La Guía Completa para la Implementación de Modelos de Lenguaje Grandes en Local: Desde el Hardware hasta las Aplicaciones Empresariales de Alto Valor

Resumen Ejecutivo

Este informe ofrece un análisis exhaustivo de los requisitos técnicos, el ecosistema de software y las estrategias de implementación para ejecutar Modelos de Lenguaje Grandes (LLM) en infraestructura local. La ejecución de LLMs en local ha pasado de ser una actividad de nicho a una opción estratégica viable para empresas que buscan garantizar la privacidad de los datos, controlar los costos y obtener un nivel de personalización inalcanzable con las API de terceros.

El análisis revela que la Memoria de Acceso Aleatorio de Vídeo (VRAM) de la Unidad de Procesamiento Gráfico (GPU) es la principal restricción de hardware que determina la viabilidad de ejecutar un modelo determinado. La capacidad de la VRAM dicta el tamaño y la complejidad del LLM que un sistema puede manejar eficientemente. En este contexto, la cuantización —la técnica de reducir la precisión numérica de los pesos de un modelo— emerge como la tecnología habilitadora clave. Permite que modelos masivos, que de otro modo requerirían hardware de centro de datos, se ejecuten en estaciones de trabajo y servidores accesibles, reduciendo drásticamente la barrera de entrada.

El ecosistema de software para LLMs locales se está bifurcando claramente en dos corrientes principales. Por un lado, herramientas fáciles de usar como Ollama y LM Studio están democratizando el acceso, permitiendo a desarrolladores y empresas experimentar rápidamente con una amplia gama de modelos de código abierto. Por otro lado, motores de inferencia de alto rendimiento como vLLM están diseñados para entornos de producción, ofreciendo la escalabilidad y el rendimiento necesarios para aplicaciones multiusuario de nivel empresarial.

Finalmente, para extraer un valor comercial tangible, las organizaciones deben ir más allá de la simple ejecución de modelos preentrenados. La decisión estratégica fundamental radica en elegir entre dos técnicas de personalización principales: la Generación Aumentada por Recuperación (RAG), que proporciona a los modelos acceso en tiempo real a bases de

conocimiento propietarias, y el ajuste fino (fine-tuning), que inculca una profunda experiencia de dominio en el propio modelo. Este informe concluye que la solución más poderosa para la mayoría de los casos de uso empresarial es un enfoque híbrido, que combina la experiencia especializada del ajuste fino con la relevancia contextual y actualizada de RAG para lograr el máximo impacto.

Parte I: La Base: Arquitectura de su Infraestructura de IA Local

Esta sección establece el conocimiento técnico fundamental necesario para construir o adquirir el hardware adecuado. Va más allá de las simples recomendaciones para explicar el *porqué* detrás de los requisitos, capacitando al lector para tomar decisiones informadas y preparadas para el futuro.

1.1 El Papel Central de la VRAM: El Principal Cuello de Botella

A diferencia de la computación tradicional, donde la CPU y la RAM del sistema son los componentes principales, los LLMs están fundamentalmente limitados por la memoria, específicamente por la Memoria de Acceso Aleatorio de Vídeo (VRAM) de la GPU. Para un cálculo paralelo eficiente, que es la principal fortaleza de una GPU, el modelo completo, o al menos porciones significativas del mismo, debe caber en la VRAM.[1] Aunque herramientas como LM Studio permiten descargar parte del modelo a la RAM del sistema, esto conlleva una penalización catastrófica en el rendimiento, haciéndolo inadecuado para un trabajo serio y profesional.[2] La dependencia de la VRAM ha impulsado la innovación en el software, llevando al desarrollo y la adopción de técnicas de optimización, principalmente la cuantización, como una forma de sortear esta barrera física del hardware.
El consumo de VRAM durante la operación de un LLM se descompone en varios componentes clave:

- **Pesos del Modelo:** Este es el componente más grande y estático. El número de parámetros del modelo (por ejemplo, 7 mil millones, 70 mil millones) determina directamente este uso de memoria. Cada parámetro es un valor numérico que debe almacenarse, y el tamaño total es el producto del número de parámetros y el tamaño de cada parámetro en bytes.[3]
- **Caché KV (Key-Value Cache):** Un componente crítico para la inferencia generativa. Su tamaño es dinámico y crece con la longitud del contexto (longitud de la secuencia) y el tamaño del lote (batch size). A medida que el modelo genera texto, almacena estados intermedios (claves y valores de atención) en esta caché para evitar recalcularlos para tokens anteriores, lo que acelera drásticamente la generación. Este es un factor crucial para aplicaciones que requieren conversaciones largas o el análisis de documentos extensos.[1]

- **Activaciones:** Son datos temporales generados durante la pasada hacia adelante (forward pass) del modelo. Este consumo de memoria es particularmente intensivo durante el entrenamiento y el ajuste fino, pero sigue siendo relevante para la inferencia, especialmente con tamaños de lote grandes.[1]
- **Sobrecarga del Framework:** Es la memoria consumida por los frameworks de software subyacentes como PyTorch, TensorFlow y los kernels de CUDA. A menudo, esto representa una sobrecarga fija de 1 a 2 GB, independientemente del modelo.[1]
- **Estados del Optimizador y Gradientes (Solo para Entrenamiento/Ajuste Fino):** Para el ajuste fino, se necesita VRAM adicional para almacenar los estados del optimizador (por ejemplo, AdamW) y los gradientes calculados durante la retropropagación. Estos componentes pueden requerir varias veces el tamaño del propio modelo, lo que explica por qué el ajuste fino completo es mucho más exigente en VRAM que la inferencia.[3]

1.2 Cálculo de sus Necesidades de VRAM: De la Teoría a la Práctica

Estimar con precisión los requisitos de VRAM es el primer paso para seleccionar el hardware adecuado. Se pueden utilizar varias fórmulas y reglas generales para guiar este proceso.

- **Fórmulas y Reglas Generales:**
  - **Fórmula Básica de Inferencia:** Una fórmula simple y potente para una estimación rápida es: $\text{VRAM (GB)} \approx \text{Parámetros del Modelo (miles de millones)} \times \text{Bytes por Parámetro}$. Esta relación lineal directa entre el tamaño del modelo y las necesidades de memoria es fundamental para la planificación.[1] Los bytes por parámetro dependen de la precisión numérica utilizada:
    - **FP32 (32 bits):** 4 bytes por parámetro. Alta precisión, pero alto consumo de VRAM.
    - **FP16/BF16 (16 bits):** 2 bytes por parámetro. El estándar de facto para la mayoría de los modelos, equilibrando precisión y uso de memoria.
    - **INT8 (8 bits):** 1 byte por parámetro. Reducción significativa de memoria con una pérdida de precisión a menudo aceptable.
    - **INT4 (4 bits):** ~0.5 bytes por parámetro. Reducción drástica de memoria, ideal para inferencia en hardware de consumo.
  - **La Regla del 20% de Sobrecarga:** Una heurística práctica es añadir un búfer del 20% para la caché KV, las activaciones y la sobrecarga del framework en tareas de inferencia típicas.[6] La fórmula se convierte en:
    $\text{VRAM (GB)} \approx (\text{Parámetros\_B} \times \text{Bytes\_por\_Para´m}) \times 1.2$.
  - **Cálculo Avanzado (Ajuste Fino):** Los requisitos de VRAM para el ajuste fino son significativamente más altos. Además de los pesos del modelo, se deben almacenar los gradientes ($\text{VRAM}_{\text{gradientes}} = \text{Para´metros Entrenables} \times \text{Bytes por Para´metro}$) y los estados del optimizador ($\text{VRAM}_{\text{optimizador}} \approx 2 \times \text{Para´metros Entrenables} \times 4$ bytes para estados FP32).[4] Esto explica por qué técnicas como LoRA (Low-Rank Adaptation) son críticas. LoRA reduce drásticamente el número de parámetros entrenables, lo que puede disminuir las necesidades de memoria

hasta 5.6 veces y hacer que el ajuste fino sea accesible en hardware de consumo.[3]

- **El Poder de la Cuantización:** Esta es la técnica más importante para hacer prácticos los LLMs locales.
    - **¿Qué es la Cuantización?** Es el proceso de reducir la precisión numérica de los pesos del modelo (por ejemplo, de flotantes de 16 bits a enteros de 4 bits) para disminuir drásticamente la huella de VRAM y, a menudo, aumentar la velocidad de inferencia. Es análogo a comprimir un archivo de audio WAV a MP3; se pierde algo de información, pero el archivo se vuelve mucho más pequeño y manejable.[1]
    - **El Compromiso entre Precisión y Rendimiento:** Existe un compromiso inherente. Mientras que la cuantización a INT8 a menudo logra un rendimiento similar a FP16, una cuantización más agresiva como INT4 o inferior puede llevar a una degradación notable en la precisión o "inteligencia" del modelo.[2] El consenso en la comunidad sugiere que los formatos Q4 (cuantización de 4 bits) son a menudo el nivel mínimo aceptable para mantener la calidad de la respuesta para la mayoría de las tareas.[12]
    - **Impacto Práctico:** Usando el modelo Llama 3 de 70 mil millones de parámetros como ejemplo: en precisión FP16, requiere aproximadamente 140 GB de VRAM, una cantidad inaccesible para la mayoría. Sin embargo, con cuantización de 4 bits, este requisito se reduce a aproximadamente 42 GB, lo que lo pone al alcance de una configuración de doble GPU para consumidores.[6]

1.3 Selección de la GPU Adecuada: Un Análisis de Mercado

El mercado de GPUs está dominado por unos pocos actores clave, pero la elección correcta depende del presupuesto, el caso de uso y el ecosistema de software.

- **El Dominio de NVIDIA:** Es innegable que NVIDIA lidera el mercado de la IA debido a la madurez y robustez de su ecosistema de software CUDA. La mayoría de los frameworks y herramientas de IA están diseñados y optimizados principalmente para GPUs NVIDIA, lo que las convierte en la "opción segura y recomendada" para evitar problemas de compatibilidad y obtener el mejor rendimiento posible.[14] AMD y Apple son alternativas viables, especialmente con la mejora del soporte de ROCm para AMD, pero pueden requerir más configuración y solución de problemas.[14]
- **Niveles de GPU y Recomendaciones:**
    - **Nivel de Entrada (Experimentación y Modelos Pequeños, <13B):** GPUs con 12-16 GB de VRAM.
        - **NVIDIA:** GeForce RTX 3060 (12 GB), RTX 4060 Ti (16 GB), RTX 4070 SUPER (12 GB).[2]
        - **AMD:** RX 7600 XT (16 GB).[14]
        - Estas tarjetas son adecuadas para ejecutar modelos de 7B cómodamente y modelos de 13B con una cuantización agresiva.

- **Gama Media / Prosumidor (Aficionados Serios y PYMES, hasta 70B cuantizados):** GPUs con 24 GB de VRAM.
  - **NVIDIA:** GeForce RTX 3090 (usada), RTX 4090. Este nivel de 24 GB se describe como el "punto dulce" para los LLMs locales. Es capaz de ejecutar modelos de 32B fácilmente e incluso modelos de 70B con cuantización de 4 bits.[15] Una configuración de doble RTX 3090 o 4090 es una opción popular y potente para los entusiastas y las pequeñas empresas.[20]
- **Gama Alta / Empresarial (Modelos Grandes y Ajuste Fino):** GPUs con 32 GB+ de VRAM.
  - **NVIDIA:** RTX 5090 (32 GB), RTX 6000 Ada (48 GB), L40S (48 GB), A100 (40/80 GB), H100/H200/B200 (80 GB+).[19] Estas son necesarias para ejecutar modelos muy grandes (>70B) sin una cuantización pesada o para realizar un ajuste fino completo.
- **Apple Silicon (Arquitectura de Memoria Unificada):** Esta es una categoría especial que desafía el modelo tradicional de GPU discreta. Los Mac Minis, Studios y Pros con chips de la serie M utilizan memoria unificada, donde la RAM del sistema también es la VRAM. Esto permite configuraciones con cantidades masivas de memoria (por ejemplo, 192 GB o 512 GB) en un formato de escritorio.[20] La arquitectura de memoria unificada permite ejecutar modelos extremadamente grandes, como uno de 671B, en una sola máquina de escritorio, algo imposible incluso con múltiples GPUs de consumo de gama alta.[22] El compromiso es un menor ancho de banda de memoria en comparación con una GPU dedicada de gama alta como una RTX 4090, lo que resulta en velocidades de generación de tokens más lentas.[20] Esto crea una división en el mercado: para obtener la máxima velocidad en modelos de tamaño moderado, las GPUs discretas de NVIDIA son la mejor opción. Pero para ejecutar los modelos más grandes posibles localmente, independientemente de la velocidad, las arquitecturas de memoria unificada presentan una alternativa convincente y a menudo más rentable.

1.4 Más Allá de la GPU: Requisitos de Hardware a Nivel de Sistema

Aunque la GPU es el componente central, el resto del sistema debe estar equilibrado para evitar cuellos de botella.

- **CPU:** Un procesador multinúcleo moderno (Intel Core i5/i7/i9, AMD Ryzen 5/7/9) es suficiente para la mayoría de las configuraciones. El papel de la CPU es principalmente la carga de datos, el preprocesamiento y la gestión del sistema, no el cálculo de inferencia principal.[25] Para construcciones de servidores dedicados, se recomiendan procesadores de grado de servidor como AMD EPYC o Intel Xeon.[21]
- **RAM del Sistema:** Debe ser suficiente para soportar el sistema operativo y las aplicaciones, pero también para manejar la carga del modelo antes de que se transfiera a la VRAM. Una buena regla general es tener al menos tanta RAM de sistema como

VRAM de GPU, con 32 GB como una base sólida y 64 GB o más recomendados para un trabajo serio.[15]

- **Almacenamiento:** Un SSD NVMe rápido es fundamental para reducir los tiempos de carga del modelo. Los archivos de los modelos pueden ser enormes (un modelo de 70B puede superar los 100 GB), por lo que un almacenamiento lento crea un cuello de botella significativo al iniciar o cambiar de modelo.[15] Se recomienda al menos 1-2 TB de almacenamiento NVMe.
- **Alimentación y Refrigeración:** Las GPUs de gama alta como la RTX 4090 pueden consumir 450W o más bajo carga.[15] Esto requiere una Fuente de Alimentación (PSU) de alta calidad de 1000W-1200W y un chasis con un excelente flujo de aire o refrigeración líquida para evitar el estrangulamiento térmico, que puede degradar severamente el rendimiento.[18]
- **Ejemplos de Configuraciones:**
    - **Configuración "Experimentador con Presupuesto" (~1500 USD):** CPU AMD Ryzen 5, 32 GB RAM DDR5, NVIDIA RTX 4060 Ti 16GB, 1 TB NVMe SSD, PSU de 750W. Ideal para modelos de hasta 13B.
    - **Configuración "Potencia Prosumidor" (~4000 USD):** CPU Intel Core i7/AMD Ryzen 7, 64 GB RAM DDR5, NVIDIA RTX 4090 24GB, 2 TB NVMe SSD, PSU de 1200W, refrigeración líquida. Capaz de ejecutar modelos de 70B cuantizados a alta velocidad.[17]
    - **Configuración "Servidor PYME" (~8000 USD):** CPU AMD EPYC, 256 GB RAM DDR4 ECC, 2x NVIDIA RTX 3090 24GB (usadas), 4 TB NVMe SSD, PSU de 1200W Platinum, chasis de servidor. Diseñado para cargas de trabajo multiusuario y ajuste fino.[21]

Parte II: La Sala de Máquinas: El Ecosistema de Software de LLM Local

Esta sección pasa del hardware a la pila de software, proporcionando un mapa claro de las herramientas necesarias para ejecutar, gestionar y construir sobre LLMs locales. Está estructurada como una guía para la toma de decisiones, ayudando al lector a elegir las herramientas adecuadas para sus objetivos específicos, desde un simple chat hasta un servicio de producción de alto rendimiento.

2.1 Sistema Operativo y Controladores Fundamentales

- **Elección del Sistema Operativo:** Aunque Windows y macOS son compatibles con muchas herramientas, Linux (específicamente una distribución basada en Ubuntu) es el sistema operativo preferido para el desarrollo serio de IA debido a un soporte de

controladores superior, una mayor compatibilidad con herramientas de desarrollo y un mejor rendimiento.[17] NixOS se menciona como una opción potente pero compleja para usuarios avanzados que buscan entornos reproducibles.[28]

- **NVIDIA CUDA Toolkit:** Este es un requisito no negociable para ejecutar LLMs en GPUs NVIDIA. Proporciona el compilador, las bibliotecas y las herramientas que permiten al software acceder a las capacidades de procesamiento paralelo de la GPU. Es crucial instalar la versión correcta que sea compatible con los frameworks elegidos (por ejemplo, PyTorch) para evitar problemas de incompatibilidad.[16]

2.2 Una Taxonomía de Herramientas de LLM Local

El panorama de herramientas puede ser confuso. Se puede categorizar según su caso de uso principal para proporcionar un modelo mental claro. Esta división del mercado en dos corrientes distintas —una centrada en la facilidad de uso para la experimentación y otra en el alto rendimiento para la producción— representa la primera decisión crítica de software que un desarrollador debe tomar.

- **Categoría 1: Interfaces Fáciles de Usar (Para Experimentación y Facilidad de Uso)**
  - **LM Studio:** Una aplicación de escritorio con una interfaz gráfica de usuario (GUI) pulida para descubrir, descargar (desde Hugging Face) y chatear con LLMs. Sus características clave son la simplicidad, un servidor local compatible con la API de OpenAI y la capacidad de chatear con documentos locales (RAG). Soporta los formatos de modelo GGUF y MLX, lo que lo hace muy versátil para los usuarios de escritorio.[30]
  - **Ollama:** Una herramienta de línea de comandos que simplifica la descarga y ejecución de modelos. Es elogiado por su simplicidad y a menudo se utiliza como un servidor backend para otras aplicaciones, como Open WebUI, que proporciona una interfaz de chat. Es ideal para desarrolladores que desean poner en marcha un modelo rápidamente con un simple comando ollama pull.[9]
  - **Oobabooga's Text-Generation-WebUI:** Una interfaz web altamente personalizable y rica en funciones. Va más allá del simple chat para incluir parámetros detallados de generación, modos de personaje, pestañas de entrenamiento y un extenso sistema de extensiones. Es la favorita de los usuarios avanzados y experimentadores que desean el máximo control sobre el comportamiento del modelo.[37]
- **Categoría 2: Motores de Inferencia de Alto Rendimiento (Para Producción y Escala)**
  - **vLLM:** Una biblioteca de código abierto de UC Berkeley diseñada para el servicio de LLM de alto rendimiento y eficiencia de memoria. Su innovación clave es **PagedAttention**, que optimiza el uso de VRAM para la caché KV, permitiendo tamaños de lote más grandes y una mayor concurrencia.[9] Los benchmarks muestran que supera drásticamente a Ollama en escenarios multiusuario, con un rendimiento hasta 19 veces mayor, lo que lo convierte en la opción preferida para

la implementación en producción.[42]

- **NVIDIA TensorRT-LLM:** La propia biblioteca de código abierto de NVIDIA para optimizar la inferencia. Logra el máximo rendimiento en hardware NVIDIA al compilar los modelos en motores altamente optimizados, utilizando técnicas como la fusión de kernels y optimizaciones de grafos. A menudo tiene una curva de aprendizaje más pronunciada, requiriendo un paso de conversión del modelo, pero puede ofrecer el mejor rendimiento absoluto, especialmente con cuantización avanzada (FP8/INT4) en GPUs empresariales.[43]

2.3 Planos de Instalación y Configuración

Esta sección proporciona guías prácticas y paso a paso para comenzar.

- **Plano A: El Comienzo Simple con Ollama**
  1. Descargar e instalar Ollama para su sistema operativo desde el sitio web oficial.[36]
  2. Abrir una terminal y ejecutar ollama pull llama3.1 (o cualquier otro modelo disponible) para descargar y configurar el modelo.[27]
  3. Ejecutar ollama run llama3.1 para comenzar a chatear directamente en la terminal.
  4. (Opcional) Instalar Open WebUI a través de Docker para obtener una interfaz similar a ChatGPT que se conecta al servidor Ollama en ejecución, proporcionando una experiencia de usuario más rica.[9]
- **Plano B: La Ruta del Rendimiento con vLLM**
  1. Configurar un entorno de Python e instalar vLLM: pip install vllm.[42]
  2. Iniciar el servidor vLLM con un comando que apunte a un modelo de Hugging Face: python -m vllm.entrypoints.openai.api_server --model meta-llama/Llama-3.1-8B-Instruct.
  3. Este comando inicia un servidor de API compatible con OpenAI, listo para recibir solicitudes.[9]
  4. Interactuar con el servidor utilizando llamadas de API estándar (por ejemplo, con curl o un script de Python), demostrando su uso como un backend para una aplicación personalizada.

2.4 Frameworks de Desarrollo de Aplicaciones

Una vez que un modelo está en funcionamiento, estos frameworks permiten construir aplicaciones complejas sobre él.

- **Hugging Face Transformers:** La biblioteca estándar de facto para interactuar con modelos transformer. Proporciona las herramientas para descargar modelos y tokenizadores desde el Hub y ejecutarlos localmente utilizando las clases pipeline o AutoModel. Esta es la capa fundamental sobre la que se construyen muchas otras herramientas.[46]
- **LangChain:** Un framework para construir aplicaciones conscientes del contexto y con capacidad de razonamiento con LLMs. Proporciona componentes modulares ("Chains")

para conectar LLMs a fuentes de datos, gestionar el historial de conversaciones e interactuar con herramientas externas. LangChain tiene integraciones para casi todos los servidores de LLM locales (Ollama, vLLM, etc.), lo que lo hace esencial para construir aplicaciones complejas sobre un modelo local.[36]

La estandarización en torno a una "API compatible con OpenAI" se ha convertido en una fuerza poderosa para la interoperabilidad del ecosistema. Herramientas de alto rendimiento como vLLM [9], plataformas fáciles de usar como LM Studio [30], e incluso bibliotecas específicas de hardware como TensorRT-LLM, exponen un punto final de API que imita el de OpenAI. Esto significa que una aplicación desarrollada inicialmente para comunicarse con el servicio en la nube de OpenAI puede ser redirigida a un servidor de modelo local con cambios mínimos en el código. Esta estandarización reduce drásticamente la fricción de pasar de un prototipo basado en la nube a una implementación local centrada en la privacidad y el control de costos, y permite que frameworks como LangChain construyan una única integración que funcione en docenas de proveedores de modelos diferentes.[52]

Tabla 1: Comparación de Herramientas de Software de LLM Local

Herramienta	Caso de Uso Principal	Característica Clave	Facilidad de Uso	Rendimiento (Usuario Único)	Rendimiento (Multiusuario)	Compatibilidad de Modelos
Ollama	Experimentación rápida, Backend para desarrolladores	Simplicidad/ CLI (ollama pull)	Muy Alta	Alto	Bajo	GGUF
LM Studio	Experimentación en escritorio, Principiantes	GUI pulida, RAG local	Muy Alta	Alto	Bajo	GGUF, MLX
Oobabooga WebUI	Experimentación avanzada, Control de parámetros	Interfaz web altamente configurable, Extensiones	Media	Alto	Bajo	Múltiples (GGUF, GPTQ, etc.)
vLLM	Servicio de producción, Alto rendimiento	PagedAttention, Alta concurrencia	Media	Alto	Muy Alto	Modelos de Hugging Face
TensorRT-LLM	Rendimiento máximo en NVIDIA,	Optimización a nivel de kernel,	Baja	Muy Alto	Muy Alto	Modelos de Hugging Face

	Entornos empresariales	Compilación de modelos				(requiere conversión)

Fuentes: [9]

Parte III: Un Estudio de los Principales LLMs de Código Abierto

Esta sección sirve como un catálogo y análisis de los modelos de código abierto más importantes, ayudando al usuario a comprender sus opciones y a seleccionar un modelo que se alinee con sus requisitos de rendimiento y caso de uso específicos.

3.1 El Panorama de los Modelos: Familias y Actores Clave

- **La Serie Llama de Meta:** La familia Llama, en particular Llama 3, se presenta como una línea de modelos de primer nivel y alto rendimiento. La familia incluye varios tamaños como 8B y 70B, que están ajustados por instrucciones para chat y tareas de propósito general. Llama 3 se destaca por sus sólidas capacidades de razonamiento y escalabilidad.[5][6]
- **Los Modelos de Mistral AI:** Una empresa francesa que ha producido modelos altamente influyentes y eficientes.
    - **Mistral 7B:** Famoso por su rendimiento, superando a modelos más grandes como Llama 2 13B siendo mucho más pequeño y rápido. Es una opción preferida para aplicaciones centradas en la eficiencia.[3][4]
    - **Modelos Mixtral:** Utilizan una arquitectura dispersa de Mezcla de Expertos (MoE). Por ejemplo, Mixtral 8x7B tiene un total de ~47 mil millones de parámetros, pero solo activa una fracción (~13 mil millones) para cualquier token dado, ofreciendo el rendimiento de un modelo mucho más grande con la velocidad de inferencia y el costo de uno más pequeño.[5][6]
- **La Serie Qwen de Alibaba:** Una potente familia de modelos, con Qwen1.5 y las más recientes líneas Qwen 2 y 3 mostrando un rendimiento muy competitivo en los benchmarks, incluyendo capacidades multilingües y de codificación.[10]
- **Modelos DeepSeek AI:** Estos modelos se destacan particularmente por sus sólidas habilidades de codificación y razonamiento, lo que los convierte en excelentes opciones para herramientas de desarrollo y tareas complejas de resolución de problemas.[3][4]
- **Otros Modelos Notables:** Una breve descripción de otros modelos importantes como Gemma de Google, Falcon de EleutherAI y modelos especializados como Vicuna (ajustado para chat).[5][6]

3.2 Categorización de Modelos por Capacidad

- **Propósito General / Razonamiento:** Llama 3 (70B), Mixtral 8x7B y los modelos Qwen más grandes son los principales contendientes para tareas que requieren razonamiento complejo, matices y generación de texto de alta fidelidad.[57]
- **Codificación y Tareas Técnicas:** DeepSeek-Coder, las variantes de codificación de Qwen y Codestral de Mistral están específicamente optimizados para la generación, depuración y explicación de código.[34]
- **Eficiencia y Velocidad:** Mistral 7B es el modelo destacado para aplicaciones donde la baja latencia y la eficiencia de recursos son primordiales, como chatbots en tiempo real o aplicaciones en el dispositivo.[58]
- **Multimodal (Visión):** La capacidad de procesar imágenes y texto es una capacidad emergente en los modelos de código abierto. Ejemplos incluyen Falcon 2 11B VLM, LLaVA y versiones con capacidad de visión de Qwen y Gemma.[34]

3.3 Análisis Comparativo: Rendimiento vs. Eficiencia

El panorama de los modelos de código abierto no es una simple escala de "bueno, mejor, óptimo". Ha evolucionado hacia distintas filosofías arquitectónicas que crean un complejo equilibrio entre la potencia bruta, la eficiencia computacional y la idoneidad para tareas específicas.

- **El Compromiso Central:** Inicialmente, el progreso en los LLMs se consideraba una función de la escala: más parámetros significaban un mejor modelo, representado por modelos grandes y densos como Llama 3 70B.[61] Sin embargo, modelos como Mistral 7B demostraron que las optimizaciones arquitectónicas podían permitir que un modelo mucho más pequeño superara a predecesores más grandes, rompiendo la simple suposición de "tamaño = calidad".[60] La introducción de la arquitectura de Mezcla de Expertos (MoE) en modelos como Mixtral representa un tercer camino, que busca proporcionar la capacidad de conocimiento de un modelo muy grande (por ejemplo, 47B de parámetros totales) mientras solo incurre en el costo de inferencia de uno mucho más pequeño (~13B de parámetros activos).[56] Esto crea un espacio de decisión multidimensional:
    - **Llama 3 70B:** Representa el rendimiento máximo. Sobresale en conversaciones complejas de múltiples turnos y análisis profundos, pero requiere un hardware significativo (por ejemplo, 2x RTX 4090 incluso cuando está cuantizado).[6]
    - **Mistral 7B / Mixtral 8x7B:** Representan la máxima eficiencia. Mistral 7B puede ejecutarse en una sola GPU de consumo con 12-16 GB de VRAM. Mixtral ofrece un rendimiento que se acerca a los modelos de la clase 70B pero con requisitos computacionales mucho menores debido a su arquitectura MoE.[61]
- **Interpretación de Benchmarks:** Si bien las tablas de clasificación públicas (MMLU,

GSM8K, etc.) son útiles, no cuentan toda la historia. El rendimiento de un modelo en una tarea empresarial específica y del mundo real puede diferir significativamente. Es fundamental realizar benchmarks de los modelos con los propios datos de la empresa para tomar una decisión informada.[63]

Los rápidos ciclos de lanzamiento y la transparencia en las licencias de actores como Mistral AI y Meta están creando un entorno hipercompetitivo. Esta competencia obliga a los creadores de modelos a superar continuamente los límites del rendimiento mientras mantienen los modelos accesibles, a menudo bajo licencias permisivas como Apache 2.0 o licencias personalizadas comercialmente viables.[56] El resultado para una empresa es que las capacidades de IA de vanguardia o casi de vanguardia se están convirtiendo en una mercancía disponible gratuitamente. La ventaja competitiva se está desplazando de simplemente
tener un modelo potente a cuán eficazmente una empresa puede *personalizar e integrar* ese modelo en sus flujos de trabajo y ecosistemas de datos únicos.

Parte IV: Implementación Estratégica para Flujos de Trabajo Empresariales

Esta sección final y crucial sintetiza toda la información técnica precedente en un marco estratégico para la aplicación empresarial. Aborda directamente la consulta del usuario sobre qué LLMs son mejores para tareas administrativas, financieras y comerciales, explicando *cómo* adaptarlos para estos roles.

4.1 Desbloqueo de Datos Empresariales: RAG vs. Ajuste Fino

Para que un LLM sea verdaderamente útil en un contexto empresarial, debe poder acceder y razonar sobre los datos y procesos específicos de la empresa. Existen dos técnicas principales para lograr esto.

- **Conceptos Fundamentales:**
    - **Ajuste Fino (Fine-Tuning):** Es el proceso de reentrenar un modelo preentrenado en un conjunto de datos más pequeño y específico del dominio. Esto modifica los pesos internos del modelo para "incorporar" nuevos conocimientos, estilo o comportamiento. Es análogo a enseñar a un generalista a convertirse en un especialista, como un abogado o un médico, aprendiendo la jerga, los matices y los patrones de un campo específico.[67]
    - **Generación Aumentada por Recuperación (RAG):** Es el proceso de proporcionar a un LLM información externa y actualizada en el momento de la inferencia. El LLM utiliza este contexto recuperado para responder a una consulta sin cambiar sus pesos internos. Es como darle a un generalista un libro abierto a la base de conocimientos privada y en tiempo real de la empresa.[67]

- **El Marco de Decisión:** La elección entre RAG y ajuste fino es una de las decisiones estratégicas más críticas.
    - **Actualidad de los Datos:** RAG sobresale con datos dinámicos y que cambian rápidamente (por ejemplo, inventario, informes financieros recientes, tickets de soporte al cliente). Los modelos ajustados son estáticos y su conocimiento se vuelve obsoleto con el tiempo.[67]
    - **Tipo de Tarea:** El ajuste fino es superior para enseñar a un modelo una nueva *habilidad* o *estilo* (por ejemplo, adoptar una voz de marca específica, generar informes estructurados, clasificar cláusulas legales). RAG es para proporcionar nuevo *conocimiento*.[68]
    - **Costo y Esfuerzo:** RAG es generalmente más barato y rápido de implementar inicialmente, requiriendo habilidades de ingeniería de datos. El ajuste fino es computacionalmente costoso y requiere experiencia en aprendizaje automático y datos etiquetados de alta calidad.[68]
    - **Explicabilidad y Confianza:** RAG es más transparente porque puede citar sus fuentes de los documentos recuperados, lo que reduce las "alucinaciones" o respuestas incorrectas. El ajuste fino es una caja negra; es más difícil rastrear por qué dio una respuesta específica.[67]
    - **Privacidad de los Datos:** RAG a menudo se considera más seguro para datos sensibles, ya que los datos permanecen en una base de datos separada y controlada y solo se pasan al modelo temporalmente como parte del prompt. El ajuste fino integra los datos en el propio modelo, lo que puede presentar un riesgo de fuga de datos si el modelo se ve comprometido.[68]
- **El Enfoque Híbrido (RAFT):** La solución más poderosa a menudo es combinar ambos. Un modelo puede ser ajustado para comprender el lenguaje y la estructura específicos de un dominio (por ejemplo, contratos legales) y luego usar RAG para acceder al contenido de un contrato nuevo y específico para responder preguntas sobre él. Esto combina una profunda experiencia de dominio con datos en tiempo real.[67] Esta comprensión de que RAG y el ajuste fino no son opciones mutuamente excluyentes, sino herramientas complementarias, es fundamental. Las soluciones de IA empresariales más sofisticadas y valiosas serán inevitablemente sistemas híbridos que aprovechen sinérgicamente ambas técnicas.

Tabla 2: Matriz de Decisión RAG vs. Ajuste Fino

Factor de Decisión	Mejor para RAG	Mejor para Ajuste Fino	Implicación Empresarial
Tipo de Datos	Dinámicos, en tiempo real (por ejemplo, tickets de soporte, precios de acciones)	Estáticos, base de conocimiento bien definida (por ejemplo, manuales de	Elija RAG para bases de conocimiento de soporte al cliente; Ajuste Fino para libros

		productos, textos legales)	de texto médicos.
Objetivo Principal	Proporcionar *conocimiento* actualizado y fáctico	Enseñar una *habilidad*, *estilo* o *formato* específico	Use RAG para responder "¿Cuál es el estado del pedido X?"; use Ajuste Fino para "Escribe un resumen de contrato en nuestro formato legal estándar".
Perfil de Costo	Menor costo inicial, mayor costo en tiempo de ejecución (recuperación)	Alto costo inicial (cómputo de entrenamiento), menor costo en tiempo de ejecución	RAG para una implementación rápida y económica; Ajuste Fino para aplicaciones de alto volumen donde la latencia es crítica.
Privacidad de Datos	Alta (los datos permanecen en una base de datos externa y controlada)	Media (los datos se integran en los pesos del modelo)	RAG es preferible para industrias altamente reguladas como la salud y las finanzas.
Explicabilidad	Alta (puede citar las fuentes recuperadas)	Baja (el razonamiento es una "caja negra")	RAG genera confianza al proporcionar evidencia para sus respuestas, crucial para la verificación de hechos.
Tiempo de Implementación	Rápido (semanas)	Lento (meses, debido a la curación de datos y el entrenamiento)	RAG permite una iteración y un tiempo de valorización más rápidos.

Fuentes: [67]

4.2 Caso de Uso Detallado: RAG para Análisis Financiero y Comercial

- **Aplicación:** Construir un asistente inteligente para analistas financieros o equipos de ventas. El objetivo es hacer preguntas en lenguaje natural sobre presentaciones recientes ante la SEC, informes de ganancias o datos de ventas internos.
- **¿Por qué RAG?** Los datos financieros y comerciales son altamente dinámicos y sensibles al tiempo. Un modelo ajustado quedaría obsoleto instantáneamente. RAG permite al LLM acceder a los informes y datos más recientes.[67]

- **Implementación:**
    1. **Ingesta de Datos:** Recopilar informes financieros (PDFs, XMLs) o datos de ventas (desde un CRM).
    2. **Fragmentación (Chunking):** Dividir documentos grandes en fragmentos más pequeños y significativos. Las técnicas avanzadas van más allá de los simples párrafos para utilizar la estructura del documento (por ejemplo, tablas, secciones en un informe 10-K) para una recuperación más precisa.[76]
    3. **Vectorización:** Utilizar un modelo de incrustación (embedding) para convertir estos fragmentos en vectores numéricos y almacenarlos en una base de datos vectorial (por ejemplo, PGVector).[56]
    4. **Recuperación y Generación:** Cuando un usuario hace una consulta ("¿Cuáles fueron los ingresos de Apple en el último trimestre?"), el sistema convierte la consulta en un vector, encuentra los fragmentos más similares en la base de datos y los alimenta al LLM como contexto junto con la pregunta original.[78]
- **Modelos Recomendados:** Un modelo con una gran ventana de contexto y un sólido razonamiento es ideal. **Llama 3 70B** o **Mixtral 8x7B** serían excelentes opciones debido a su capacidad para procesar y razonar sobre información compleja recuperada.[58]

4.3 Caso de Uso Detallado: Ajuste Fino para Flujos de Trabajo Administrativos y Legales

- **Aplicación:** Crear un asistente para analizar contratos legales, identificando cláusulas específicas (por ejemplo, terminación, responsabilidad), señalando riesgos y resumiéndolos en un formato estandarizado.
- **¿Por qué Ajuste Fino?** El lenguaje legal es altamente especializado. El objetivo es enseñar al modelo la *estructura* y los *matices* de los documentos legales, una habilidad que es consistente en todos los contratos. El modelo necesita aprender a "pensar como un abogado".[68]
- **Implementación:**
    1. **Creación del Conjunto de Datos:** Este es el paso más crítico. Implica reunir un gran conjunto de contratos y crear ejemplos de alta calidad de la salida deseada (por ejemplo, pares de cláusulas de contrato y su correspondiente análisis de riesgo o resumen). La calidad del modelo final es un reflejo directo de la calidad de este conjunto de datos.[80]
    2. **Selección del Modelo:** Un modelo ajustado por instrucciones es el mejor punto de partida. **Llama 3 8B** o **Mistral 7B** son buenos candidatos, ya que son más fáciles y baratos de ajustar que los modelos más grandes.[58]
    3. **Entrenamiento:** Utilizar un método de ajuste fino eficiente en parámetros (PEFT) como LoRA con un framework como Unsloth o Axolotl para entrenar el modelo en el conjunto de datos curado. Esto ajusta un pequeño número de parámetros, haciendo que el proceso sea factible en hardware de gama media.[3]

4. **Evaluación:** Probar la capacidad del modelo ajustado para analizar con precisión contratos nuevos y no vistos.

- **Modelos Recomendados:** Para el ajuste fino, a menudo es más efectivo comenzar con un modelo más pequeño y capaz. **Mistral 7B** es conocido por ser altamente adaptable y eficiente para el ajuste fino.[58]

La barrera principal para la adopción empresarial de LLMs locales personalizados no es el rendimiento del modelo, sino la preparación de los datos y la ingeniería de pipelines. Para RAG, el éxito depende de la calidad del pipeline de datos: fragmentación adecuada de documentos, incrustaciones efectivas y una base de datos vectorial bien mantenida.[69] Para el ajuste fino, la parte más crítica y laboriosa es la creación de un conjunto de datos etiquetado de alta calidad.[80] Las habilidades requeridas para estas tareas son principalmente de ingeniería de datos, gobernanza de datos y experiencia en el dominio, no habilidades exóticas de investigación en IA.[71] Por lo tanto, el viaje hacia un LLM local personalizado comienza no con la elección de una GPU, sino con una estrategia de datos.

4.4 Integración de LLMs en la Automatización de Procesos de Negocio

- **El Puente de la API:** Los LLMs locales que se ejecutan en servidores como vLLM u Ollama exponen una API. Esta API es la clave para la integración.
- **Conexión a Plataformas de Automatización:** Herramientas como Zapier, Make o la de código abierto n8n pueden realizar solicitudes HTTP. Al llamar al punto final de la API del LLM local, se puede insertar la toma de decisiones o la generación de texto impulsada por IA en cualquier flujo de trabajo automatizado.[82]
- **Flujo de Trabajo de Ejemplo:**
    1. **Disparador:** Llega un nuevo correo electrónico de soporte al cliente a una bandeja de entrada compartida.
    2. **Acción 1 (n8n):** Leer el contenido del correo electrónico.
    3. **Acción 2 (n8n):** Enviar el contenido del correo electrónico a un modelo local **Mistral 7B** a través de su API con un prompt como "Clasifica este correo electrónico en una de las siguientes categorías: Soporte Técnico, Consulta de Facturación, Cliente Potencial de Ventas".
    4. **Acción 3 (n8n):** Según la clasificación del modelo, enrutar automáticamente el correo electrónico al canal de Slack o a la cola del CRM del departamento correcto.

Parte V: Recomendaciones Finales y Perspectivas Futuras

5.1 Hojas de Ruta Accionables para la Implementación

- **Para el Desarrollador Individual / Investigador:** Comenzar con una GPU NVIDIA de consumo (RTX 3060 12GB o 4060 Ti 16GB). Utilizar Ollama o LM Studio para una fácil experimentación con modelos cuantizados de 7B y 13B como Mistral 7B y Llama 3 8B. Construir aplicaciones utilizando LangChain para orquestar flujos de trabajo complejos.
- **Para la Pequeña y Mediana Empresa (PYME):** Invertir en una estación de trabajo "prosumidor" con una RTX 4090 (24GB). Comenzar con una implementación de RAG utilizando un modelo potente como Mixtral o Llama 3 70B (cuantizado) servido a través de vLLM. Esto proporciona un valor inmediato al desbloquear los datos propietarios de la empresa con una sobrecarga técnica menor que el ajuste fino.
- **Para la Startup Centrada en IA / Laboratorio de Investigación:** Construir un servidor dedicado con múltiples GPUs (2x-4x RTX 3090/4090 o tarjetas empresariales como A100). Seguir una estrategia híbrida de RAG + Ajuste Fino. Ajustar modelos más pequeños y especializados en datos propietarios y utilizarlos dentro de un marco RAG para obtener lo mejor de ambos mundos.

5.2 El Futuro de la IA Local

- **Tendencias de Hardware:** El enfoque continuará en el ancho de banda y la capacidad de la memoria. El auge de los aceleradores específicos para IA y los SoCs con grandes pools de memoria unificada continuará, desafiando el modelo tradicional de GPU discreta para ciertos casos de uso.[22]
- **Tendencias de Modelos:** Se observará un movimiento hacia arquitecturas aún más eficientes (por ejemplo, MoE avanzado, modelos de espacio de estado) que ofrezcan el rendimiento de los modelos más grandes de hoy con una fracción del costo computacional. La cuantización se volverá aún más agresiva y efectiva, potencialmente con una pérdida de calidad mínima.
- **Tendencias de Software:** Los motores de inferencia se volverán más optimizados y fáciles de usar, difuminando la línea entre las categorías de "facilidad de uso" y "alto rendimiento". Las herramientas para la preparación y evaluación de datos (tanto para RAG como para ajuste fino) se volverán más sofisticadas y automatizadas, reduciendo la carga de la ingeniería de datos.

Obras citadas

1. Optimizing VRAM for efficient LLM performance - Barrage, fecha de acceso: septiembre 21, 2025, https://www.barrage.net/blog/technology/optimizing-vram-for-efficient-llm-performance
2. Sizing VRAM to Generative AI & LLM Workloads - Puget Systems, fecha de acceso: septiembre 21, 2025, https://www.pugetsystems.com/labs/articles/sizing-vram-to-generative-ai-and-ll

m-workloads/
3. How Much VRAM Do You Need for LLMs | Hyperstack, fecha de acceso: septiembre 21, 2025, https://www.hyperstack.cloud/blog/case-study/how-much-vram-do-you-need-for-llms
4. How To Calculate GPU VRAM Requirements for an Large-Language Model, fecha de acceso: septiembre 21, 2025, https://apxml.com/posts/how-to-calculate-vram-requirements-for-an-llm
5. Can You Run This LLM? VRAM Calculator (Nvidia GPU and Apple Silicon), fecha de acceso: septiembre 21, 2025, https://apxml.com/tools/vram-calculator
6. How much VRAM do I need for LLM inference? | Modal Blog, fecha de acceso: septiembre 21, 2025, https://modal.com/blog/how-much-vram-need-inference
7. LLM Model Size, Memory Requirements, and Quantization - Birow, fecha de acceso: septiembre 21, 2025, https://www.birow.com/llm-meret-memoriaigeny-kvantalas
8. medium.com, fecha de acceso: septiembre 21, 2025, https://medium.com/@lmpo/a-guide-to-estimating-vram-for-llms-637a7568d0ea#:~:text=To%20perform%20large%20language%20model,Precision%20%2F%208)%20%C3%97%201.2.
9. LLM VRAM Calculator for Self-Hosting - Research AIMultiple, fecha de acceso: septiembre 21, 2025, https://research.aimultiple.com/self-hosted-llm/
10. Optimizing generative AI models with quantization | Red Hat Developer, fecha de acceso: septiembre 21, 2025, https://developers.redhat.com/articles/2025/08/18/optimizing-generative-ai-models-quantization
11. How Much VRAM My LLM Model Needs? - YouTube, fecha de acceso: septiembre 21, 2025, https://www.youtube.com/watch?v=lJufykNYGRs
12. Q2 models are utterly useless. Q4 is the minimum quantization level that doesn't ruin the model (at least for MLX). Example with Mistral Small 24B at Q2 - Reddit, fecha de acceso: septiembre 21, 2025, https://www.reddit.com/r/LocalLLaMA/comments/1ji7oh6/q2_models_are_utterly_useless_q4_is_the_minimum/
13. Should I just get more RAM? - Beginners - Hugging Face Forums, fecha de acceso: septiembre 21, 2025, https://discuss.huggingface.co/t/should-i-just-get-more-ram/132589
14. PSA: Local LLM Hardware Requirements : r/homeassistant - Reddit, fecha de acceso: septiembre 21, 2025, https://www.reddit.com/r/homeassistant/comments/1hovutx/psa_local_llm_hardware_requirements/
15. Choosing the Right GPU for Local LLM Use | by Thongchan Thananate - Medium, fecha de acceso: septiembre 21, 2025, https://klaothongchan.medium.com/choosing-the-right-gpu-for-local-llm-use-35392b4822a8
16. CUDA Toolkit - Free Tools and Training | NVIDIA Developer, fecha de acceso: septiembre 21, 2025, https://developer.nvidia.com/cuda-toolkit

17. Hardware requirements for running the large language model Deepseek R1 locally., fecha de acceso: septiembre 21, 2025, https://www.rnfinity.com/news/Hardware-requirements-for-running-large-language-model-Deepseek-R1-on-a-local-machine
18. Running LLMs on your computer locally — focus on the hardware! | by Michael McAnally, fecha de acceso: septiembre 21, 2025, https://michael-mcanally.medium.com/running-llms-on-your-computer-locally-75717bd38d5e
19. General recommended VRAM Guidelines for LLMs - DEV Community, fecha de acceso: septiembre 21, 2025, https://dev.to/simplr_sh/general-recommended-vram-guidelines-for-llms-4ef3
20. Hardware requirement for coding with local LLM ? : r/LocalLLM - Reddit, fecha de acceso: septiembre 21, 2025, https://www.reddit.com/r/LocalLLM/comments/1l0kwyr/hardware_requirement_for_coding_with_local_llm/
21. Building a PC to run local LLMs and Gen AI : r/LocalLLM - Reddit, fecha de acceso: septiembre 21, 2025, https://www.reddit.com/r/LocalLLM/comments/1ioxntp/building_a_pc_to_run_local_llms_and_gen_ai/
22. Local LLM Hardware Guide 2025: Pricing & Specifications - Introl, fecha de acceso: septiembre 21, 2025, https://introl.com/blog/local-llm-hardware-pricing-guide-2025
23. Top NVIDIA GPUs for LLM Inference | by Bijit Ghosh - Medium, fecha de acceso: septiembre 21, 2025, https://medium.com/@bijit211987/top-nvidia-gpus-for-llm-inference-8a5316184a10
24. BEST hardware for running LLMs locally xpost from r/localllLlama : r/LocalLLM - Reddit, fecha de acceso: septiembre 21, 2025, https://www.reddit.com/r/LocalLLM/comments/1it9x5w/best_hardware_for_running_llms_locally_xpost_from/
25. Running LLM Inference Locally: A Guide to Deploying Large Language Models on Your Computer | by Naresh Kumar Amrutham | Medium, fecha de acceso: septiembre 21, 2025, https://namrutham.medium.com/running-llm-inference-locally-a-guide-to-deploying-large-language-models-on-your-computer-2c931e5780e9
26. Recommended Hardware for Running LLMs Locally - GeeksforGeeks, fecha de acceso: septiembre 21, 2025, https://www.geeksforgeeks.org/deep-learning/recommended-hardware-for-running-llms-locally/
27. Building a Low-Cost Local LLM Server to Run 70 Billion Parameter Models - Comet, fecha de acceso: septiembre 21, 2025, https://www.comet.com/site/blog/build-local-llm-server/
28. Top 7 BEST Linux Distros For Software Developers in 2025! (#3 Will SHOCK YOU), fecha de acceso: septiembre 21, 2025, https://www.youtube.com/watch?v=qxs9HtDe8nU

29. NVIDIA GeForce RTX AI PCs | Powering Advanced AI, fecha de acceso: septiembre 21, 2025, https://www.nvidia.com/en-us/ai-on-rtx/
30. What is LM Studio? Features, Pricing, and Use Cases - Walturn, fecha de acceso: septiembre 21, 2025, https://www.walturn.com/insights/what-is-lm-studio-features-pricing-and-use-cases
31. LM Studio - Local AI on your computer, fecha de acceso: septiembre 21, 2025, https://lmstudio.ai/
32. LM Studio | AI Agents Directory, fecha de acceso: septiembre 21, 2025, https://aiagentslist.com/agent/lmstudio
33. Get started with LM Studio | LM Studio Docs, fecha de acceso: septiembre 21, 2025, https://lmstudio.ai/docs/app/basics
34. How to Run a Local LLM: Complete Guide to Setup & Best Models (2025) - n8n Blog, fecha de acceso: septiembre 21, 2025, https://blog.n8n.io/local-llm/
35. Ask HN: What's Your Useful Local LLM Stack? - Hacker News, fecha de acceso: septiembre 21, 2025, https://news.ycombinator.com/item?id=44572043
36. Guide to Local LLMs - Scrapfly, fecha de acceso: septiembre 21, 2025, https://scrapfly.io/blog/posts/guide-to-local-llm
37. How To Install TextGen WebUI and Use ANY MODEL Locally! - GPU Mart, fecha de acceso: septiembre 21, 2025, https://www.gpu-mart.com/blog/how-to-install-textgen-webui
38. text-generation-webui v3.4: Document attachments (text and PDF files), web search, message editing, message "swipes", date/time in messages, branch chats at specific locations, darker UI + more! : r/Oobabooga - Reddit, fecha de acceso: septiembre 21, 2025, https://www.reddit.com/r/Oobabooga/comments/1kyqkyx/textgenerationwebui_v34_document_attachments_text/
39. Home · oobabooga/text-generation-webui Wiki - GitHub, fecha de acceso: septiembre 21, 2025, https://github.com/oobabooga/text-generation-webui/wiki
40. Text Generation WebUI on RunPod: Run LLMs with Ease, fecha de acceso: septiembre 21, 2025, https://www.runpod.io/articles/guides/text-generation-web-ui
41. Text Generation Web UI - Voxta Documentation, fecha de acceso: septiembre 21, 2025, https://doc.voxta.ai/docs/text-generation-web-ui/
42. Ollama vs. vLLM: A deep dive into performance benchmarking | Red Hat Developer, fecha de acceso: septiembre 21, 2025, https://developers.redhat.com/articles/2025/08/08/ollama-vs-vllm-deep-dive-performance-benchmarking
43. vLLM vs TensorRT-LLM: Key differences, performance, and how to run them - Northflank, fecha de acceso: septiembre 21, 2025, https://northflank.com/blog/vllm-vs-tensorrt-llm-and-how-to-run-them
44. vLLM vs. TensorRT-LLM: In-Depth Comparison for Optimizing Large Language Model Inference - Inferless, fecha de acceso: septiembre 21, 2025, https://www.inferless.com/learn/vllm-vs-tensorrt-llm-which-inference-library-is-best-for-your-llm-needs

45. Benchmarking NVIDIA TensorRT-LLM - Jan.ai, fecha de acceso: septiembre 21, 2025, https://jan.ai/post/benchmarking-nvidia-tensorrt-llm
46. Hugging Face Local Pipelines | 🦜 LangChain, fecha de acceso: septiembre 21, 2025, https://python.langchain.com/docs/integrations/llms/huggingface_pipelines/
47. Transformers - Hugging Face, fecha de acceso: septiembre 21, 2025, https://huggingface.co/docs/transformers/index
48. Using transformers at Hugging Face, fecha de acceso: septiembre 21, 2025, https://huggingface.co/docs/hub/transformers
49. Downloading models - Hugging Face, fecha de acceso: septiembre 21, 2025, https://huggingface.co/docs/hub/models-downloading
50. Models - Hugging Face, fecha de acceso: septiembre 21, 2025, https://huggingface.co/docs/transformers/main_classes/model
51. Run models locally | 🦜 LangChain, fecha de acceso: septiembre 21, 2025, https://python.langchain.com/docs/how_to/local_llms/
52. Build a simple LLM application with chat models and prompt templates | 🦜 LangChain, fecha de acceso: septiembre 21, 2025, https://python.langchain.com/docs/tutorials/llm_chain/
53. LLMs - LangChain, fecha de acceso: septiembre 21, 2025, https://python.langchain.com/docs/integrations/llms/
54. LangChain: A Powerful Tool for Local LLM Execution | by Rishabh Nimje | Medium, fecha de acceso: septiembre 21, 2025, https://medium.com/@rishabhnimje123/langchain-a-powerful-tool-for-local-llm-execution-441df81e2ab
55. Choosing your LLM framework: a comparison of Ollama, vLLM, fecha de acceso: septiembre 21, 2025, https://medium.com/ordina-data/choosing-your-llm-framework-a-comparison-of-ollama-vllm-sglang-and-tensorrt-llm-e0cb4a0d1cb8
56. Top 10 open source LLMs for 2025 - NetApp Instaclustr, fecha de acceso: septiembre 21, 2025, https://www.instaclustr.com/education/open-source-ai/top-10-open-source-llms-for-2025/
57. Top 10 Open-Source LLMs in 2025 and Their Use Cases - Great Learning, fecha de acceso: septiembre 21, 2025, https://www.mygreatlearning.com/blog/top-open-source-llms/
58. Mistral vs LLaMA 3: Which Model Solves Your Domain-Specific AI Needs? - Amplework, fecha de acceso: septiembre 21, 2025, https://www.amplework.com/blog/mistral-vs-llama-3-domain-specific-ai/
59. Mistral vs Llama 3: Key Differences & Best Use Cases - Openxcell, fecha de acceso: septiembre 21, 2025, https://www.openxcell.com/blog/mistral-vs-llama-3/
60. Mistral vs Llama 3: Which One Should You Choose? - Novita AI Blog, fecha de acceso: septiembre 21, 2025, https://blogs.novita.ai/mistral-vs-llama-3-which-one-should-you-choose/
61. Choosing Your AI Champion: Mixtral vs. LLaMA 3 in Real-World Use Cases - Medium, fecha de acceso: septiembre 21, 2025,

https://medium.com/@mukulchhabra23/choosing-your-ai-champion-mixtral-vs-llama-3-in-real-world-use-cases-3418fd3e9140
62. Model Catalog - LM Studio, fecha de acceso: septiembre 21, 2025, https://lmstudio.ai/models
63. I locally benchmarked 41 open-source LLMs across 19 tasks and ranked them - Reddit, fecha de acceso: septiembre 21, 2025, https://www.reddit.com/r/LocalLLaMA/comments/1n57hb8/i_locally_benchmarked_41_opensource_llms_across/
64. Best LLMs for Business in 2025: Use-Case Comparison - Tech Research Online, fecha de acceso: septiembre 21, 2025, https://techresearchonline.com/blog/best-llm-for-business-use-case-comparison/
65. Mistral vs Llama 3: Complete Comparison for Voice AI Applications - Vapi AI Blog, fecha de acceso: septiembre 21, 2025, https://vapi.ai/blog/mistral-vs-llama-3
66. Mistral vs Llama: benchmark on your own data - Promptfoo, fecha de acceso: septiembre 21, 2025, https://www.promptfoo.dev/docs/guides/mistral-vs-llama/
67. RAG vs. LLM fine-tuning: Which is the best approach? - Glean, fecha de acceso: septiembre 21, 2025, https://www.glean.com/blog/rag-vs-llm
68. RAG vs. Fine-Tuning: How to Choose - Oracle, fecha de acceso: septiembre 21, 2025, https://www.oracle.com/artificial-intelligence/generative-ai/retrieval-augmented-generation-rag/rag-fine-tuning/
69. RAG vs Fine-Tuning for Enterprise LLMs | by Paul Ferguson, Ph.D. - Towards AI, fecha de acceso: septiembre 21, 2025, https://pub.towardsai.net/rag-vs-fine-tuning-for-enterprise-llms-56f204546a16
70. Fine-tuning LLMs | Thoughtworks United States, fecha de acceso: septiembre 21, 2025, https://www.thoughtworks.com/en-us/insights/decoder/f/fine-tuning-llms
71. RAG vs. fine-tuning - Red Hat, fecha de acceso: septiembre 21, 2025, https://www.redhat.com/en/topics/ai/rag-vs-fine-tuning
72. RAG vs. Fine-tuning - IBM, fecha de acceso: septiembre 21, 2025, https://www.ibm.com/think/topics/rag-vs-fine-tuning
73. A complete guide to retrieval augmented generation vs fine-tuning - Glean, fecha de acceso: septiembre 21, 2025, https://www.glean.com/blog/retrieval-augemented-generation-vs-fine-tuning
74. RAG vs Fine-Tuning: Enterprise AI Strategy Guide - Matillion, fecha de acceso: septiembre 21, 2025, https://www.matillion.com/blog/rag-vs-fine-tuning-enterprise-ai-strategy-guide
75. RAG vs. Fine-Tuning: Why Real-Time AI Outperforms Static Training - DataMotion, fecha de acceso: septiembre 21, 2025, https://datamotion.com/rag-vs-fine-tuning/
76. FinRAG: A Retrieval-Based Financial Analyst - Stanford University, fecha de acceso: septiembre 21, 2025, https://web.stanford.edu/class/cs224n/final-reports/256911814.pdf
77. Financial Report Chunking for Effective Retrieval Augmented Generation - arXiv, fecha de acceso: septiembre 21, 2025, https://arxiv.org/abs/2402.05131

78. Intelligent Financial Data Analysis System Based on LLM-RAG - arXiv, fecha de acceso: septiembre 21, 2025, https://arxiv.org/abs/2504.06279
79. Evaluating Retrieval-Augmented Generation Models for Financial Report Question and Answering - MDPI, fecha de acceso: septiembre 21, 2025, https://www.mdpi.com/2076-3417/14/20/9318
80. Preparing data for fine-tuning LLMs for contract analysis using Data Prep Kit (DPK), fecha de acceso: septiembre 21, 2025, https://developer.ibm.com/tutorials/dpk-fine-tuning-llms/
81. Fine-tuning LLMs Guide | Unsloth Documentation, fecha de acceso: septiembre 21, 2025, https://docs.unsloth.ai/get-started/fine-tuning-llms-guide
82. Best reliable business process automation tools? Bonus points if they're user friendly!, fecha de acceso: septiembre 21, 2025, https://www.reddit.com/r/automation/comments/1mvh4ea/best_reliable_business_process_automation_tools/