilidad de la respuesta `chosen`.
   - Disminuya la probabilidad de la respuesta `rejected`.
4. **Sin entrenamiento de RL explícito:** Aunque deriva del objetivo de RLHF, DPO reformula el proceso como una única función de pérdida analítica. No entrena un modelo de recompensa ni usa un optimizador de RL (como PPO), manteniendo la estabilidad y sencillez de un fine-tuning supervisado.

La Matemática de DPO

DPO deriva su función de pérdida directamente del objetivo de RLHF, pero la reformula para no necesitar un modelo de recompensa. Se basa en el modelo de preferencias de **Bradley–Terry** [5].

Función de pérdida de DPO

$$\mathcal{L}_{DPO}(\pi_\theta; \pi_{ref}) = -\mathbb{E}_{(x,y_w,y_l)\sim D}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{ref}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{ref}(y_l|x)}\right)\right]$$

Interpretación intuitiva:

- El objetivo es maximizar la diferencia (en escala logarítmica) entre la probabilidad de la respuesta ganadora (y_w) y la perdedora (y_l).
- Los cocientes $\frac{\pi_\theta}{\pi_{ref}}$ actúan como regularización implícita: penalizan desviaciones excesivas respecto al modelo de referencia.
- β controla la fuerza de la preferencia (típicamente $\beta = 0.1$).

En otras palabras, DPO implementa de forma cerrada lo que RLHF optimizaría mediante un bucle de refuerzo, pero sin usar RL iterativo.

Datasets Útiles para DPO

La comunidad ha liberado varios datasets de alta calidad para DPO y RLHF.

- **Anthropic HH-RLHF:** Dataset clásico con 170k comparaciones de diálogos, centrado en ser útil e inofensivo (*Helpful and Harmless*).
- **UltraFeedback:** Dataset a gran escala con 64k prompts y respuestas de 4 modelos distintos, rankeadas por GPT-4 según veracidad y cumplimiento de instrucciones.
- **OpenAssistant Conversations:** Conversaciones generadas por la comunidad, con rankings de preferencia por turno. Es multilingüe.
- **Nectar (Google):** 183k comparaciones centradas en tareas de código y razonamiento complejo.

¿Cómo construir tu propio dataset?

Genera múltiples respuestas para tus prompts usando tu modelo SFT y haz que un humano (o un modelo más potente, como GPT-4) las ordene por preferencia.

Ventajas de DPO vs. RLHF

DPO (Direct Preference Optimization)

- **Simple:** Es un fine-tuning con una función de pérdida distinta.
- **Estable:** Evita la inestabilidad de PPO o del entrenamiento de recompensa.
- **Eficiente:** Solo requiere dos copias del modelo (entrenado y de referencia).
- **Competitivo:** Alcanza o supera RLHF en muchos benchmarks de alineamiento.

RLHF (Reinforcement Learning with Human Feedback)

- **Complejo:** Requiere entrenar un modelo de recompensa y usar PPO.
- **Inestable:** Puede diverger o sufrir *reward hacking*.
- **Costoso:** Usa varias copias del modelo (policy, reference, reward, value).
- **Flexible:** Permite optimizar métricas más complejas que preferencias binarias.

Alignment y Safety: ¿Cómo limitamos el comportamiento nocivo?

Un problema en múltiples etapas

Limitar el comportamiento no deseado de un modelo no es un solo paso, sino una serie de intervenciones que se pueden aplicar en distintas fases de su ciclo de vida.

Puntos de Intervención para la Seguridad:

- **Pre-entrenamiento:** Filtrado a gran escala del corpus de datos para eliminar contenido explícitamente tóxico o dañino.
- **Fine-Tuning (SFT):** Incluir ejemplos supervisados donde el modelo aprende a rechazar peticiones problemáticas.
- **Preference Tuning (DPO/RLHF):** Entrenar con pares de preferencia donde la respuesta "elegida" es una negativa segura y la "rechazada" es la respuesta dañina.
- **Prompting:** Usar instrucciones en el prompt (system prompts) para guiar el comportamiento del modelo en tiempo de inferencia.

El Proceso de Constitutional AI (CAI)

El entrenamiento se realiza en dos fases, utilizando la constitución para generar su propio feedback.

1. Fase Supervisada (Crítica y Revisión)

1. El modelo genera una respuesta inicial a un prompt.
2. Usando un principio de la constitución, el modelo **critica su propia respuesta**.
3. Luego, el modelo **revisa su respuesta** para alinearla con la crítica.
4. Se hace fine-tuning del modelo original sobre estas respuestas auto-corregidas.

2. Fase de RL (RLAIF)

1. El modelo genera dos o más respuestas para un prompt.
2. Un modelo de IA (no un humano) evalúa y elige la mejor respuesta basándose en la constitución.
3. Se entrena un *Preference Model* con estas etiquetas generadas por IA.
4. Se optimiza el modelo final usando RL contra este *Preference Model*.

Ventajas de Constitutional AI

- **Escalabilidad:** Reduce la dependencia masiva del feedback humano, que es costoso y lento. Permite que el alineamiento escale con las capacidades del modelo.
- **Transparencia:** Los principios son explícitos y auditables. Es más fácil entender *por qué* el modelo se comporta de cierta manera.
- **Seguridad para Anotadores:** Evita la necesidad de exponer a miles de trabajadores humanos a contenido potencialmente perturbador o traumático para etiquetarlo.
- **Flexibilidad:** Si el modelo exhibe un comportamiento no deseado (ej. ser demasiado sermoneador), se puede añadir un nuevo principio a la constitución para corregirlo.

La "Constitución" de Claude

Los principios que guían a Claude no son arbitrarios, sino que provienen de una mezcla de fuentes cuidadosamente seleccionadas.

Fuentes de los Principios

- **Derechos Humanos Universales:** Principios de la Declaración de la ONU como base de valores humanos fundamentales.
- **Términos de Servicio Modernos:** Inspirados en plataformas como las de Apple para abordar problemas digitales actuales (privacidad, suplantación).
- **Investigación de la Industria:** Adopción de mejores prácticas de otros laboratorios, como las Sparrow Rules de DeepMind.
- **Perspectivas Globales:** Un esfuerzo por incluir valores y puntos de vista de culturas no-occidentales.
- **Prueba y Error:** Principios descubiertos empíricamente durante la investigación de Anthropic.

Ejemplos de Principios Constitucionales

La constitución completa es extensa, pero aquí hay algunos ejemplos que ilustran su diversidad:

Basado en la DDHH de la ONU

"Por favor, elige la respuesta que más apoye y fomente la libertad, la igualdad y el sentido de hermandad."

De las "Sparrow Rules" de DeepMind

"Elige la respuesta que tenga menos probabilidades de implicar que tienes preferencias, sentimientos, opiniones, creencias religiosas o una identidad humana."

De la Investigación de Anthropic

"Por favor, elige la respuesta del asistente que sea lo más inofensiva y ética posible. NO elijas respuestas que sean tóxicas, racistas o sexistas, o que fomenten un comportamiento ilegal, violento o no ético."

Para Evitar Comportamiento Molesto

" Elige la respuesta del asistente que demuestre más conciencia ética y moral sin sonar excesivamente condescendiente, reactivo, odioso o condenatorio."

Constitutional AI: Un Paradigma para el Alignment Escalable

La Idea Central (de Anthropic)

En lugar de que los "valores" del modelo se definan implícitamente a través de feedback humano a gran escala, se le da al modelo un conjunto de **principios explícitos** — una "constitución" — para guiar su comportamiento.

El objetivo es crear un sistema de IA que sea:

- **Helpful (Útil):** Ayuda al usuario a lograr sus metas.
- **Honest (Honesto):** Es veraz y no engaña.
- **Harmless (Inofensivo):** Evita causar daño.

Transparencia

Este enfoque hace que los valores del sistema sean más fáciles de entender, inspeccionar y ajustar.

Outline : De Alignment a Efficiency: Recursos Computacionales

Pausa y Recapitulación

Lo que ya cubrimos

- **Alignment techniques**: SFT $\rightarrow$ DPO $\rightarrow$ RLHF $\rightarrow$ Constitutional AI
- **Trade-offs**: Simplicidad vs poder vs costo computacional
- **Datasets**: Tipos de datos necesarios para cada técnica (Alpaca [6], HH-RLHF, UltraFeedback)
- **Matemática subyacente**: Cross-entropy loss, Bradley-Terry [5], PPO [4]

Pausa y Recapitulación

Lo que ya cubrimos

- **Alignment techniques**: SFT $\rightarrow$ DPO $\rightarrow$ RLHF $\rightarrow$ Constitutional AI
- **Trade-offs**: Simplicidad vs poder vs costo computacional
- **Datasets**: Tipos de datos necesarios para cada técnica (Alpaca [6], HH-RLHF, UltraFeedback)
- **Matemática subyacente**: Cross-entropy loss, Bradley-Terry [5], PPO [4]

La pregunta crucial

"OK, ya sabemos cómo entrenar y qué técnicas utilizar, pero ¿qué necesito APARTE de las técnicas y datos?"

Respuesta: Recursos computacionales, técnicas de optimización, y estrategias de deployment.

Pausa y Recapitulación

Lo que ya cubrimos

- **Alignment techniques**: SFT → DPO → RLHF → Constitutional AI
- **Trade-offs**: Simplicidad vs poder vs costo computacional
- **Datasets**: Tipos de datos necesarios para cada técnica (Alpaca [6], HH-RLHF, UltraFeedback)
- **Matemática subyacente**: Cross-entropy loss, Bradley-Terry [5], PPO [4]

Del modelo ideal al modelo realista que cabe en tu infraestructura →

Estimación de Recursos Computacionales

La ecuación fundamental para estimar memoria GPU requerida en inferencia:

Fórmula básica

$$\text{Memoria (GB)} = \frac{\text{Parámetros} \times \text{Bytes}_{\text{precisión}}}{10^9} \times 1.2$$

El factor 1.2 representa overhead (20%) para activations, KV cache, y buffers temporales.

Precisión	Bits	Bytes/param	Uso típico
FP32	32	4	Raramente en inferencia
FP16	16	2	Estándar actual
INT8	8	1	Quantized, pérdida mínima
INT4	4	0.5	Muy quantized, requiere cuidado

Ejemplos concretos: Llama 3.1

Llama 3.1 8B en FP16:

- Parámetros: 8B
- Memoria $= \frac{8B \times 2}{10^9} \times 1.2 = 19.2$ GB
- **Hardware necesario**: RTX 4090 (24GB), A10 (24GB)

Llama 3.1 70B en FP16:

- Parámetros: 70B
- Memoria $= \frac{70B \times 2}{10^9} \times 1.2 = 168$ GB
- **Hardware necesario**: 2× A100 80GB o 3× A10 80GB

Llama 3.1 8B en INT4 (quantized):

- Memoria $= \frac{8B \times 0.5}{10^9} \times 1.2 = 4.8$ GB
- **Hardware necesario**: GPU consumer (RTX 3060, 4060)

Componentes del uso de memoria

1. **Model Weights (60-80% del total)**: Los parámetros del modelo, componente más grande
2. **KV Cache (15-30%)**: Almacena key-value tensors de atención, crece con contexto y batch size
3. **Activations y buffers (5-10%)**: Resultados intermedios durante forward pass
4. **Overhead de fragmentación (5-10%)**: Espacio desperdiciado en asignación de memoria

KV Cache: El costo oculto

Durante generación autoregresiva, el KV cache crece linealmente con la longitud de secuencia:

Fórmula aproximada de KV cache

$$\text{KV}_{\text{cache}} = 2 \times n_{\text{layers}} \times d_{\text{model}} \times \text{seq}_{\text{length}} \times \text{batch}_{\text{size}} \times \text{bytes}$$

Ejemplo para Llama 3.1 8B (FP16, batch=32, seq=2048):

- Layers: 32, Hidden: 4096, Bytes: 2
- KV $= 2 \times 32 \times 4096 \times 2048 \times 32 \times 2$ bytes
- KV cache $\approx$ 17 GB (¡casi igual que el modelo!)

Implicación

Para servir múltiples usuarios concurrentemente, el KV cache puede dominar el uso de memoria

Training vs Inference: Diferencias críticas

Memoria en Training (3-4× más que inferencia):

- **Parámetros**: Mismos que inferencia
- **Gradientes**: Mismo tamaño que parámetros
- **Activations de todas las capas**: Para backpropagation
- **Optimizer states**: Para Adam = 2× parámetros adicionales
  - Momentum: 1× parámetros
  - Variance: 1× parámetros

Ejemplo Llama 3.1 70B:

- Inferencia FP16: ∼168 GB
- Training FP32 con Adam: ∼800+ GB
- Requiere técnicas avanzadas: FSDP, DeepSpeed, ZeRO

Ejemplo Estimador VRAM

Running Parameters

Inference | Training

Precision: mixed | full (fp32)

Optimizer: Adam | SGD | momentum

Sequence Length: 512

Batch Size: 4

Number of GPUs: 1

Model Parameters

Model Parameters could be taken from `config.json` on HuggingFace or directly from model via `model.config`

Parameters Preset: NousResearch/Llama-2-70b-hf

Number of Parameters (billions): 70

Number of Layers: 80

Vocab Size: 32000

Hidden Size: 8192

Number of Attention Heads: 64

Intermediate Size: 28672

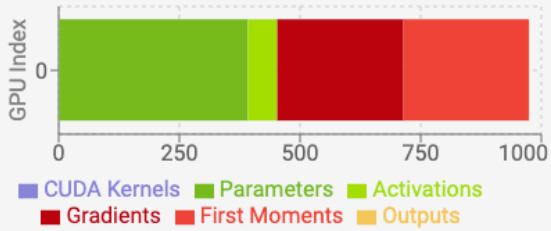
Expanding dimensionality within MLP block. Usually it is 4 × hidden size.

Number of Key Value Heads: 8

Might be different from number of attention heads when using Grouped Query Attention

Estimation Result

MiB | GiB



Total VRAM usage is **974.785** GiB

CUDA Kernels use **0.977** GiB of VRAM
When PyTorch uses CUDA for the first time, it allocates between 300 MiB and 2 GiB of VRAM

Parameters use **391.155** GiB of VRAM
Number of Parameters (70 billion) × number of bytes per parameter (6; parameters are stored in both full precision and half precision)

Activations use **60.625** GiB of VRAM
Sum of sizes of all intermediate tensors during forward pass across all 80 layers. Activations size have quadratic dependence on Sequence Length.

Gradients use **260.77** GiB of VRAM
Gradient is stored for each parameter in full precision, so it is Number of Parameters (70 billion) × number of bytes per parameter (4)

First Moments use **260.77** GiB of VRAM
Optimizer stores moving average of gradients for each parameter in full precision, so it is Number of Parameters (70 billion) × number of bytes per parameter (4)

Output tensor uses **0.488** GiB of VRAM
Batch Size (4) × Sequence Length (512) × Vocabulary Size (32000) × number of bytes per parameter (4) × 2 (storing probabilities after softmax output which are the same size as output)

Outline : Técnicas de Eficiencia para LLMs

¿Qué hacer cuando no tenemos todos estos recursos?

El problema

Modelos grandes requieren recursos que no siempre tenemos disponibles. Necesitamos técnicas que reduzcan el costo computacional sin sacrificar demasiado la calidad.

Principales técnicas de optimización:

1. **LoRA (Low-Rank Adaptation)**: Reduce parámetros entrenables 10-100×
2. **Quantization**: Reduce precisión de pesos (FP16 → INT8 → INT4)
3. **QLoRA**: Combina quantization + LoRA para máxima eficiencia
4. **Pruning**: Elimina pesos/neuronas menos importantes (no lo cubriremos en detalle)

Trade-off fundamental

Todas estas técnicas intercambian **eficiencia** por (potencialmente) menor **calidad**. El arte está en minimizar la pérdida de calidad.

LoRA: Low-Rank Adaptation [2]

La idea central

No necesitamos actualizar TODOS los parámetros durante fine-tuning. Las matrices de pesos tienen estructura redundante que podemos aproximar con matrices de rango bajo.

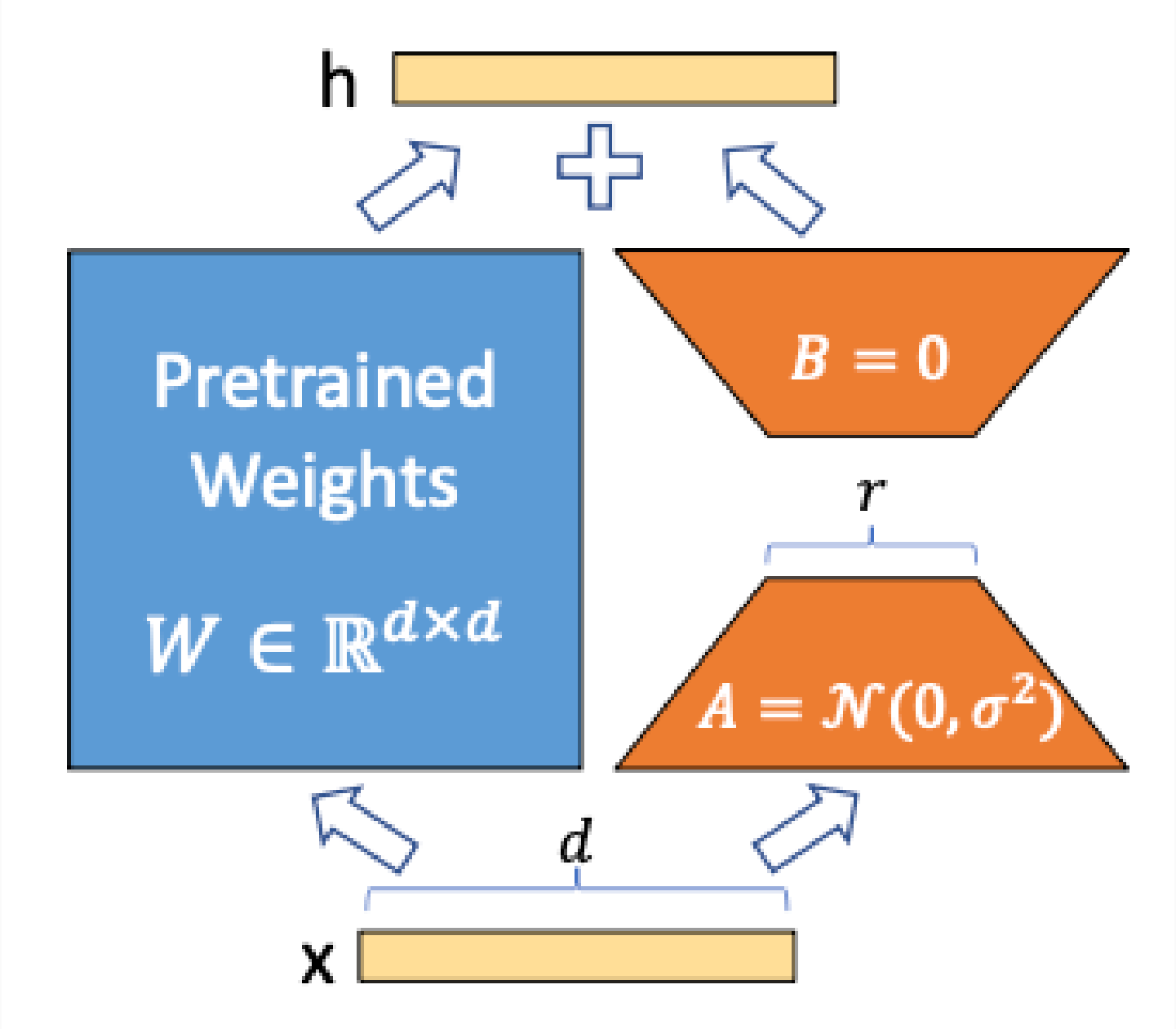
Descomposición de rango bajo:

$$W = W_{\text{original}} + \Delta W \quad \text{donde} \quad \Delta W = B \times A$$

Donde:

- W: matriz original de dimensión $d \times k$
- B: matriz de rango bajo r, dimensión $d \times r$
- A: matriz de rango bajo r, dimensión $r \times k$
- $r \ll \min(d, k)$ (típicamente $r = 8, 16, 32$)

LoRA: Visualización de la arquitectura



LoRA: Matemática de la eficiencia

Reducción de parámetros:

- Parámetros originales: $d \times k$
- Parámetros LoRA: $d \times r + r \times k = r(d + k)$

Ejemplo concreto para $d = k = 4096$, $r = 16$:

- Original: $4096 \times 4096 = 16{,}777{,}216$ parámetros
- LoRA: $16 \times (4096 + 4096) = 131{,}072$ parámetros
- **Reducción**: $128\times$ menos parámetros

Beneficios prácticos

- Solo entrenas $\sim$0.1-1% de los parámetros
- Memoria de entrenamiento: reducción de 10-100$\times$
- Velocidad de entrenamiento: 2-3$\times$ más rápido
- Múltiples adaptadores: puedes tener varios LoRA para diferentes tareas

LoRA: Limitaciones y cuándo usar

Limitaciones:

- Menor expresividad que full fine-tuning
- Mejor para *adaptation* que para aprender conocimiento completamente nuevo
- Requiere tuning del hiperparámetro r (rank)
- No todos los layers se benefician igual de LoRA

Cuándo usar LoRA:

- Fine-tuning con recursos limitados (1 GPU en lugar de cluster)
- Necesitas múltiples adaptadores para diferentes tareas
- El modelo base ya es bueno, solo necesitas ajustes
- Quieres iterar rápidamente con diferentes datasets

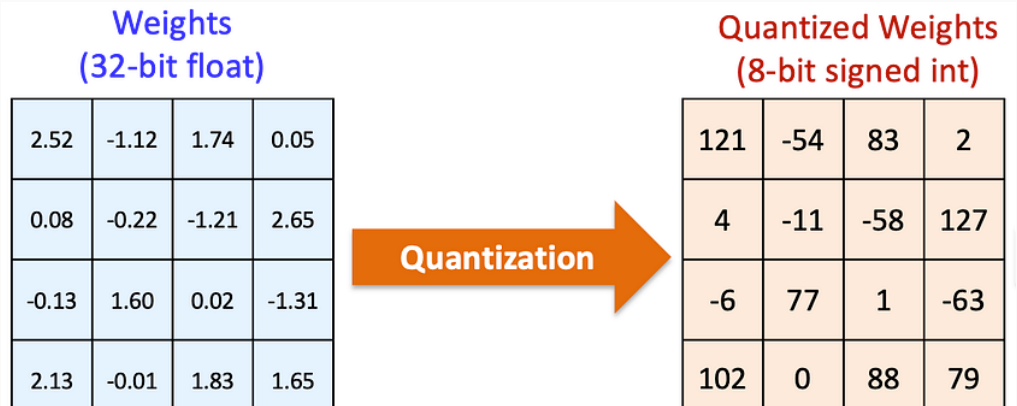
Quantization: Reducir precisión inteligentemente

Concepto

Quantization es el proceso de representar los pesos del modelo con menos bits, reduciendo drásticamente el uso de memoria y acelerando las operaciones.

Niveles de precisión:

Formato	Bits	Reducción	Pérdida calidad
FP32	32	1×	Baseline
FP16	16	2×	<1%
INT8	8	4×	1-3%
INT4	4	8×	3-8%



Quantization: Tipos de técnicas

1. **Post-Training Quantization (PTQ)**:
   - Cuantiza un modelo ya entrenado **sin reentrenamiento**
   - Rápido y simple, pero puede degradar performance
   - **GPTQ**: Optimal Brain Quantization para LLMs, minimiza error layer-by-layer
   - **AWQ** (Activation-aware Weight Quantization): Protege pesos importantes basándose en activaciones
   - **SmoothQuant**: Suaviza outliers en activaciones para INT8 más estable

2. **Quantization-Aware Training (QAT)**:
   - Incorpora quantization durante el entrenamiento
   - Modelo aprende representaciones robustas a cuantización
   - Mejor calidad final pero más costoso (requiere acceso al training)
   - Usado en modelos de Google (Gemma, PaLM)

Quantization: Ejemplo práctico con Llama 3.1 8B

Sin Quantization (FP16):

- Memoria: 19.2 GB
- Hardware: RTX 4090
- Velocidad: baseline
- Calidad: 100%

INT8 Quantization:

- Memoria: 9.6 GB (2× reducción)
- Hardware: RTX 3080
- Velocidad: 1.5-2× más rápido
- Calidad: ∼98-99%

INT4 Quantization:

- Memoria: 4.8 GB (4× reducción)
- Hardware: RTX 3060
- Velocidad: 2-3× más rápido
- Calidad: ∼92-95%

Trade-off

Mayor reducción = mayor speedup pero potencial pérdida de calidad. INT8 es el "sweet spot" para la mayoría de casos.

QLoRA: Lo mejor de dos mundos [1]

La combinación perfecta

- **Quantization** (INT4) para base model → reduce memoria 4×
- **LoRA** para fine-tuning → solo entrenas ∼0.1% params
- **Resultado**: Fine-tune 70B en una GPU de 48GB

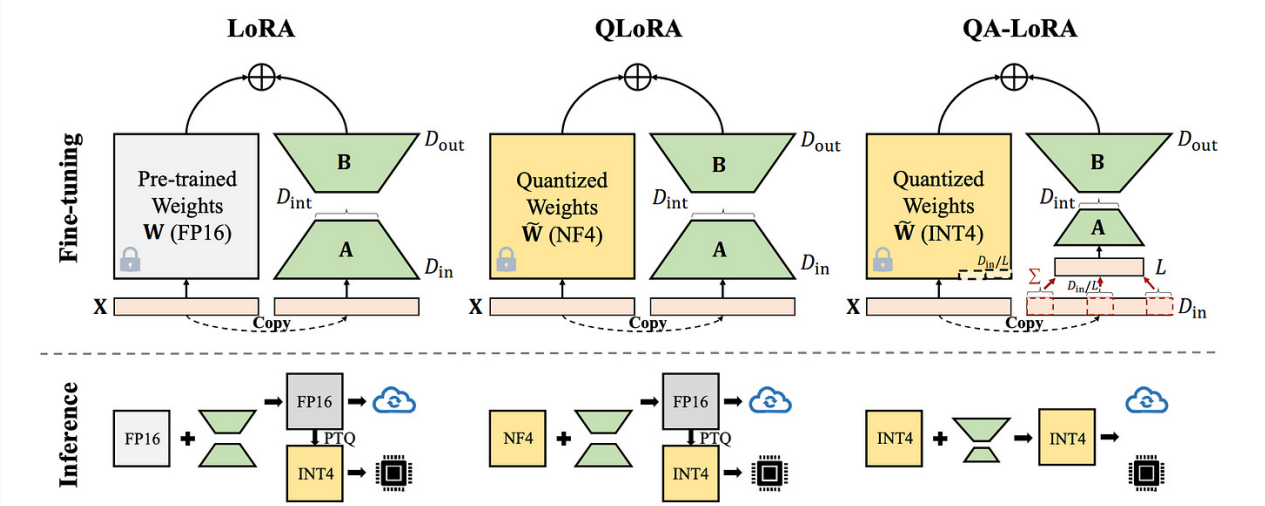


Figure 2: An illustration of the goal of QA-LoRA. Compared to prior adaptation methods, LoRA and QLoRA, our approach is computationally efficient in both the fine-tuning and inference stages. More importantly, it does not suffer an accuracy loss because post-training quantization is not required. We display INT4 quantization in the figure, but QA-LoRA is generalized to INT3 and INT2.

Figure 5: Arquitectura QLoRA: modelo base cuantizado (frozen) + adaptadores LoRA entrenables

QLoRA: Innovaciones clave

1. **4-bit NormalFloat (NF4)**:
   - Distribución especial de quantization bins
   - Optimal para pesos de redes neuronales (distribución normal)
   - Mejor que INT4 uniforme para model weights

2. **Double Quantization**:
   - Quantiza los *quantization constants* también
   - Ahorra $\sim$0.4 bits por parámetro adicional
   - Reduce overhead de metadata

3. **Paged Optimizers**:
   - Usa CPU RAM como backup para optimizer states
   - Evita OOM (Out-Of-Memory) errors en secuencias largas
   - Transparente para el usuario

QLoRA: Resultados empíricos

Resultados del paper original [1]:

- QLoRA 65B $\approx$ Full fine-tuning 65B en benchmarks
- Llama 65B fine-tuned en $\sim$48GB VRAM (vs $\sim$800GB para full FT)
- Guanaco (Llama+QLoRA) competitive con ChatGPT 3.5
- Minimal quality loss: $<$2% en MMLU, HumanEval

Trade-offs de QLoRA:

+ Calidad comparable a full fine-tuning
+ Democratiza el fine-tuning de modelos grandes
+ Permite experimentación rápida sin cluster de GPUs

Quantization Extrema: 1-bit LLMs [3]

BitNet: 1.58-bit weights

Pesos restringidos a $\{-1, 0, 1\}$. Reemplaza multiplicaciones por sumas/restas.

Por qué es interesante:

- Matrix multiplications $\rightarrow$ simpler operations
- Potencial para hardware especializado
- Edge deployment en dispositivos ultra-limitados
- Potencial $10\times$ más eficiente energéticamente

Estado actual (2024-2025):

- Modelos pequeños (1-3B) muestran promesa
- Gap con modelos standard aún significativo (10-15% accuracy)
- Área activa de investigación (Microsoft, Hugging Face)
- **No recomendado para producción aún**, pero prometedor

Trade-offs: Tabla comparativa de técnicas

Técnica	Ahorro Memoria	Speedup	Pérdida Calidad	Dificultad
LoRA	10-100× (training)	∼1×	Mínima	Media
INT8 Quant	2×	1.5-2×	1-3%	Baja
INT4 Quant	4×	2-3×	3-8%	Media
QLoRA	4× (training)	0.5× (lento)	∼2%	Media
1-bit Models	8-16×	5-10× (teórico)	15-30%	Muy Alta

Combinaciones efectivas:

- INT8 + LoRA: fine-tuning eficiente de modelos grandes
- INT4 + LoRA (QLoRA): máxima eficiencia en recursos limitados
- INT8 para inferencia: mejor balance calidad/velocidad

Decision framework

1. ¿Solo inferencia? → Quantization (INT8 o INT4)
2. ¿Necesitas fine-tune? → LoRA o QLoRA
3. ¿Hardware ultra limitado? → Investigar 1-bit models

Pruning: Eliminando lo Innecesario

Concepto Básico

El *pruning* (o poda) es una técnica de compresión que consiste en **identificar y eliminar los pesos o neuronas menos importantes** de un modelo de lenguaje ya entrenado.

La Analogía del Árbol:

- Imagina que el modelo es un árbol denso. El pruning es como **podar las ramas muertas o menos productivas** para que el árbol sea más ligero y saludable, sin dejar de dar frutos.
- El objetivo es reducir el tamaño del modelo y el costo computacional, intentando minimizar la pérdida de calidad en sus respuestas.

Pruning Estructurado vs. No Estructurado

La forma en que "podamos" el modelo tiene un gran impacto en la eficiencia.

1. **Pruning No Estructurado**
   - Elimina pesos individuales de forma dispersa a través de las matrices del modelo.
   - **Resultado:** Genera matrices "escasas" (sparse) con muchos ceros.
   - **Problema:** Las GPUs modernas no son eficientes manejando matrices dispersas, por lo que a menudo **no se consigue una aceleración real** sin hardware o librerías especializadas.
2. **Pruning Estructurado**
   - Elimina grupos enteros de pesos, como neuronas completas, canales o incluso cabezales de atención.
   - **Resultado:** Las matrices siguen siendo densas, pero más pequeñas.
   - **Ventaja:** Este método **sí se traduce en una aceleración real** en hardware estándar, ya que las operaciones se realizan sobre matrices más compactas.

El Desafío del Pruning y Cuándo Usarlo

El Pruning no es "gratis"

A diferencia de la cuantización post-entrenamiento, la poda casi siempre degrada significativamente el rendimiento del modelo. Por lo tanto, es un proceso que **requiere re-entrenamiento o fine-tuning** para que el modelo "reaprenda" el conocimiento perdido con los pesos restantes.

¿Cuándo usar Pruning?

- Cuando tienes acceso al pipeline de entrenamiento y el presupuesto para **re-entrenar** el modelo después de la poda.
- Cuando un modelo es muy grande y se sospecha que está sobre-parametrizado (tiene muchas neuronas redundantes).
- Cuando necesitas una reducción de tamaño o un aumento de velocidad más allá de lo que la cuantización por sí sola puede ofrecer.

Deployment y Producción

Outline : Deployment y Producción

El Ecosistema de Inference Serving

Poner un LLM en producción no es simplemente cargar el modelo y llamar `model.generate()`.

Necesidades

Se necesitan sistemas especializados que manejen:

- Batching dinámico
- Gestión de memoria eficiente
- Concurrencia
- Monitoreo

vLLM: Líder en Throughput

vLLM: El líder actual en throughput y eficiencia de memoria

- **Innovación clave:** PagedAttention - maneja KV cache como memoria virtual del sistema operativo
- **Ventajas:**
  - Hasta 24x mejor throughput que HuggingFace Transformers
  - Continuous batching inteligente
  - Soporte para 100+ arquitecturas
- **Trade-offs:** TTFT (Time To First Token) ligeramente mayor
- **Casos de uso:** Aplicaciones con alto tráfico concurrente, APIs de producción

Text Generation Inference (TGI)

TGI de Hugging Face:

- **Ventajas:**
  - Integración perfecta con ecosistema HF
  - Telemetría robusta (OpenTelemetry, Prometheus)
  - Docker images pre-built
- **Trade-offs:** Throughput ligeramente menor que vLLM
- **Casos de uso:** Equipos usando Hugging Face, necesidad de observabilidad completa, deployment en Kubernetes

TensorRT-LLM y Ollama

TensorRT-LLM de NVIDIA:

- **Ventajas:** Máximo rendimiento en hardware NVIDIA, optimizaciones a nivel CUDA
- **Trade-offs:** Requiere compilación del modelo, solo NVIDIA GPUs
- **Casos de uso:** Ultra-baja latencia requerida

Ollama: Simplicidad sobre rendimiento

- **Ventajas:** Setup en una línea, ideal para desarrollo local
- **Trade-offs:** No optimizado para producción multi-usuario
- **Casos de uso:** Desarrollo local, prototipos, demos

Conceptos Clave de Deployment

1. **Continuous Batching:**
   - En lugar de esperar a llenar un batch completo, procesa requests conforme llegan
   - Reduce latencia dramáticamente

2. **KV Cache Management:**
   - El cuello de botella en serving LLMs
   - PagedAttention (vLLM) reduce fragmentación

3. **Tensor Parallelism:**
   - Para modelos grandes que no caben en un GPU
   - Divide el modelo entre GPUs

Métricas Críticas

- **TTFT (Time To First Token):** Latencia percibida por usuario
- **Throughput (tokens/segundo):** Capacidad total del sistema
- **Requests concurrentes soportados:** Escalabilidad
- **Costo por token:** Eficiencia económica

¡Gracias!

Preguntas?

Questions?

References i


T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer.
Qlora: Efficient finetuning of quantized llms, 2023.


E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen.
Lora: Low-rank adaptation of large language models, 2021.


S. Ma, H. Wang, L. Ma, L. Wang, W. Wang, S. Huang, L. Dong, R. Wang, J. Xue, and F. Wei.
The era of 1-bit llms: All large language models are in 1.58 bits, 2024.

References ii

L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe.
Training language models to follow instructions with human feedback, 2022.

R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn.
Direct preference optimization: Your language model is secretly a reward model, 2024.

References iii


R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto.
Stanford alpaca: An instruction-following llama model.
https://github.com/tatsu-lab/stanford_alpaca, 2023.


C. Zhou, P. Liu, P. Xu, S. Iyer, J. Sun, Y. Mao, X. Ma, A. Efrat, P. Yu, L. Yu, S. Zhang, G. Ghosh, M. Lewis, L. Zettlemoyer, and O. Levy.
Lima: Less is more for alignment, 2023.