Desarrollo y despliegue de los LLM

"La IA generativa es la clave para resolver algunos de los mayores problemas del mundo, como el cambio climático, la pobreza y las enfermedades. Tiene el potencial de hacer del mundo un lugar mejor para todos".

Mark Zuckerberg[37]

En esta sección se analizan los aspectos clave del proceso de desarrollo y despliegue de LLM. Se examinan los componentes principales, como los datos y la arquitectura del modelo, así como las etapas de preentrenamiento, *fine-tuning* e implementación. Además, se discuten los principales retos y consideraciones que deben tenerse en cuenta para garantizar un desarrollo ético, robusto y alineado con los objetivos de una organización.

Aspectos clave en el desarrollo de los LLM

El desarrollo de un LLM es un proceso complejo que implica múltiples componentes y decisiones críticas. A continuación, se describen los principales componentes que es necesario conocer sobre el desarrollo de los LLM y algunos aspectos clave sobre ellos.

Datos

Los datos son el fundamento sobre el cual se construyen los LLM, y su calidad, diversidad y representatividad tienen un impacto directo en el rendimiento y los sesgos del modelo resultante. Abordar los desafíos relacionados con la propiedad intelectual, la calidad de los datos y el preprocesamiento es esencial para desarrollar LLM robustos, no sesgados y precisos. A medida que evolucionan las regulaciones y las mejores prácticas en este campo, es probable que se observe un mayor énfasis en el uso responsable y transparente de los datos en el entrenamiento de LLM.

Algunos aspectos clave sobre los datos de entrenamiento de un LLM son:

- Corpus de entrenamiento[38]: los LLM se entrenan con grandes corpus de datos, a menudo extraídos de internet, que incluyen billones de palabras y abarcan una amplia gama de dominios y géneros, como libros, artículos de noticias, páginas web, redes sociales y más. Estos corpus masivos permiten a los LLM aprender patrones y representaciones del lenguaje a gran escala, lo que les otorga una capacidad sin precedentes para comprender y generar texto coherente y contextualizado. Por ejemplo, corpus comunes para el entrenamiento incluyen BookCorpus[39], Gutenberg[40], Wikipedia[41] o CodeParrot[42].

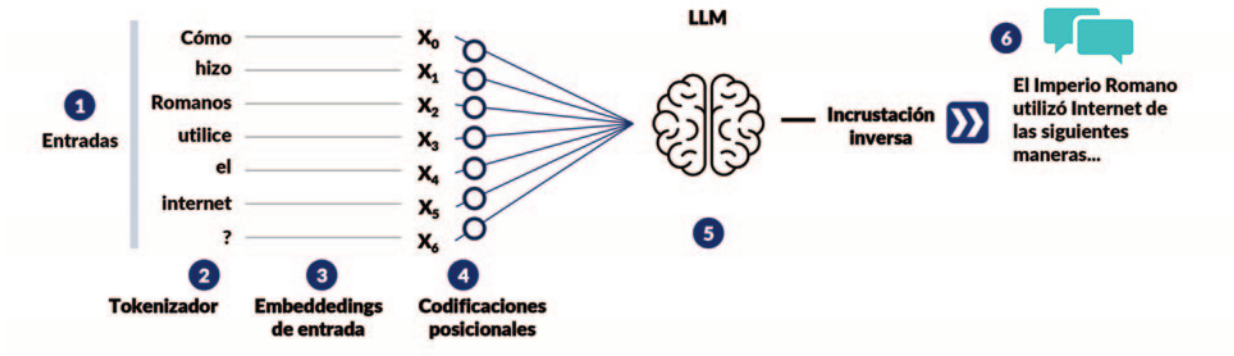
- Propiedad intelectual y derechos de autor[43]: la extracción y uso de datos de internet para entrenar LLM plantea desafíos relacionados con la propiedad intelectual y los derechos de autor. Muchos de estos datos están protegidos por derechos de autor, y su uso sin permiso o compensación adecuada puede ser problemático. El AI Act en Europa aborda este tema imponiendo nuevos requisitos a los desarrolladores de LLM, como la obligación de revelar las fuentes de datos utilizadas y obtener las licencias necesarias.

- Calidad y representatividad de los datos[44]: como cualquier modelo, un LLM será tan bueno como los datos utilizados en su entrenamiento. Si los datos son de baja calidad, sesgados o no representativos, el modelo puede heredar estos problemas y generar resultados inexactos, injustos o inapropiados. Por lo tanto, es crucial asegurar que los corpus de entrenamiento sean diversos, equilibrados y representen adecuadamente los diferentes grupos demográficos[457], opiniones y perspectivas.

- Iniciativas de datos de alta calidad[46]: algunas iniciativas recientes se centran en construir LLM con menos parámetros, pero datos de mayor calidad, como corpus de entrenamiento más pequeños, pero cuidadosamente seleccionados y filtrados[47], que incluyen contenido de alta calidad como libros, artículos científicos y publicaciones

[37] Mark Zuckerberg (n. 1984), cofundador y CEO de Facebook y de Meta, una de las mayores compañías de redes sociales, tecnología e inteligencia artificial del mundo.
[38] Liu (2024).
[39] Soskek (2019).
[40] Project Gutenberg (2024).
[41] Wikipedia Dumps (2024).
[42] Hugging Face Datasets (2024).
[43] Li (2024), Chu (2023).
[44] Alabdulmohsin (2024).
[45] Yogarajan (2023).
[46] Sachdeva (2024).
[47] Tirumala (2023).

Fig. 5. Etapas del procesamiento de datos de entrada a un LLM.



respetadas. Por ejemplo, estos filtros pueden limitarse a un único idioma, o a un sector o temática, lo que reduce drásticamente el tamaño del corpus. Esta estrategia puede resultar en LLM con mejor rendimiento y menos sesgos que los modelos entrenados en datos masivos no filtrados.

- Preprocesamiento y etiquetado de datos[48]: antes de entrenar o *fine-tunear* un LLM, los datos deben ser preprocesados y, en algunos casos, como el *fine-tuning* supervisado o el uso de un conjunto de datos específico, etiquetados. El preprocesamiento implica limpiar y formatear los datos[49], eliminar el ruido y los errores, y aplicar técnicas como la *tokenización* y la normalización (p. ej., LayerNorm[50] para *Transformers*).

Tokenización y encoding

La *tokenización* se refiere al proceso de dividir un texto en unidades más pequeñas llamadas "tokens", que son las unidades procesadas por el LLM durante el entrenamiento y la inferencia de respuestas. Estos tokens pueden ser palabras, partes de una palabra (p. ej., lemas) o caracteres. Por ejemplo, una de las formas más sencillas de generar tokens es separar el corpus según los espacios entre palabras. El *encoding* es el proceso de representar esas unidades de texto en forma numérica para que pueda ser procesado por el modelo.

Algunos puntos clave sobre la *tokenización* en los LLM:

- Se realiza partiendo del corpus de textos disponible, con objeto de dividir el texto de partida en unidades más pequeñas de forma óptima. El resultado final de la *tokenización* es un *encoding*.

- Los *encodings* impactan de forma significativa en el rendimiento del LLM[51], ya que definen la mínima unidad de procesamiento que van a recibir y determinan el vocabulario al que tiene acceso el LLM.

- Existen distintos algoritmos de *encoding* en el mercado[52] que se diferencian en la manera en la que dividen el texto en función de palabras, frases u oraciones, uso de espacios, mayúsculas o formatos, aparición de caracteres en distintos idiomas, o errores presentes en el texto.

- Los principales *encodings*[53] usados son *BytePairEncoding*, *SentencePieceEncoding* y *WordPieceEncoding*.

El resultado de la *tokenización* se emplea como punto de partida en el modelo de *embedding*.

Embedding

Los *embeddings* son representaciones numéricas de palabras, frases, oraciones o incluso párrafos que capturan su significado semántico y las relaciones entre ellos. Para ello, parten del corpus de entrada del LLM dividido en tokens. Son un componente fundamental de los LLM y desempeñan un papel crucial tanto en el preentrenamiento como en el *fine-tuning* y el uso posterior de estos modelos.

Los *embeddings* en los LLM:

- Están diseñados para capturar las relaciones semánticas entre las palabras, de manera que palabras con significados similares tengan vectores similares. Esto permite al modelo comprender la similitud y las analogías entre palabras y conceptos.

[48] Chen (2023).
[49] Wenzek (2019), Penedo (2023).
[50] Zhao (2023).
[51] Rejeleene (2024).
[52] Minaee (2024).
[53] Kudo (2018).

- No son valores universales, sino que varían entre distintos modelos en función del espacio vectorial en el que se hayan definido.

- Son contextuales, lo que significa que la representación de una palabra puede variar según el contexto en el que aparece. Esto permite capturar matices de significado y desambiguar palabras polisémicas. No están predefinidos, sino que se aprenden a partir de los datos de entrenamiento partiendo del modelo de *embeddings* del LLM. Durante el preentrenamiento, el modelo ajusta los *embeddings* para maximizar su capacidad de predecir palabras en contexto (p. ej., a través de marcos de *embeddings* como *SentenceTransformers*). No obstante, los *embeddings* por sí solos ya son un modelo que es necesario ajustar durante el proceso.

Preentrenamiento

El preentrenamiento es una etapa fundamental en el desarrollo de LLM, durante la cual los modelos adquieren un conocimiento general y profundo del lenguaje a partir de grandes cantidades de datos no etiquetados. Aunque este proceso es computacionalmente intensivo y costoso, permite la adaptación del modelo a una amplia gama de tareas.

El objetivo principal del preentrenamiento es que el modelo adquiera un conocimiento amplio y profundo del lenguaje, incluyendo su estructura, semántica, sintaxis y contexto. Durante este proceso, el LLM aprende a predecir palabras o fragmentos de texto (i.e., tokens) basándose en el contexto circundante, lo que le permite capturar relaciones y patrones lingüísticos complejos. Este conocimiento general se convierte en la base sobre la cual el modelo puede ser adaptado posteriormente para tareas específicas mediante el *fine-tuning*.

Existen varias técnicas populares para el preentrenamiento de LLM, como:

- El modelado autorregresivo de lenguaje o modelado unidireccional (p. ej., modelado autorregresivo[54]), que consiste en entrenar el modelo para predecir la siguiente palabra o fragmento de texto dado el contexto anterior. Esta tarea permite al modelo aprender las probabilidades condicionales del lenguaje y generar texto coherente. Son ejemplos los modelos GPT y Claude.

[54]Devlin (2018), Liu (2022).

Tipologías de *embeddings*

Los *embeddings* se utilizan en los LLM para poder establecer una métrica que defina la similitud entre los significados de las palabras y para incorporar información sobre la posición de las palabras en una oración. Esto es crucial, ya que el orden de las palabras afecta al significado. Existen tres tipos principales de *embeddings* posicionales:

- *Embedding* posicional absoluto[1]: asigna a cada palabra –o a cada unidad mínima de texto o *token*– un vector que representa su posición exacta en la oración (p. ej., primera, segunda, tercera posición, etc.).

- *Embedding* posicional relativo[2]: en lugar de basarse en posiciones absolutas, representa la posición de una palabra en relación con las demás (p. ej., dos palabras antes, una palabra después, etc.).

- *Embedding* posicional rotatorio[3]: combina la información de posiciones absolutas y relativas, utilizando funciones trigonométricas para crear representaciones vectoriales más complejas.

En un *transformer*, un *embedding* posicional simple para una palabra en una posición dada se puede representar matemáticamente usando funciones seno y coseno. En concreto, un *embedding* posicional E para un *token* i con posición P se puede representar matemáticamente en su forma más sencilla como:

$$E\,(P, 2i) = \sin\frac{P}{10000^{\frac{2i}{d}}}$$

$$E\,(P, 2i+1) = \cos\frac{P}{10000^{\frac{2i}{d}}}$$

donde P es la posición del *token* en la secuencia de entrada, y d es la dimensión de capas ocultas del *transformer*.

La elección del tipo de *embedding* posicional puede afectar al rendimiento del LLM, ya que determina la cantidad y el tipo de información posicional disponible para el modelo durante el entrenamiento.

[1]Vaswani (2017).
[2]Shaw (2018).
[3]Su (2021).

- El modelo no autorregresivo[55], usado en modelos como Gemini, en los que no se obtiene la respuesta secuencialmente palabra a palabra, sino que se transforma y refina en conjunto.

- El modelado de lenguaje enmascarado[56], popularizado por modelos como BERT, que consiste en enmascarar aleatoriamente algunas palabras en el texto de entrada y entrenar el modelo para predecir estas palabras enmascaradas basándose en el contexto circundante. Esta técnica permite un aprendizaje bidireccional y una mejor comprensión del contexto. Algunas arquitecturas de LLM (p. ej., *transformers* bidireccionales) usan esta técnica.

- El modelado secuencia a secuencia[57] (p. ej., seq2seq[58]), en el que el modelo se entrena para generar secuencias de texto en función de otras secuencias de entrada. Es usado en modelos como T5, BART o ProphetNET.

- El preentrenamiento contrastivo[59], utilizado en modelos como CLIP y ALIGN[60], que implica entrenar el modelo para identificar pares de texto e imagen que están semánticamente relacionados, lo que le permite aprender representaciones multimodales y transferir conocimiento entre diferentes modalidades[61].

El preentrenamiento de LLM es un proceso computacionalmente intensivo que requiere enormes cantidades de datos, tiempo y recursos de *hardware*. Los modelos más grandes pueden tener en torno a 1 billón (10^{12}) de parámetros y requerir miles de GPU de gama alta durante semanas o meses de entrenamiento. Esto hace que el preentrenamiento sea extremadamente costoso y solo esté al alcance de unas pocas empresas y organizaciones en el mundo con los recursos necesarios.

Cuantización

Durante el entrenamiento de los LLM, se ajustan los pesos de las neuronas para realizar predicciones más precisas. Estos pesos se almacenan típicamente como números de alta precisión, lo que puede resultar en modelos de gran tamaño y computacionalmente costosos.

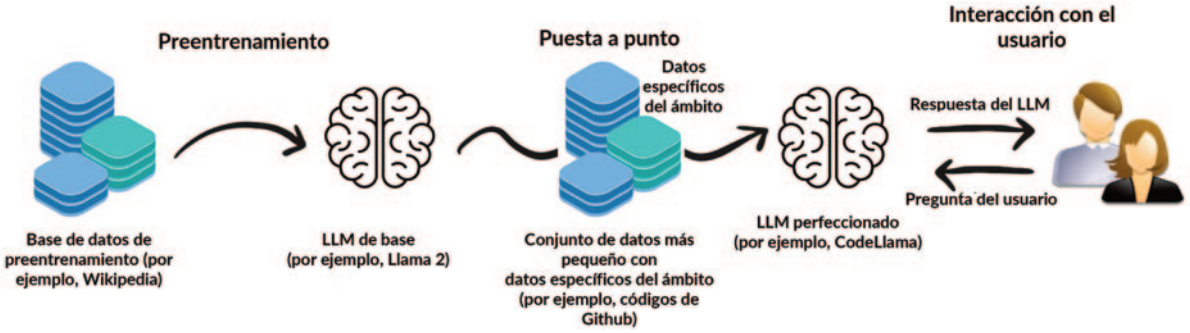
La *cuantización* postentrenamiento es una técnica[62] que permite reducir la precisión de los parámetros del modelo sin afectar significativamente el rendimiento del modelo. Por ejemplo, redes neuronales que almacenen sus parámetros usando números de coma flotante de 32 bits pueden pasar a usar únicamente 16 bits u 8 bits dependiendo del tipo de *cuantización*. Esto resulta en modelos más pequeños y rápidos, ya que requieren menos memoria y pueden realizar operaciones más eficientemente con el hardware adecuado.

Recientemente, ha surgido la tendencia de desarrollar modelos de menor tamaño (*small language models*, SLM), o incluso los llamados "*tiny* LLM"[63], modelos que mantienen un alto rendimiento a pesar de su tamaño mucho más reducido. Estos modelos compactos se logran mediante una combinación de técnicas, entre ellas la *cuantización* postentrenamiento.

Mediante la aplicación hábil de estas técnicas, los SLM y los tiny LLM están logrando en algunos casos un rendimiento comparable al de modelos mucho mayores[64], lo que los hace atractivos para aplicaciones con restricciones de recursos computacionales o de memoria.

[55] Xu (2021).
[56] Devlin (2019), Sinha (2021).
[57] Lee (2022).
[58] Sutskever (2014).
[59] Zeng (2023).
[60] Jia (2021).
[61] Cui (2022).
[62] Li (2024).
[63] Tian (2024).
[64] Fu (2024).

Fig. 6. Fine-tuning de un LLM.



Fine-tuning, instruction-tuning y RAG

El *fine-tuning* es el proceso de adaptar un LLM preentrenado a una tarea específica utilizando un conjunto de datos más pequeño. Esta técnica permite aprovechar el conocimiento general adquirido durante el preentrenamiento y especializarlo para obtener un alto rendimiento en la tarea objetivo.

El objetivo principal del *fine-tuning* (Fig. 6) es adaptar un LLM preentrenado a una tarea concreta, como la clasificación de sentimientos, la respuesta a preguntas, la traducción automática o la generación de resúmenes. Durante este proceso, el modelo aprende a utilizar su conocimiento general del lenguaje y aplicarlo de manera efectiva al dominio y los requisitos específicos de la tarea en cuestión. Los LLM disponibles en el mercado, sean propietarios o de código abierto, típicamente están preentrenados (y por tanto son de propósito general), pero no han recibido *fine-tuning*, que los adaptaría a un propósito específico.

El *fine-tuning* ofrece varios beneficios significativos:

- **Aprovecha el conocimiento previo:** al partir de un modelo preentrenado, el *fine-tuning* permite aprovechar el vasto conocimiento general del lenguaje adquirido durante el preentrenamiento, lo que acelera el aprendizaje y mejora el rendimiento en la tarea específica.
- **Requiere menos datos y recursos:** en comparación con el entrenamiento desde cero, el *fine-tuning* necesita mucha menos cantidad de datos etiquetados y recursos computacionales, lo que lo hace más accesible y económico para una amplia gama de organizaciones y aplicaciones.
- **Permite la especialización:** el *fine-tuning* permite adaptar los LLM a dominios y tareas concretas, lo que resulta en modelos altamente especializados y efectivos para aplicaciones específicas.
- **Facilita la transferencia de aprendizaje:** los modelos *fine-tuned* pueden recibir un *fine-tuning* adicional para tareas relacionadas, lo que permite la transferencia de aprendizaje y la creación de modelos aún más especializados con relativamente pocos datos adicionales.

A pesar de sus beneficios, el *fine-tuning* también presenta algunos desafíos:

- **Sobre-especialización**[65]: si el modelo se somete a un *fine-tuning* en un conjunto de datos demasiado específico, puede perder parte de su capacidad de generalización y funcionar mal con datos desconocidos o ligeramente diferentes.

[65] Wang (2024).

Entrenando LLM: funciones de pérdida

Los LLM, como otros modelos de aprendizaje profundo, aprenden ajustando sus parámetros para minimizar una función de pérdida. Esta función mide la diferencia entre las predicciones del modelo y los resultados esperados, guiando al modelo hacia un mejor rendimiento.

La elección de la función de pérdida depende del tipo de tarea para la que se esté entrenando el LLM. Por ejemplo, para un modelo que predice la siguiente palabra en una frase (modelado autorregresivo del lenguaje), una función común es la entropía cruzada. Esta función compara la distribución de probabilidad de las palabras predichas por el modelo con la distribución real observada en los datos de entrenamiento.

Matemáticamente, la función de pérdida de entropía cruzada para un modelo autorregresivo se puede expresar como una suma de los logaritmos negativos de las probabilidades asignadas a las palabras correctas en cada posición de la secuencia.

En concreto, dada una función de pérdida como la entropía cruzada, y una tipología de entrenamiento como el modelado autorregresivo del lenguaje, se puede definir la función de pérdida a minimizar como:

$$f_L(\varphi) = \sum_{i=1}^{N} -\log P(x_i \mid x_i, 1 \ldots N, \varphi)$$

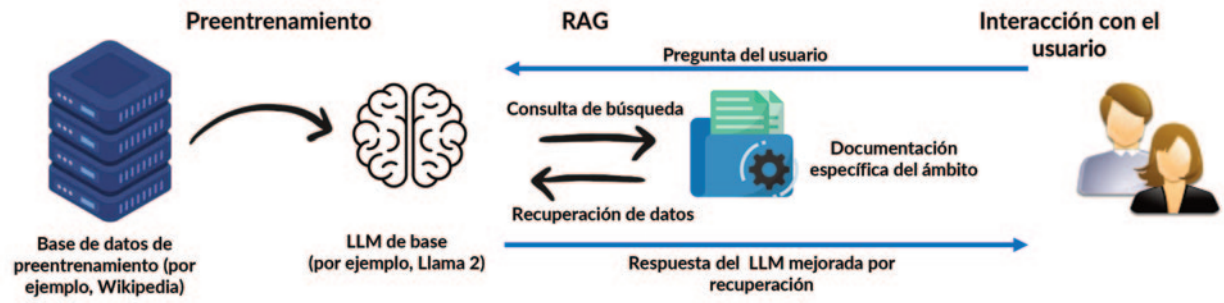
donde φ representa los parámetros del modelo, i se refiere al número de token en una secuencia determinada con N tokens, P es la probabilidad de predecir el token i en función de la secuencia x de tokens anteriores.

Durante el ajuste fino (fine-tuning) de los embeddings del modelo, se pueden utilizar funciones de pérdida especializadas para afinar las representaciones vectoriales de las palabras. Algunas opciones populares incluyen:

- Pérdida por similitud del coseno: ajusta los embeddings para que palabras similares tengan vectores más parecidos.
- Pérdida por error cuadrático medio: minimiza la diferencia cuadrática entre los embeddings predichos y los esperados.
- Pérdida por ranking de múltiples negativos: asocia los embeddings de palabras relacionadas de manera que estén más cerca entre sí que los de palabras no relacionadas.
- Pérdida por tripletes, de Matryoshka o contrastiva: variantes más avanzadas que consideran relaciones entre tríos o grupos de embeddings.

La selección cuidadosa de la función de pérdida es crucial para entrenar LLM efectivos y eficientes que puedan capturar los matices del lenguaje natural.

Fig. 7. Funcionamiento del RAG.



- **Olvido catastrófico**[66]: durante el *fine-tuning* es posible que un modelo olvide conocimientos críticos aprendidos previamente.

- **Inestabilidad**[67]: el proceso de *fine-tuning* puede ser sensible a factores como la inicialización de pesos, los hiperparámetros y la selección de datos, lo que puede llevar a resultados inconsistentes o variaciones en el rendimiento.

- **Herencia de sesgos**[68]: los modelos que han recibido *fine-tuning* pueden heredar y amplificar los sesgos presentes tanto en los datos de preentrenamiento como en los datos de *fine-tuning*, lo que requiere una cuidadosa consideración y mitigación.

Existen varios tipos de *fine-tuning* que deben seleccionarse en función de cuánto sea necesario modificar el modelo de partida para adecuarlo a una tarea en un dominio más específico. Los principales métodos son:

- ***Fine-tuning* supervisado**[69]: requiere conjuntos de datos etiquetados de entrada y respuesta del LLM, que se usan para mejorar su respuesta ante tareas específicas. Un método popular de *fine-tuning* supervisado es el llamado *instruction-tuning*[70], que consiste en alinear las respuestas del modelo a lo esperado por sus usuarios a través de interacciones con el modelo.

- *Reinforcement learning*: métodos basados en aprendizaje por refuerzo que se centran en mejorar la calidad de la respuesta del LLM, en este caso con base en el *feedback* de usuario o modelos de recompensa (p. ej., optimización directa por preferencia[71]).

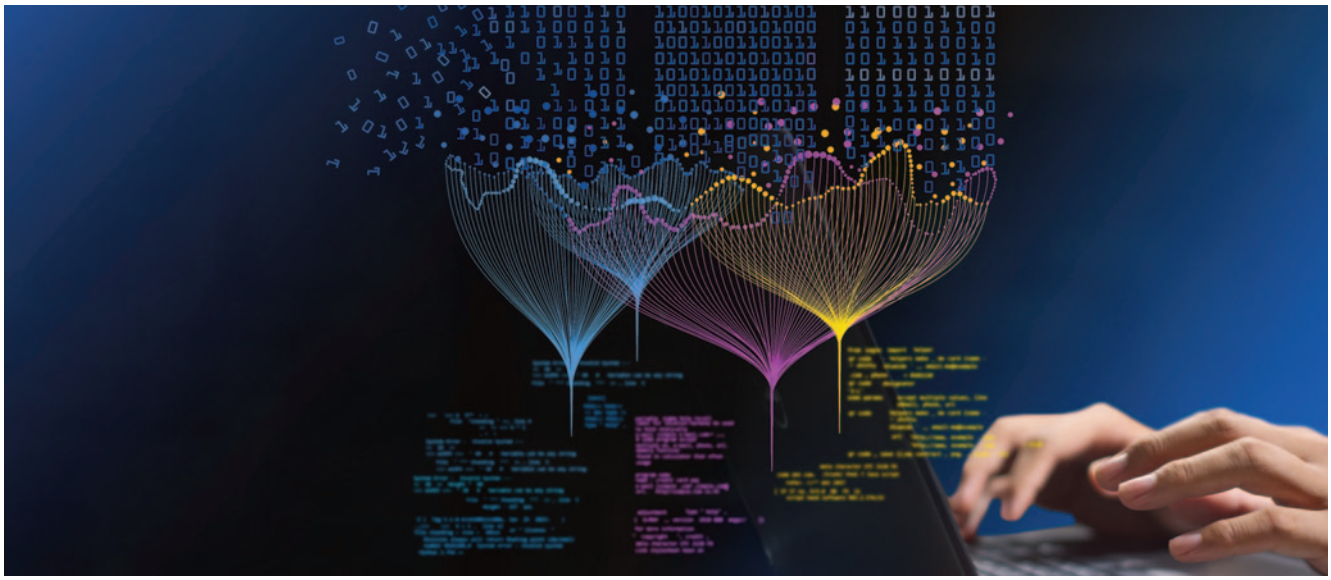
- *Fine-tuning* no supervisado[72]: se trata de un método que no requiere conjuntos de datos etiquetados, sino que se basa en reentrenar el modelo con las mismas metodologías usadas durante el preentrenamiento (p. ej., predecir el siguiente token).

- ***Parameter efficient*[73] *fine-tuning* (PEFT):** otros métodos de *fine-tuning* buscan aumentar su eficiencia y reducir el esfuerzo necesario para reentrenar el modelo. Por ejemplo, las técnicas basadas en LoRA[74] (*low-rank adaptation*) como QLoRA o LongLoRA[75], que permiten hacer *fine-tuning* del modelo sin la necesidad de modificar sus pesos y almacenan el conocimiento aprendido durante el proceso de *fine-tuning* en parámetros adicionales del modelo.

En muchos casos de uso de LLM, no es necesario emplear un *fine-tuning* para mejorar sus capacidades en un dominio específico. La generación aumentada de recuperación[76] (RAG) es una técnica que mejora el rendimiento del LLM a través del uso de fuentes de conocimiento externas al modelo.

Las técnicas de RAG (Fig. 7, funcionan buscando documentos dentro de una base de datos que se parezcan o refieran al *prompt* de entrada. Esta búsqueda y sus resultados se añaden a la generación de la respuesta del LLM para enriquecerla, proporcionando un contexto específico.

[66] Luo (2024).
[67] Zhang (2024).
[68] Zhang (2024).
[69] Ovadia (2024).
[70] Zhang (2023).
[71] Rafailov (2023).
[72] Zhou (2023).
[73] Xu (2023).
[74] Dettmers (2023).
[75] Chen (2023).
[76] Lewis (2020) y Neelakantan (2022).



Implementación y uso

Una vez entrenado y validado, el LLM debe desplegarse en un entorno de producción para su uso en aplicaciones reales. Esto conlleva la integración del modelo en sistemas y flujos de trabajo existentes, así como la creación de interfaces y APIs para interactuar con él.

Este proceso implica varios aspectos clave, entre los que se incluyen aspectos de integración y de monitoreo.

Integración en sistemas y flujos de trabajo

- **Infraestructura**[77]: los LLM suelen ser modelos grandes y computacionalmente intensivos, por lo que requieren una infraestructura robusta para su implementación. Esto puede implicar el uso de hardware especializado, como GPU o TPU, y plataformas de computación en la nube optimizadas para realizar eficientemente el proceso de inferencia.

- **Interfaces y API**[78]: para facilitar el uso del LLM en aplicaciones y servicios, es necesario desarrollar interfaces y API que permitan a otros sistemas interactuar con el modelo de manera eficiente y segura. Esto puede incluir *endpoints*, bibliotecas de cliente en varios lenguajes de programación e interfaces gráficas de usuario para usuarios no técnicos.

- **Integración con otros componentes:** en muchos casos, los LLM forman parte de un sistema más amplio que incluye otros componentes, como bases de datos, servicios de procesamiento de lenguaje natural y aplicaciones de usuario final. La integración fluida y eficiente del LLM con estos componentes es crucial para garantizar un rendimiento y una experiencia de usuario óptimos.

Monitoreo y mantenimiento

- **Monitoreo del rendimiento**[79]: una vez implementado, es esencial monitorear de cerca el rendimiento del LLM en condiciones reales. Esto implica realizar un seguimiento de métricas como la latencia, el rendimiento, la precisión y el uso de recursos, y establecer umbrales en el consumo de recursos y costes, y alertas para detectar y abordar cualquier degradación o anomalía.

- **Actualización y reentrenamiento**[80]: a medida que se dispone de nuevos datos o se identifican áreas de mejora, puede ser necesario actualizar o reentrenar el LLM. Esto requiere un proceso bien definido para recopilar y preparar nuevos datos, realizar un *fine-tuning*, y desplegar la versión actualizada del modelo sin interrupciones en el servicio.

- **Gestión de versiones**[81]: con las actualizaciones y mejoras continuas, es importante mantener un control de versiones riguroso del LLM y sus componentes asociados. Esto facilita la reproducibilidad, el *debugging* y la capacidad de revertir a versiones anteriores si es necesario.

Como se puede apreciar, el desarrollo y despliegue de LLM es un proceso complejo y multifacético que requiere una cuidadosa consideración de múltiples aspectos, desde la selección y preparación de los datos hasta la implementación y el uso responsable del modelo. Una comprensión profunda de los componentes clave, como el preentrenamiento, el *fine-tuning* y los *embeddings*, así como la conciencia de los desafíos y riesgos asociados, es esencial para aprovechar todo el potencial de los LLM de manera ética, sostenible y rentable, así como alineada con los objetivos de cada organización.

[77] Wan (2024).
[78] Abhyankar (2024).
[79] Goyal (2024).
[80] Lester (2021).
[81] Banerjee (2023).

Arquitectura de los LLM

La arquitectura de los LLM se refiere a la estructura y organización de las redes neuronales que componen estos modelos. La elección de la arquitectura y sus componentes tiene un impacto significativo en el rendimiento, la eficiencia y las capacidades del LLM. Esta sección explorará las principales arquitecturas utilizadas en los LLM y sus características, ventajas y limitaciones.

Transformers: el estado del arte en LLM

Los *transformers*, introducidos[82] en 2017, se han convertido en la arquitectura dominante para los LLM. A diferencia de las arquitecturas anteriores basadas en redes neuronales recurrentes (RNN) o redes neuronales convolucionales (CNN), los *transformers* se basan únicamente en mecanismos de atención para procesar y generar secuencias de texto (Fig. 8).

La arquitectura del *transformer* consta de dos componentes principales: el codificador (*encoder*) y el decodificador (*decoder*), y existen *transformers* con solo codificador, solo decodificador o con ambos componentes. El codificador procesa la secuencia de entrada y genera una representación contextual para cada token, mientras que el decodificador genera la secuencia de salida a partir de la representación del codificador y las predicciones anteriores.

La clave de los *transformers* es el mecanismo de atención, que permite al modelo poner atención en diferentes partes de la secuencia de entrada (atención del codificador) y en las predicciones anteriores (atención del decodificador) para generar la siguiente palabra o token. Esto permite capturar dependencias a largo plazo y generar secuencias coherentes.

Los *transformers* también introducen el concepto de atención multi-cabezal (*multi-head attention*), donde múltiples mecanismos de atención operan en paralelo, lo que permite al modelo capturar diferentes tipos de relaciones y patrones en los datos.

La arquitectura del *transformer* ha demostrado un rendimiento excepcional en una amplia gama de tareas de procesamiento del lenguaje natural, y ha sido adoptada por la mayoría de los LLM considerados de referencia.

Variantes y extensiones de los transformers

Desde la introducción de los *transformers*, se han propuesto numerosas variantes y extensiones para mejorar su eficiencia, escalabilidad y capacidad de modelado.

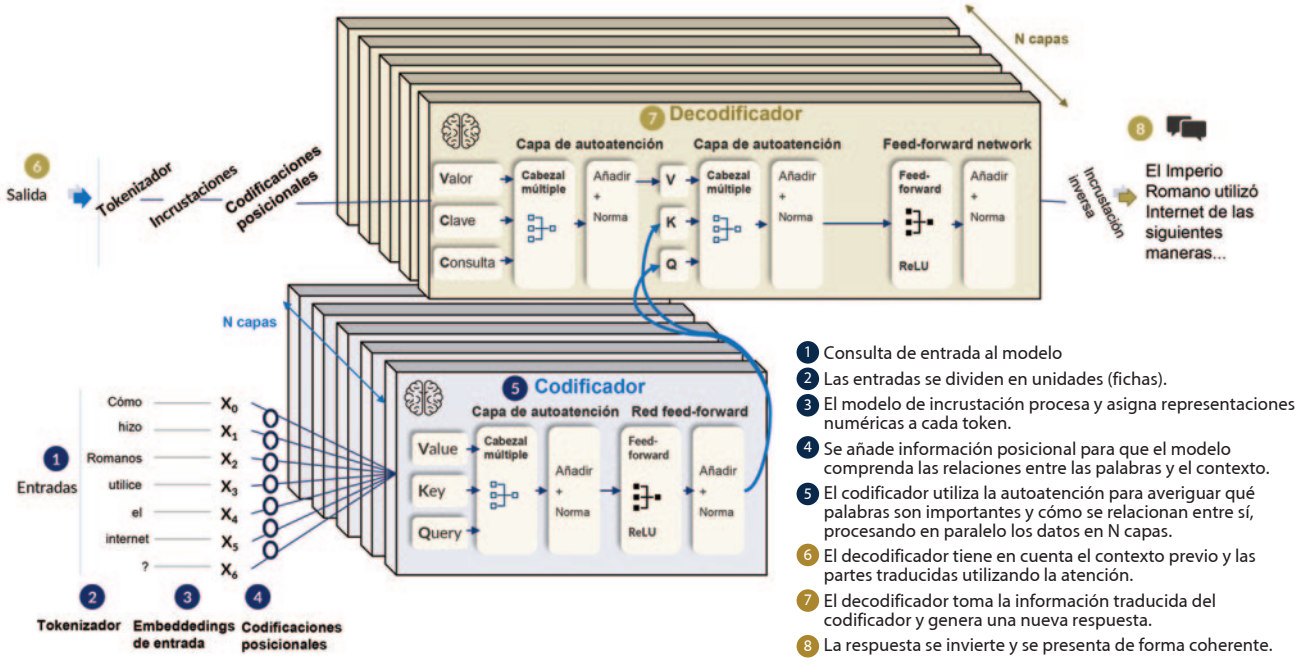
- Una variante popular es el *transformer* bidireccional, que permite al modelo atender tanto al contexto izquierdo como al derecho de cada token. Esto se logra mediante el uso de un objetivo de preentrenamiento de modelado de lenguaje enmascarado (MLM), donde algunos tokens se enmascaran aleatoriamente y el modelo debe predecirlos basándose en el contexto circundante.

- Otra variante es el *transformer* generativo, como GPT, que utiliza un enfoque de modelado de lenguaje unidireccional. Esto permite generar texto de manera eficiente y coherente, ya que el modelo solo puede atender al contexto izquierdo de cada token.

- También se han propuesto extensiones para hacer que los *Transformers* sean más eficientes y escalables, como el *transformer* disperso (*sparse transformer*), que utiliza

[82] Vaswani (2017).

Fig. 8. Funcionamiento de un transformer.



Prompt engineering en los LLM: principios y mejores prácticas

Prompt engineering se refiere al proceso de diseñar y optimizar los *prompts* (entradas de texto) para obtener los mejores resultados posibles de los LLM. Esta disciplina emergente contiene una serie de principios y mejores prácticas que permiten aprovechar al máximo las capacidades de estos modelos. Entre ellos:

- Ser claro y específico: las instrucciones dadas al modelo deben indicar de forma explícita el formato, la longitud y el nivel de detalle esperado en la respuesta. Por ejemplo, en lugar de simplemente pedir "Analiza la situación financiera de la empresa X", es mejor dar una instrucción como "Escribe un informe de 1000 palabras sobre la situación financiera de la empresa X, cubriendo su rentabilidad, liquidez, solvencia y perspectivas futuras".
- Descomponer tareas complejas: conviene dividir los problemas en subtareas más manejables para los LLM. Por ejemplo, en lugar de pedir "Desarrolla un plan estratégico para la empresa Y", se pueden solicitar subtareas como "Realiza un análisis DAFO de la empresa Y", "Define los objetivos estratégicos clave para Y", "Propón iniciativas para alcanzar cada objetivo", etc.
- Proporcionar ejemplos ilustrativos (*few-shot learning*): unos pocos ejemplos bien elegidos pueden ayudar enormemente a comunicar la tarea deseada. Supongamos que se quiere generar propuestas de valor para productos; se podrían dar dos ejemplos: "Nuestro software CRM permite a los equipos de ventas cerrar acuerdos un 50% más rápido" y "Nuestra *app* de bienestar ayuda a los empleados a reducir el estrés y aumentar su productividad en un 25%".
- Solicitar razonamiento paso a paso: instruir al LLM para que verbalice su proceso de pensamiento a menudo conduce a resultados más robustos. Esto es especialmente útil para tareas de análisis o solución de problemas en el ámbito empresarial. Por ejemplo: "Describe paso a paso cómo calcularías el ROI de este proyecto de inversión".
- Solicitar las referencias empleadas: instruir al LLM para que indique en su proceso de razonamiento las referencias a documentos que ha empleado, incluyendo citas del texto original al que tiene acceso.
- Pedir al LLM que adopte un personaje: antes de la tarea principal, se puede primero instruir al modelo para que adopte un determinado rol, tono o estilo. Por ejemplo: "Actúa como un experto analista financiero y proporciona una valoración objetiva de la empresa X". Esto ayuda a orientar su comportamiento.
- Aprovechar conocimiento externo: proporcionar información adicional permite complementar la base de conocimientos del LLM. Por ejemplo, para responder preguntas sobre una industria específica, se podrían primero recuperar informes sectoriales relevantes y pasarlos al modelo.
- Iterar y refinar sistemáticamente: la evaluación continua del rendimiento del modelo permite identificar áreas de mejora y ajustar los *prompts* en consecuencia. Métricas cuantitativas y juicios cualitativos de expertos del dominio pueden guiar este proceso iterativo.

Mediante la aplicación de estos principios de *prompt* engineering, se demuestra estadísticamente que los LLM proporcionan un resultado más preciso y fiable.

Considerando todo esto, un mal *prompt* para que un LLM escriba una columna sobre *prompt* engineering sería: "Escribe un artículo sobre *prompt* engineering".

Y un buen *prompt* para escribir esa columna sería:

> "Actúa como un experto en inteligencia artificial y escribe una columna de divulgación de 600 palabras sobre los principios clave del *prompt* engineering para obtener los mejores resultados de los LLM. Estructura la columna con una introducción breve y atractiva, 4-5 párrafos cubriendo los puntos principales (ser específico, descomponer tareas, dar ejemplos...), y una conclusión con los beneficios de aplicar estas técnicas. Utiliza un tono divulgativo pero riguroso, apto para un público empresarial. Incluye ejemplos concretos para ilustrar las ideas".

Fuentes: Guía de *prompt* engineering de OpenAI[1], soporte de Anthropic Claude Opus y elaboración propia.

[1]OpenAI (2024).

atención dispersa para reducir la complejidad computacional, y el *transformer* comprimido (*compressed transformer*), que utiliza técnicas de compresión para reducir el tamaño del modelo.

Comparación con arquitecturas anteriores

Antes de los *transformers*, las arquitecturas dominantes para el modelado de secuencias eran las redes neuronales recurrentes (RNN), como *Long Short-Term Memory* (LSTM) y *Gated Recurrent Unit* (GRU), y las redes neuronales convolucionales (CNN).

- Las RNN pueden capturar dependencias a largo plazo en las secuencias, pero sufren de problemas como el desvanecimiento del gradiente (*vanishing gradient*) y la dificultad para paralelizar el entrenamiento. Además, las RNN tienen dificultades para capturar dependencias muy largas debido a su naturaleza secuencial y al uso de recurrencias de alcance constante.

- Las CNN pueden capturar patrones locales en las secuencias y son eficientes en términos de computación, pero tienen dificultades para modelar dependencias a largo plazo y requieren un tamaño de contexto fijo.

En comparación, los *transformers* superan estas limitaciones al utilizar mecanismos de atención que pueden capturar dependencias a largo plazo de manera eficiente y paralela. Además, los *transformers* son más flexibles en términos de manejo de secuencias de longitud variable y pueden ser preentrenados en grandes cantidades de datos no etiquetados.

La arquitectura de los *transformers* ha revolucionado el campo de los LLM y ha permitido avances significativos en una amplia gama de tareas de procesamiento del lenguaje natural. Sin embargo, aún quedan desafíos por abordar, como la escalabilidad, la interpretabilidad y la eficiencia de estos modelos. A medida que la investigación avance, es probable que aparezcan nuevas arquitecturas y técnicas que superen estas limitaciones y lleven a los LLM a nuevas cotas de rendimiento y capacidad.

LLMOps

MLOps *(Machine Learning Operations)* es una metodología y conjunto de prácticas diseñadas para gestionar el ciclo de vida completo de los modelos de *machine learning,* desde el desarrollo y entrenamiento hasta la implementación y mantenimiento en producción.

En los últimos años, ha surgido una adaptación de la metodología MLOps específicamente orientada a los LLM, conocida como LLMOps *(Large Language Model Operations)*. Esta disciplina se centra en gestionar de manera eficiente el ciclo de vida completo de los LLM, abarcando desde su desarrollo y entrenamiento hasta su implementación y mantenimiento en entornos de producción.

LLMOps integra los procesos tradicionales de desarrollo de software con herramientas y técnicas diseñadas para abordar los desafíos únicos que presentan los modelos de lenguaje de gran escala. Algunos de estos desafíos incluyen:

- **Gestión de grandes volúmenes de datos:** los LLM requieren cantidades masivas de datos de entrenamiento, lo que implica la necesidad de infraestructuras de almacenamiento y procesamiento escalables y eficientes.

- **Escalado de recursos computacionales:** el entrenamiento y la inferencia de LLM demandan enormes recursos de cómputo, haciendo necesario el uso de técnicas de

paralelización y distribución, así como la optimización del uso de hardware especializado como GPU y TPU.

- **Monitorización y mantenimiento:** una vez desplegados en producción, los LLM deben ser monitorizados de cerca para detectar y corregir problemas de rendimiento, sesgos, riesgos como las alucinaciones, y la degradación del modelo a lo largo del tiempo.

- **Versionado y reproducibilidad:** dado el tamaño y complejidad de los LLM, es crucial mantener un riguroso control de versiones y maximizar la reproducibilidad de los experimentos y resultados.

Para abordar estos desafíos, LLMOps se apoya en una serie de herramientas y *frameworks* específicos, como MLFlow[83], CometML[84] y Weights & Biases[85]. Estas plataformas ofrecen funcionalidades para el seguimiento de experimentos, gestión de modelos, monitorización del rendimiento y colaboración entre equipos.

Además, LLMOps promueve prácticas como la automatización de procesos, el *testing* continuo, la documentación exhaustiva y la gobernanza de modelos. Esto permite no solo mejorar la eficiencia y calidad del desarrollo de los LLM, sino también garantizar su uso ético y responsable.

Retos

El desarrollo y despliegue de LLM presenta una serie de retos significativos que deben abordarse para garantizar su uso responsable, ético y seguro. Esta sección explorará varios de los principales desafíos para las organizaciones en su despliegue y uso de los LLM.

Sesgos, alucinaciones y confiabilidad

Uno de los mayores retos de los LLM es la presencia de sesgos y alucinaciones en sus resultados y predicciones. Los sesgos pueden surgir de varias fuentes, como la parcialidad en los datos de entrenamiento, las limitaciones de las arquitecturas de los modelos o los prejuicios humanos implícitos en las tareas de anotación y evaluación. Por otro lado, las alucinaciones se refieren a la generación de información o contenido que parece plausible pero que no se basa en hechos reales o en el conocimiento adquirido durante el entrenamiento.

Los sesgos en los LLM pueden manifestarse de diversas maneras, como la perpetuación de estereotipos de género, raza o edad, la discriminación en tareas de clasificación o la generación de contenido ofensivo o inapropiado. Estos sesgos pueden tener consecuencias graves, especialmente cuando los LLM se utilizan en aplicaciones sensibles como la toma de decisiones en el ámbito legal, financiero o médico. Por su parte, las alucinaciones pueden llevar a la difusión de información errónea o engañosa, lo que puede tener un impacto negativo en la confianza de los usuarios y en la credibilidad de las aplicaciones basadas en LLM.

Para abordar el reto de los sesgos, es necesario desarrollar técnicas robustas para detectar, medir y mitigar su presencia en los LLM. Esto implica la creación de conjuntos de datos de evaluación específicos para sesgos, el uso de métricas de equidad y la aplicación de técnicas de eliminación de sesgos (*debiasing*) tanto en el preentrenamiento como en el *fine-tuning*. Además, es crucial establecer procesos de auditoría y monitoreo continuo para garantizar que los LLM sigan siendo imparciales a lo largo del tiempo.

[83] Zaharia (2018).
[84] CommetML: https://www.comet.com/
[85] Weights and biases: https://wandb.ai/site

Para abordar las alucinaciones en los LLM, se están desarrollando diversos métodos centrados en la mejora de los datos de entrenamiento, la aplicación de técnicas de regularización robustas y el uso de retroalimentación humana para ajustar las respuestas del modelo. Además, se están investigando cambios arquitectónicos en los modelos para hacerlos inherentemente menos propensos a alucinar. Los métodos de generación de texto y el contexto de entrada también pueden ser optimizados para reducir las alucinaciones. La supervisión humana y la evaluación rigurosa son esenciales para detectar y corregir información inexacta. Asimismo, el desarrollo de herramientas específicas, como modelos de evaluación de alucinaciones y técnicas de ofuscación, puede contribuir a mejorar la precisión de los LLM.

Explicabilidad y accountability

Otro gran desafío de los LLM es su opacidad y falta de explicabilidad. Debido a su complejidad y a la naturaleza de sus arquitecturas, es difícil entender cómo estos modelos llegan a sus resultados.

Esta falta de transparencia plantea problemas de *accountability*, especialmente cuando los LLM se utilizan en contextos de alta sensibilidad, donde las decisiones tienen un impacto significativo en las personas (p. ej., uso de LLM en medicina, investigación farmacéutica, infraestructura crítica o acceso al mercado laboral). Sin una comprensión clara de cómo funcionan estos modelos, es difícil determinar la responsabilidad en caso de errores o comportamientos no deseados.

Para abordar este reto, es necesario desarrollar técnicas y herramientas que permitan una mayor interpretabilidad y explicabilidad de los LLM. Esto incluye métodos para visualizar y analizar los mecanismos internos de atención, técnicas de atribución para identificar las partes más relevantes de la entrada, y enfoques para generar explicaciones en lenguaje natural de las predicciones del modelo.

Además, es importante establecer marcos de *accountability* claros que definan las responsabilidades de los desarrolladores, implementadores y usuarios de los LLM, como en Europa propone el AI Act. Esto puede implicar la creación de estándares y directrices para el desarrollo ético de los LLM, mecanismos de supervisión y auditoría externa, y canales para que las partes interesadas planteen inquietudes.

Confidencialidad y protección de la información

Los LLM a menudo se entrenan con grandes cantidades de datos que pueden contener información personal, sensible o confidencial. Además, cuando se despliegan en aplicaciones del mundo real, estos modelos pueden estar expuestos a entradas de usuario que también pueden incluir datos privados.

Esto plantea importantes desafíos de privacidad y seguridad, ya que los LLM pueden memorizar y reproducir información confidencial de sus datos de entrenamiento, o ser vulnerables a ataques que intenten extraer datos privados a través de consultas cuidadosamente diseñadas.

Para abordar este reto, es necesario desarrollar técnicas de preservación de la privacidad en el entrenamiento y despliegue de LLM (p. ej., Digger[86] para detectar información protegida, el uso de datos ficticios[87] durante el entrenamiento para detectar el material con copyright).

Además, es crucial establecer protocolos robustos de seguridad y control de acceso para proteger los LLM y sus datos asociados de accesos no autorizados o usos maliciosos. Esto puede implicar el uso de técnicas de autenticación y autorización, monitoreo de seguridad y detección de anomalías.

[86] Li (2024).
[87] Meeus (2024).

Consumo racional de los recursos

El entrenamiento y despliegue de LLM requiere cantidades masivas de recursos computacionales, almacenamiento y energía. Con modelos que alcanzan los cientos de miles de millones o incluso los billones de parámetros, el coste financiero y ambiental de desarrollar y operar estos sistemas puede ser muy significativo[88].

Este alto consumo de recursos plantea desafíos de eficiencia, escalabilidad y sostenibilidad. A medida que la demanda de LLM más grandes y potentes sigue creciendo, es necesario encontrar formas de optimizar su rendimiento y reducir su huella de recursos.

Para abordar este reto, se están explorando varias direcciones de investigación. Una de ellas es el diseño de arquitecturas de modelos más eficientes, como el uso de mecanismos de atención dispersa o técnicas de compresión que reducen el tamaño y la complejidad computacional de los LLM sin comprometer significativamente su rendimiento.

También se está investigando en la mejora de las técnicas de preentrenamiento continuo[89] y *fine-tuning* continuo[90], que buscan integrar la capacidad de usar información de diversos dominios sin la necesidad de depender de un reentrenamiento exhaustivo y costoso con nuevos datos específicos. Asimismo, se está avanzando en el uso de sistemas innovadores y el diseño de algoritmos verdes de IA, que permiten hacer frente a los costes computacionales y ambientales asociados a la IA (p. ej., el sistema GreenLightningAI, de Qsimov Quantum Computing[91], desarrolla el reentrenamiento incremental y proporciona una interpretabilidad directa).

Otra dirección es el desarrollo de infraestructuras y plataformas de computación más sostenibles, como el uso de hardware especializado de bajo consumo, sistemas de enfriamiento más eficientes y fuentes de energía renovables para alimentar los centros de datos donde se entrenan y despliegan los LLM.

Además, es importante promover prácticas de uso racional y compartido de los recursos, como la reutilización y adaptación de modelos preentrenados en lugar de entrenar nuevos modelos desde cero para cada tarea, y el intercambio de recursos y conocimientos entre organizaciones y comunidades de investigación.

Otros desafíos

De entre los muchos retos adicionales a los que se enfrentan las organizaciones en el desarrollo, implementación y uso de los LLM, por su importancia cabe mencionar brevemente:

- **Dependencia y lock-in:** las organizaciones que dependen de LLM proporcionados por terceros pueden enfrentarse a riesgos de dependencia y *lock-in,* especialmente si los modelos se basan en datos o infraestructura propietaria. Es importante considerar estrategias de diversificación y planes de contingencia.

- **Riesgos de seguridad y uso malicioso**[92]: los LLM pueden ser vulnerables a ataques adversarios, como la inyección de datos envenenados o la ingeniería inversa. Además, pueden utilizarse de forma maliciosa para generar desinformación, *spam* o contenido engañoso. Es esencial establecer medidas de seguridad robustas y diseñar los modelos con salvaguardias contra el uso indebido.

- **Cuestiones de propiedad intelectual y licencias:** el uso de LLM plantea preguntas sobre la propiedad intelectual y las licencias de los datos de entrenamiento, los modelos y los resultados generados. Adicionalmente, existe un riesgo de robo de información o datos personales de usuarios que lancen consultas a LLM desplegados en nubes de terceros. Es necesario cumplir con la regulación y con los marcos éticos para equilibrar los derechos de los creadores, los usuarios y el interés público, y, en el caso de las organizaciones, para evitar riesgos legales y de cumplimiento.

- **Escalabilidad de la arquitectura de los LLM**[93]: un desafío adicional es la escalabilidad de los *transformers* a medida que aumenta el tamaño de las secuencias y los modelos. Los mecanismos de atención tienen una complejidad cuadrática con respecto a la longitud de la secuencia, lo que limita su aplicación a secuencias muy largas.

[88] iDanae 1T24 (2024).
[89] Yıldız (2024).
[90] Mehta (2023).
[91] iDanae 1T24 (2024).
[92] Pankajakshan (2024).
[93] Rae (2021).