PNAV

PROGRAMA NACIONAL
DE ALGORITMOS
VERDES

Guía de buenas prácticas

Guía de datos y algoritmos sostenibles

Financiado por la Unión Europea
NextGenerationEU

MINISTERIO
PARA LA TRANSFORMACIÓN DIGITAL
Y DE LA FUNCIÓN PÚBLICA

Plan de Recuperación, Transformación y Resiliencia

ENIA Estrategia Nacional de Inteligencia Artificial

España | digital 2026

Financiado por la Unión Europea - NextGenerationEU. Sin embargo, los puntos de vista y las opiniones expresadas son únicamente los del autor o autores y no reflejan necesariamente los de la Unión Europea o la Comisión Europea. Ni la Unión Europea ni la Comisión Europea pueden ser consideradas responsables de las mismas

Índice

Introducción

El desarrollo y despliegue de modelos de inteligencia artificial conlleva un alto consumo energético y un impacto ambiental significativo. A medida que la IA avanza y se integra en más ámbitos, surge la necesidad de adoptar enfoques sostenibles que optimicen el uso de recursos sin comprometer el rendimiento.

Esta guía presenta estrategias y buenas prácticas para la sostenibilidad en IA, abarcando desde la obtención y procesamiento eficiente de datos hasta el diseño y reutilización de modelos optimizados. También se exploran casos de éxito y herramientas para evaluar el impacto energético de los modelos, facilitando la toma de decisiones informadas para minimizar su huella de carbono.

A través de esta guía, se busca fomentar el desarrollo de algoritmos más eficientes y responsables con el medio ambiente, alineando la innovación en IA con los principios de sostenibilidad y eficiencia energética.

1. Obtención, selección y tratamiento de conjuntos de datos

El uso eficiente de los datos es un factor clave en la sostenibilidad de los algoritmos de IA. La forma en que se recopilan, almacenan y procesan los datos impacta significativamente el consumo energético y la huella de carbono. Para minimizar el impacto ambiental, se deben considerar las siguientes prácticas a lo largo del ciclo de vida:

1.1 Minimización de datos innecesarios

Para optimizar el uso de los datos y reducir el consumo energético, es fundamental implementar estrategias que permitan seleccionar solo la información relevante y eliminar redundancias innecesarias. A continuación, se detallan las principales técnicas:

1.1.1 Filtrado de datos

El filtrado de datos es el proceso mediante el cual se eliminan registros irrelevantes o redundantes, mejorando la eficiencia del almacenamiento y procesamiento. Algunas técnicas incluyen:

- **Filtrado basado en reglas**: Se establecen criterios específicos para seleccionar los datos útiles y descartar el resto. Ejemplo: eliminación de datos fuera de un rango determinado.
- **Filtrado estadístico**: Uso de métricas como la media, la desviación estándar y los percentiles para identificar y eliminar valores atípicos o poco representativos.
- **Filtrado por correlación**: Se eliminan variables altamente correlacionadas para reducir la redundancia en modelos de aprendizaje automático.
- **Filtrado por varianza**: Se descartan características con baja variabilidad en el conjunto de datos, ya que aportan poca información útil al modelo.

- **Filtrado semántico**: Se utilizan técnicas de procesamiento de lenguaje natural (NLP) para filtrar textos irrelevantes en análisis de datos no estructurados.

1.1.2 Reducción de redundancia

Eliminar datos duplicados o redundantes ayuda a optimizar el almacenamiento y la eficiencia computacional. Algunas estrategias clave incluyen:

- **Detección y eliminación de duplicados**: Uso de herramientas como fuzzy matching para identificar registros similares que puedan representar la misma entidad.
- **Agrupación y normalización**: Consolidación de registros que contienen información repetida en múltiples fuentes.
- **Uso de técnicas de hashing**: Aplicación de funciones hash para identificar y eliminar duplicados con eficiencia computacional.
- **Reducción de características (Feature Selection)**: Eliminación de variables redundantes mediante técnicas como selección basada en importancia de características y algoritmos de selección secuencial.

1.1.3 Uso de técnicas de compresión

La compresión de datos permite reducir el tamaño del almacenamiento y la cantidad de datos transmitidos, disminuyendo el consumo energético. Algunas técnicas destacadas incluyen:

- **Compresión sin pérdida**: Se aplican algoritmos como Huffman coding, Run-Length Encoding (RLE) y Lempel-Ziv-Welch (LZW), que permiten reconstruir los datos originales sin pérdida de información.
- **Compresión con pérdida**: Se utilizan métodos como la Transformada Discreta de Fourier (DFT) o la Transformada Wavelet para reducir la resolución de imágenes y audios sin afectar significativamente la calidad perceptual.
- **Codificación de datos**: Técnicas como quantization y bit-packing reducen la representación de datos numéricos sin perder precisión significativa.
- **Almacenamiento eficiente en bases de datos**: Uso de formatos como Parquet y ORC, diseñados para reducir el almacenamiento y acelerar las consultas.
- **Compresión diferencial**: Utilizada en series temporales, permite almacenar solo las diferencias entre registros consecutivos en lugar de los valores completos.

La combinación de estas estrategias en la fase de obtención y tratamiento de datos permite una reducción significativa del consumo energético, asegurando al mismo tiempo la calidad y utilidad de la información utilizada en los modelos de IA.

1.2 Fuentes de datos sostenibles

La selección de fuentes de datos sostenibles contribuye a la reducción del impacto ambiental al disminuir la necesidad de recopilar, generar y transferir grandes volúmenes de datos. La

reutilización de datasets existentes evita procesos intensivos en energía asociados a la captura, limpieza y almacenamiento inicial de nuevos datos, mientras que la selección de fuentes de datos cercanas al entorno de procesamiento reduce el consumo energético derivado de la transmisión de información a través de redes.

- **Uso de datasets abiertos:** El uso de datasets abiertos y reutilizables permite aprovechar datos ya existentes, evitando la generación de nuevos conjuntos de datos, que suele implicar procesos costosos en términos de energía y recursos (captura, etiquetado, almacenamiento y mantenimiento). La reutilización de datos promueve la eficiencia del ciclo de vida del dato y reduce la huella ambiental asociada al desarrollo de modelos de IA.

1.3 Minimización de transferencia de datos

La selección de fuentes de datos locales o cercanas al entorno de cómputo reduce el volumen de datos transmitidos a través de redes, disminuyendo el consumo energético asociado a operaciones de transferencia, especialmente relevante en escenarios de grandes volúmenes de datos o procesamiento distribuido

Estrategias de preprocesamiento eficientes:

El preprocesamiento eficiente de datos no solo mejora la calidad del modelo, sino que también contribuye a la reducción del consumo energético. Algunas técnicas clave incluyen:

- **Normalización y estandarización**: Escalado de valores para evitar desbalances en el modelo y reducir la complejidad del entrenamiento.
- **Reducción de dimensionalidad**: Aplicar técnicas como PCA (Análisis de Componentes Principales), t-SNE o Autoencoders para reducir la cantidad de variables sin comprometer la calidad del modelo, optimizando el procesamiento de datos.
- **Transformaciones de datos optimizadas**: Uso de funciones matemáticas eficientes. Empleo de transformaciones y operaciones matemáticas optimizadas que reducen el número de cálculos necesarios durante el entrenamiento y la inferencia. Algunos ejemplos incluyen el uso de operaciones vectorizadas frente a bucles iterativos, funciones de activación computacionalmente eficientes (como ReLU frente a funciones exponenciales), aproximaciones numéricas simplificadas y transformaciones lineales precomputadas. Estas prácticas permiten reducir el tiempo de cómputo y el consumo energético asociado al procesamiento de datos. Para convertir datos de manera que se reduzca el tiempo de cómputo en modelos de IA.
- **Uso de técnicas de balanceo de datos**: Aplicación de métodos como oversampling y undersampling para equilibrar clases sin generar grandes volúmenes de datos adicionales.

- **Procesamiento distribuido**: Implementación de frameworks para distribuir la carga de trabajo y mejorar la eficiencia energética en el preprocesamiento de datos a gran escala.

1.4 Almacenamiento energético eficiente

La eficiencia en el almacenamiento de datos permite reducir costos y consumo de energía, contribuyendo a un menor impacto ambiental. Algunas estrategias incluyen:

- **Uso de infraestructuras cloud sostenibles**: Elegir proveedores de almacenamiento en la nube que utilicen energía renovable y optimización de recursos.
- **Almacenamiento en bases de datos eficientes**: Implementar esquemas de indexación y modelos de almacenamiento optimizados como NoSQL, Parquet y ORC, que permiten consultas rápidas y reducen el uso de recursos.
- **Compresión de almacenamiento**: Uso de formatos optimizados como Gzip, Parquet u ORC con Zstandard en escenarios donde el volumen de datos almacenados o transferidos es elevado y el acceso es mayoritariamente secuencial o por lotes. En estos casos, la reducción del tamaño de los datos disminuye las operaciones de lectura/escritura en disco y la transferencia por red, lo que puede traducirse en un menor consumo energético total, especialmente cuando el coste energético de la compresión y descompresión es inferior al ahorro obtenido en almacenamiento y E/S.
- **Implementación de almacenamiento en caché**: Uso de soluciones como Redis o Memcached para reducir accesos a bases de datos y mejorar la eficiencia en la recuperación de datos.
- **Estrategias de eliminación y archivado**: Eliminación periódica de datos obsoletos y almacenamiento en frío para datos de baja frecuencia de acceso, reduciendo la carga en sistemas activos.

1.5 Datos sintéticos y aumentación de datos

La generación de datos sintéticos y la aumentación de datos son técnicas clave para reducir la dependencia de grandes volúmenes de datos reales, cuya recopilación, almacenamiento y preprocesamiento suelen ser intensivos en recursos. Al generar datos artificiales de forma controlada o reutilizar datos existentes mediante transformaciones, es posible entrenar modelos con conjuntos de datos más compactos o mejor informados, evitando la necesidad de recopilar y almacenar grandes cantidades de datos brutos. Esto puede reducir el número de iteraciones de entrenamiento necesarias y, en consecuencia, el consumo computacional y energético asociado al procesamiento de datos.

Estas técnicas permiten crear conjuntos de datos diversificados y representativos sin incurrir en los costes ambientales de la recolección de datos a gran escala, siempre que el coste computacional de la generación o aumentación de datos se vea compensado por la reducción del volumen de datos reales y del esfuerzo de entrenamiento posterior.

1.5.1 Generación de datos sintéticos

Los datos sintéticos son creados artificialmente mediante técnicas de simulación, modelos generativos o redes neuronales. Su uso es especialmente útil en entornos donde la recopilación de datos reales es costosa, poco ética o insostenible. Algunas técnicas comunes incluyen:

- **Redes Generativas Adversariales (GANs)**: Modelos que generan datos sintéticos indistinguibles de los reales, útiles para tareas como la generación de imágenes, texto o audio.
- **Simulación**: Uso de entornos virtuales para generar datos sintéticos, especialmente en aplicaciones como vehículos autónomos o robótica.
- **Modelos basados en reglas**: Creación de datos mediante reglas predefinidas, útiles en aplicaciones como la generación de transacciones financieras o datos médicos.

1.5.2 Aumentación de datos

La aumentación de datos consiste en aplicar transformaciones a los datos existentes para crear nuevas muestras, diversificando el conjunto de datos sin necesidad de recopilar información adicional. Algunas técnicas incluyen:

- **Aumentación de imágenes**: Rotación, escalado, cambio de brillo o contraste en imágenes para crear nuevas variantes.
- **Aumentación de texto**: Uso de sinónimos, reordenación de frases o traducción inversa para generar nuevas muestras de texto.
- **Aumentación de audio**: Cambio de tono, velocidad o adición de ruido de fondo en archivos de audio.

1.5.3 Beneficios en eficiencia energética

- **Reducción de la recopilación de datos**: Evita el consumo energético asociado a la recolección y almacenamiento de grandes volúmenes de datos reales.
- **Optimización del entrenamiento**: El uso de datos sintéticos y técnicas de aumentación puede contribuir a mejorar la capacidad de generalización del modelo cuando los datos generados son representativos y están correctamente alineados con el dominio del problema. En estos casos, una mejor generalización puede ayudar a estabilizar el proceso de entrenamiento y reducir la necesidad de ajustes reiterados o reentrenamientos extensivos. No obstante, este efecto no es automático y depende de la calidad de los datos sintéticos, del modelo empleado y del contexto de aplicación.
- **Ahorro de recursos**: Permite trabajar con conjuntos de datos más pequeños, pero igualmente efectivos, reduciendo el tiempo de cómputo y el consumo energético.

1.6 Selección activa de datos (Active Learning)

La selección activa de datos, o **Active Learning**, es una técnica que permite a los modelos de IA seleccionar de manera inteligente los datos más informativos para su entrenamiento, reduciendo la cantidad de datos necesarios y, por tanto, el consumo energético asociado al procesamiento.

1.6.1 ¿Cómo funciona el Active Learning?

En lugar de entrenar con todo el conjunto de datos disponible, el modelo identifica iterativamente las muestras más relevantes para mejorar su rendimiento. Este proceso se realiza en ciclos:

1. **Entrenamiento inicial**: El modelo se entrena con un pequeño subconjunto de datos.
2. **Selección de muestras**: El modelo identifica las muestras más útiles (por ejemplo, aquellas con mayor incertidumbre o error).
3. **Etiquetado y reentrenamiento**: Las muestras seleccionadas se etiquetan (manual o automáticamente) y se añaden al conjunto de entrenamiento.
4. **Repetición**: El proceso se repite hasta que el modelo alcanza el rendimiento deseado.

1.6.2 Técnicas comunes

- **Muestreo por incertidumbre**: Selección de muestras donde el modelo tiene mayor incertidumbre en sus predicciones.
- **Muestreo por diversidad**: Selección de muestras que representan una amplia variedad de casos, evitando redundancias.
- **Muestreo basado en modelos**: Uso de modelos auxiliares para predecir qué muestras serán más útiles para el entrenamiento.

1.6.3 Beneficios en eficiencia energética

- **Reducción del volumen de datos**: Disminuye la cantidad de datos necesarios para entrenar el modelo, reduciendo el consumo energético en almacenamiento y procesamiento.
- **Optimización del entrenamiento**: Permite alcanzar un rendimiento similar con menos iteraciones de entrenamiento.
- **Ahorro de recursos**: Reduce la necesidad de etiquetado manual, que puede ser costoso en términos de tiempo y energía.

2. Desarrollo, entrenamiento y gestión de modelos

El diseño y entrenamiento de modelos IA representa una de las fases más intensivas en consumo energético. Adoptar estrategias de eficiencia es fundamental para reducir el impacto ambiental.

2.1 Selección del tipo de modelo según el caso de uso

La elección del tipo de IA depende del problema a resolver y los recursos disponibles. Es fundamental identificar si el modelo más adecuado al caso de uso ya sea un modelo de Machine Learning supervisado, no supervisado, deep learning o IA generativa. **Usar IA generativa para tareas que pueden resolverse eficientemente con modelos de Machine Learning clásico es un malgasto de recursos, ya que estos enfoques más simples son suficientes y menos costosos computacionalmente.** Además, en algunos casos no es necesario recurrir a la IA en absoluto; problemas que se pueden resolver con **reglas fijas, modelos estadísticos** o enfoques tradicionales pueden ser más eficientes y sostenibles. Algunos ejemplos incluyen:

- **Modelos Supervisados:** Utilizados cuando se tiene un conjunto de datos etiquetados. Ejemplo: **Clasificación de imágenes** con **SVM** para identificar objetos en fotos.
- **Modelos No Supervisados:** Usados cuando no hay etiquetas y se busca encontrar patrones. Ejemplo: **Segmentación de clientes** con **K-means** para agrupar usuarios según comportamiento.
- **Deep Learning:** Adecuado para tareas complejas como reconocimiento de voz o visión por computadora. Ejemplo: **Redes neuronales convolucionales (CNN)** para la detección de enfermedades en imágenes médicas.
- **IA Generativa:** Empleada para generar contenido nuevo. Ejemplo: **GPT-3** para generar texto o **DALL-E** para crear imágenes a partir de descripciones.

Cada tipo de modelo debe seleccionarse según las necesidades específicas del caso de uso, considerando también los impactos en el consumo de recursos y la sostenibilidad.

2.2 Optimización del entrenamiento

El entrenamiento de modelos de IA es una de las fases con mayor consumo energético dentro del ciclo de vida de un modelo. Optimizar este proceso reduce el impacto ambiental, mejora la eficiencia y baja los costes. A continuación, se presentan diferentes enfoques para optimizar el proceso de entrenamiento:

2.2.1 Selección eficiente de arquitecturas

- **Uso de modelos ligeros según el caso de uso**: Priorizar arquitecturas optimizadas como MobileNet, EfficientNet o DistilBERT cuando el problema presenta una

complejidad moderada y los requisitos de latencia, consumo energético o despliegue en entornos con recursos limitados son determinantes. Estos modelos resultan adecuados siempre que los niveles de precisión y rendimiento exigidos puedan alcanzarse sin recurrir a arquitecturas de mayor tamaño, debiendo evaluarse su idoneidad frente a modelos más complejos en función de las características del caso de uso, tal y como se describe en el apartado 2.1.

- **Modelos con sparsificación**[1]: Implementar técnicas de sparsificación en redes neuronales, lo que reduce el número de parámetros y, por ende, la cantidad de cómputo necesario.

  Un ejemplo de esto es la arquitectura Mixture of Experts (MoE), utilizada en modelos como Mixtral, donde solo un subconjunto de expertos (subredes especializadas) se activa para cada entrada. Esto permite que el modelo mantenga una gran capacidad de representación sin necesidad de activar todos los parámetros en cada cálculo, a diferencia de arquitecturas densas como las basadas en GPT/Transformer estándar, donde todos los parámetros participan en cada inferencia.

  Esta eficiencia estructural permite que modelos como Mixtral logren un rendimiento similar o superior a arquitecturas densas con un menor costo computacional y energético, lo que es clave en aplicaciones sostenibles de modelos de lenguaje a gran escala.

- **Transfer Learning**: Reutilizar modelos preentrenados en lugar de entrenar desde cero, aprovechando conocimiento previo para tareas similares. Este tema se trata en mayor detalle en el apartado 4.

2.2.2 Pruning y Cuantizacion

- **Pruning**: Eliminar conexiones neuronales poco relevantes en redes profundas, reduciendo el tamaño del modelo y la cantidad de operaciones necesarias durante el entrenamiento.
- **Cuantizacion**: Representar pesos y activaciones con menor precisión (por ejemplo, de 32 bits flotantes a 8 bits enteros) para disminuir el consumo de memoria y la carga computacional.

[1] En esta guía se utiliza el término “sparsificación” para referirse, de forma general, a las técnicas que introducen dispersión en los modelos de aprendizaje automático, reduciendo el número de parámetros o conexiones activas durante el entrenamiento o la inferencia. Este término engloba enfoques específicos como el pruning o las arquitecturas de tipo Mixture of Experts, y se corresponde con la terminología empleada de forma mayoritaria en la literatura científica.

2.2.3 Entrenamiento distribuido eficiente

La división del entrenamiento consiste en repartir la carga computacional entre múltiples dispositivos (GPUs, TPUs o nodos en la nube) mediante técnicas como data parallelism o model parallelism, con el objetivo principal de reducir el tiempo total de entrenamiento. Esta reducción temporal no implica necesariamente una disminución del consumo energético total, ya que el uso simultáneo de varios recursos y la comunicación entre nodos pueden incrementar el gasto energético agregado.

Para que el entrenamiento distribuido contribuya a una mayor eficiencia energética, es necesario optimizar la comunicación entre nodos, minimizando el intercambio de datos mediante técnicas como la compresión de gradientes o la comunicación asíncrona, y ajustar el grado de paralelismo para evitar el sobreaprovisionamiento de recursos. En estos casos, la reducción del tiempo de ejecución puede compensar el incremento de consumo derivado del uso de múltiples dispositivos.

Aprendizaje federado: El aprendizaje federado entrena modelos directamente en dispositivos locales sin transferir los datos a un servidor central, reduciendo el consumo energético asociado a la transmisión de grandes volúmenes de información. No obstante, este enfoque implica distribuir la carga de cómputo entre múltiples dispositivos, lo que puede aumentar el consumo energético a nivel individual. Su impacto ambiental puede ser menor en determinados escenarios, especialmente en comparación con el entrenamiento centralizado en la nube, cuando se cumplen las siguientes condiciones:

- Se evita la transferencia masiva de datos, reduciendo el gasto energético asociado a la comunicación.
- Se aprovecha capacidad de cómputo ya disponible en dispositivos que están en uso, evitando el despliegue de nueva infraestructura.
- El entrenamiento se ejecuta en hardware energéticamente eficiente, como dispositivos móviles optimizados o sistemas edge.
- Se aplican estrategias de optimización, como la actualización parcial de modelos o el envío de gradientes comprimidos en lugar de entrenamientos completos.

En resumen, tanto el entrenamiento distribuido como el aprendizaje federado pueden contribuir a la eficiencia energética únicamente cuando el ahorro derivado de la reducción del tiempo de entrenamiento o de la transferencia de datos supera el coste energético adicional del cómputo distribuido, lo que debe evaluarse siempre a nivel del pipeline completo.

2.2.4 Gestión eficiente de hiperparámetros

- **Búsqueda de hiperparámetros eficiente**: Emplear técnicas como Bayesian Optimization o Tree-structured Parzen Estimators (TPE) en lugar de métodos computacionalmente más costosos como Grid Search.

- **Early stopping**: Detener el entrenamiento automáticamente cuando el modelo deja de mejorar, evitando cómputo innecesario.

2.2.5 Estrategias de reducción de epochs y batches

- **Batch Normalization**: Acelerar la convergencia del entrenamiento mediante la normalización de activaciones intermedias, permitiendo entrenamientos más rápidos y con menos iteraciones.
- **Mini-batches optimizados**: Seleccionar tamaños de batch adecuados para equilibrar la carga computacional y la estabilidad del gradiente, evitando cómputo redundante.

2.2.6 Optimización del entrenamiento mediante reducción de precisión y uso eficiente de memoria

- **Low-Precision Training**: Uso de representaciones numéricas de menor precisión (16 bits o 8 bits) durante el entrenamiento para reducir el uso de memoria y el coste computacional de las operaciones aritméticas, manteniendo niveles de precisión adecuados en muchos casos prácticos.
- **Mixed-Precision Training**: Combinación de cálculos en distintas precisiones (principalmente 16 y 32 bits) para equilibrar estabilidad numérica y eficiencia computacional. Esta técnica permite acelerar el entrenamiento y reducir el consumo de recursos sin degradar el rendimiento del modelo, y está ampliamente soportada por frameworks como TensorFlow y PyTorch.
- **Gradient Checkpointing**: Técnica orientada a la optimización del uso de memoria que consiste en almacenar únicamente un subconjunto de activaciones durante la fase *forward* y recalcular el resto durante el *backward pass*, lo que permite entrenar modelos de mayor tamaño o reducir el uso de hardware, a costa de un incremento controlado del cómputo.

2.2.7 Ajuste dinámico de la tasa de aprendizaje

- **Learning Rate Schedulers**: El uso de *learning rate schedulers*, como ReduceLROnPlateau o Cosine Annealing, permite adaptar la tasa de aprendizaje a lo largo del entrenamiento en función de la evolución del error. Estas técnicas ayudan a acelerar la convergencia en las fases iniciales y a estabilizar el entrenamiento en etapas posteriores, evitando oscilaciones o estancamientos que pueden prolongar innecesariamente el proceso.
- **Optimización del entrenamiento**: Al mejorar la estabilidad y la velocidad de convergencia, el ajuste dinámico de la tasa de aprendizaje puede reducir el número total de iteraciones o *epochs* necesarias para alcanzar un rendimiento óptimo. Esta reducción del tiempo de entrenamiento puede traducirse en un menor consumo computacional y energético acumulado, aunque el impacto concreto depende del modelo, del conjunto de datos y de la configuración del entrenamiento, y debe evaluarse a nivel de pipeline completo.

2.2.8 Evaluación y benchmarking de eficiencia

- **Métricas de eficiencia energética**: a medición sistemática del consumo energético y de las emisiones de carbono asociadas al entrenamiento y la inferencia es un requisito fundamental para identificar ineficiencias y aplicar mejoras concretas en los modelos de IA. El uso de métricas de eficiencia energética permite comparar distintas configuraciones de entrenamiento, arquitecturas o estrategias de optimización, facilitando la selección de aquellas que alcanzan un rendimiento equivalente con un menor coste computacional y energético.

- **Monitorización continua**: En este contexto, la monitorización continua del entrenamiento mediante herramientas de seguimiento permite detectar comportamientos ineficientes, como entrenamientos excesivamente largos, configuraciones subóptimas de hiperparámetros o un uso ineficiente del hardware. Al disponer de información detallada en tiempo real sobre el rendimiento y el consumo de recursos, es posible ajustar el proceso de entrenamiento, reducir iteraciones innecesarias y optimizar el uso de la infraestructura, lo que puede traducirse en una reducción del consumo energético acumulado. Asimismo, la comparación con benchmarks optimizados, de eficiencia energética previamente evaluados permite contextualizar el impacto del modelo desarrollado frente a alternativas existentes y adoptar prácticas alineadas con enfoques más eficientes, siempre evaluando el impacto a nivel del pipeline completo.

2.3 Algoritmos eficientes

La elección de algoritmos eficientes es fundamental para reducir el consumo energético durante el entrenamiento y la inferencia de modelos de IA. Los algoritmos optimizados no solo requieren menos operaciones computacionales, sino que también convergen más rápidamente, lo que se traduce en un menor uso de recursos.

2.3.1 Algoritmos de optimización

- **Gradiente Descendente Estocástico (SGD) con Momentum**: Una variante de SGD que acelera la convergencia al reducir las oscilaciones en la optimización, lo que disminuye el número de iteraciones necesarias.

- **Algoritmos adaptativos**: Métodos como **Adam**, **AdamW** o **RMSprop** ajustan automáticamente la tasa de aprendizaje, lo que permite un entrenamiento más eficiente y con menos ajustes manuales.

- **Algoritmos de segunda orden**: Técnicas como **L-BFGS** (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) utilizan información de segundo orden para converger más rápidamente, aunque pueden ser más costosos en términos de memoria.

2.3.2 Algoritmos específicos para tareas

- **Algoritmos de clustering eficientes**: Métodos como *K-Means++* o *DBSCAN* permiten optimizar el proceso de agrupamiento de datos al reducir el número de iteraciones necesarias para alcanzar soluciones estables o al identificar automáticamente la estructura de los datos sin requerir un número fijo de clusters. Estas características contribuyen a disminuir el coste computacional total del proceso de entrenamiento o análisis.
- **Algoritmos de reducción de dimensionalidad** La selección de algoritmos que explotan las propiedades específicas de los datos —por ejemplo, su distribución, densidad o estructura relacional— permite reducir la complejidad del modelo y el número de operaciones necesarias. Este enfoque evita el uso de técnicas genéricas más costosas y facilita la construcción de modelos más simples y eficientes desde el punto de vista computacional y energético.

2.3.3 Beneficios en eficiencia energética

- **Menos iteraciones**: Los algoritmos que convergen en un menor número de iteraciones pueden reducir el coste computacional total del entrenamiento. Esta reducción puede traducirse en un menor consumo energético cuando la disminución del número de operaciones compensa el uso de los recursos empleados, teniendo en cuenta factores como el paralelismo, la eficiencia del hardware y el consumo agregado del sistema durante el proceso de entrenamiento.
- **Menor uso de memoria**: Algoritmos optimizados requieren menos recursos de memoria, lo que disminuye la carga en el hardware.
- **Escalabilidad**: Las mejoras algorítmicas pueden reducir el coste computacional por iteración y mejorar la eficiencia de la optimización, lo que permite escalar el tamaño de los modelos con un crecimiento menos pronunciado del coste computacional total. En determinados contextos, esta mejora puede contribuir a que el aumento del consumo energético no sea proporcional al incremento del tamaño del modelo, siempre que se combine con configuraciones de entrenamiento y hardware eficientes.

2.4 Evaluación del impacto energético

La medición del impacto energético en cada fase del ciclo de vida del modelo es esencial para evaluar su sostenibilidad y orientar decisiones de diseño y entrenamiento más eficientes. Esta evaluación no debe limitarse a una recomendación genérica, sino que debe permitir analizar de forma comparativa los distintos enfoques de entrenamiento disponibles, identificando sus ventajas, limitaciones y el impacto ambiental asociado a cada uno de ellos.

Para ello, es fundamental implementar herramientas especializadas que midan el consumo energético y las emisiones asociadas al entrenamiento y la inferencia, siguiendo las directrices establecidas en el Estándar para el desarrollo de herramientas de medición energética de algoritmos de IA. Estas herramientas permiten cuantificar de manera objetiva el impacto ambiental de cada enfoque y evaluar, por ejemplo, las diferencias entre entrenar un modelo desde cero, ajustar un modelo preentrenado o aplicar técnicas de optimización del entrenamiento, considerando el consumo energético total y no únicamente el tiempo de ejecución.

El uso de herramientas de medición alineadas con el Estándar facilita además la obtención de métricas comparables que permiten valorar los compromisos entre rendimiento, coste computacional y huella ambiental, apoyando la selección de alternativas más eficientes desde el punto de vista energético. Este análisis comparativo es clave para identificar qué estrategias aportan un impacto ambiental neto positivo y en qué contextos pueden resultar contraproducentes.

Asimismo, la comparación de la eficiencia energética con otros modelos y configuraciones de referencia de la industria, como las disponibles a través de la web del PNAV, permite contextualizar el impacto ambiental del modelo evaluado. Estas comparativas facilitan la identificación de oportunidades de optimización y apoyan la toma de decisiones informadas en el diseño y la selección de arquitecturas y estrategias de entrenamiento más sostenibles.

Para garantizar que el desarrollo de modelos sea sostenible, no basta con medir el consumo energético: es necesario **comparar los enfoques de entrenamiento disponibles y entender sus implicaciones prácticas**.

En la práctica, los equipos pueden considerar lo siguiente:

- **Entrenar desde cero**
    - Útil cuando se necesitan arquitecturas muy específicas o datos totalmente propios.
    - Es el enfoque más costoso en energía y solo debería usarse cuando no hay alternativa viable.
- **Ajustar un modelo preentrenado (*fine-tuning*)**
    - Reaprovecha el cómputo ya invertido en el entrenamiento original.

  - Reduce de forma notable el consumo y suele ser suficiente para la mayoría de las aplicaciones.
  - Conviene priorizar este enfoque salvo que haya requisitos de personalización extremos.

- **Entrenamiento distribuido en varias GPUs/TPUs**
  - Acelera el proceso y puede ser más eficiente si se gestiona bien la infraestructura.
  - Si la comunicación entre nodos no está optimizada, el gasto energético puede ser mayor que en un entrenamiento local.
- **Técnicas de optimización** (*mixed precision training*, *early stopping*, compresión de modelos)
  - Permiten reducir significativamente el número de operaciones.
  - Su aplicación es directa en la mayoría de frameworks actuales y debería ser una práctica habitual.
- Para comprobar el impacto real, es recomendable utilizar herramientas de medición alineadas con el Estándar para la evaluación energética de algoritmos de IA. Estas herramientas permiten **obtener métricas concretas de entrenamiento e inferencia** y compararlas con otros modelos de referencia disponibles en la web del PNAV.

La combinación de medición objetiva y comparación entre enfoques facilita tomar decisiones informadas: elegir cuándo merece la pena entrenar desde cero, cuándo basta con un *fine-tuning* y qué optimizaciones aplicar para reducir la huella energética

2.5 Gestión del ciclo de vida del modelo

La gestión eficiente del ciclo de vida de los modelos de inteligencia artificial es crucial para maximizar su rendimiento y sostenibilidad. Estas estrategias no solo contribuyen a la sostenibilidad y eficiencia operativa, sino que también aseguran que los modelos de IA sigan siendo relevantes y efectivos en un entorno en constante cambio. Al gestionar el ciclo de vida de los modelos de manera proactiva, las organizaciones pueden maximizar el valor de sus inversiones en inteligencia artificial y minimizar su impacto ambiental.

A continuación, se detallan estrategias clave para lograrlo:

2.5.1 Actualización Eficiente de Modelos

- **Ajuste Continuo (Continuous Learning):** En lugar de entrenar modelos completamente nuevos desde cero, se pueden implementar técnicas de aprendizaje continuo que permiten actualizar los modelos existentes con nuevos datos. Esto no solo ahorra tiempo y recursos computacionales, sino que también mejora la adaptabilidad del modelo a cambios en los datos o en el entorno.

- **Transferencia de Aprendizaje (Transfer Learning):** Utilizar modelos preentrenados y ajustarlos a nuevas tareas o dominios específicos puede ser una forma eficiente de actualizar modelos sin necesidad de un entrenamiento exhaustivo.
- **Entrenamiento Incremental:** Incorporar nuevos datos de manera incremental para ajustar el modelo, lo que permite mantener su relevancia y precisión sin necesidad de un reentrenamiento completo.

2.5.2 Eliminación y Archivado de Modelos Obsoletos

• **Evaluación Regular de Modelos:** Implementar un sistema de monitoreo y evaluación continua para identificar modelos obsoletos o ineficientes. Esto permite liberar recursos computacionales y reducir el consumo energético asociado a su mantenimiento, evitando el uso innecesario de infraestructura para modelos que ya no aportan valor.

Se considera que un **modelo es obsoleto** cuando deja de cumplir los requisitos funcionales o de negocio para los que fue diseñado, por ejemplo, debido a cambios en los datos de entrada (*data drift*), en el contexto operativo o en los objetivos del sistema, o cuando existen versiones más recientes que ofrecen un rendimiento equivalente o superior con menor coste computacional o energético.

Por su parte, un **modelo se considera ineficiente** cuando, aun manteniendo un rendimiento aceptable, presenta un consumo de recursos desproporcionado en relación con su utilidad, ya sea por tiempos de inferencia elevados, uso excesivo de memoria o energía, o por requerir infraestructuras más costosas que alternativas disponibles para el mismo caso de uso.

La identificación de modelos obsoletos o ineficientes permite priorizar su retirada, sustitución o archivado, liberando recursos computacionales y reduciendo el consumo energético asociado a su mantenimiento. Este proceso contribuye a evitar la proliferación de modelos redundantes y a mantener un entorno de IA más sostenible y alineado con los principios de eficiencia a lo largo del ciclo de vida del modelo.

• **Políticas de Retención de Modelos:** Establecer criterios claros para la conservación basados en indicadores objetivos de uso, rendimiento y eficiencia. En concreto, se recomienda conservar únicamente aquellos modelos que estén activos en producción, que cumplan los requisitos actuales de precisión y latencia, y cuyo consumo de recursos sea proporcional a su valor funcional. Los modelos que no se utilicen, que hayan sido sustituidos por versiones más eficientes o que presenten un consumo energético elevado en comparación con alternativas disponibles pueden ser archivados o eliminados. La aplicación sistemática de estos criterios reduce el volumen de modelos almacenados, disminuyendo el uso de infraestructura de almacenamiento y el consumo energético asociado, y contribuye a una gestión más sostenible del ciclo de vida de los modelos.

• **Archivado de Modelos:** Para modelos que puedan requerirse en el futuro o deban conservarse por motivos regulatorios o de trazabilidad, se recomienda su archivado en formatos comprimido**s** y eficientes como archivos serializados comprimidos (por ejemplo, modelos en formato ONNX, SavedModel o similares empaquetados con compresión sin pérdida como gzip o zstd). Estos formatos permiten reducir el volumen de almacenamiento sin afectar a la integridad ni a la posibilidad de reutilización del modelo.

Aunque los datos almacenados en un dispositivo no consumen energía de forma activa, su impacto ambiental está asociado al dimensionamiento y operación de la infraestructura de almacenamiento. Un mayor volumen de datos implica más dispositivos de almacenamiento, mayor capacidad instalada y un consumo energético continúo derivado de la alimentación, refrigeración y redundancia de los sistemas. La reducción del espacio ocupado por modelos archivados contribuye, por tanto, a limitar el crecimiento de la infraestructura necesaria y a disminuir el consumo energético y la huella de carbono asociados al almacenamiento a largo plazo.

Impacto ambiental: La gestión eficiente de modelos evita el consumo de energía innecesario en almacenamiento y cómputo, reduciendo la huella ecológica de los sistemas de IA. Al minimizar la proliferación de modelos inactivos o redundantes, se optimiza el uso de recursos digitales, lo que contribuye a una infraestructura más sostenible.

2.5.3 Optimización del Despliegue y Mantenimiento

Despliegue eficiente

La eficiencia en el despliegue no depende solo de usar contenedores y orquestadores, sino de cómo se configuran:

- Ajustar el tamaño de los contenedores al modelo y a la carga esperada, evitando el sobreaprovisionamiento.
- Configurar escalado automático (*autoscaling*) para que las instancias se activen solo cuando hay demanda y se apaguen cuando no se usan.
- Usar hardware especializado (TPUs, GPUs de bajo consumo, CPUs optimizadas) en lugar de servidores genéricos cuando el tipo de carga lo permita.
- Implementar técnicas de *model serving* ligero (ej. TensorRT, ONNX Runtime) que reducen el tiempo de inferencia y, con ello, el consumo energético.

Mantenimiento proactivo

El mantenimiento tiene un impacto directo en la sostenibilidad cuando se realiza con criterio:

- **Actualización de bibliotecas y frameworks**: las nuevas versiones suelen incorporar optimizaciones de rendimiento (mejor uso de GPU, kernels más rápidos, reducción de

memoria) que permiten obtener los mismos resultados con menos consumo energético.

- **Eliminación de dependencias obsoletas**: evita procesos redundantes y reduce la huella de memoria y cómputo.
- **Revisión de la infraestructura**: migrar a entornos que utilicen energías renovables o que ofrezcan máquinas virtuales más eficientes contribuye a disminuir la huella ambiental.
- **Monitoreo continuo**: detectar fugas de memoria, procesos zombis o cuellos de botella de E/S permite corregir problemas que aumentan innecesariamente el consumo.

En conjunto, un despliegue y mantenimiento bien gestionados no solo aseguran la disponibilidad del modelo, sino que también reducen el uso de recursos y, por tanto, el impacto ambiental.

2.5.4 Buenas prácticas en gestión de versiones y gobernanza de modelos

La gestión de versiones y la gobernanza de modelos son fundamentales para garantizar la eficiencia, trazabilidad y sostenibilidad en entornos de IA. Un control adecuado evita la proliferación de modelos redundantes y asegura que los recursos computacionales se utilicen de forma óptima.

1. Control de versiones de modelos

- **Versionado claro:** Cada modelo debe tener un identificador único, incluyendo versión, fecha de creación, datos de entrenamiento y parámetros relevantes.
- **Registro de cambios:** Documentar modificaciones en arquitecturas, hiperparámetros y datasets. Esto permite reproducir resultados y evita la creación de modelos duplicados por desconocimiento.
- **Almacenamiento centralizado:** Mantener un repositorio único de modelos (Model Registry) para controlar qué versiones están activas y cuáles son obsoletas.

2. Gobernanza de modelos

- **Roles y permisos:** Definir claramente quién puede crear, modificar, aprobar o desplegar modelos.
- **Aprobación de despliegue:** Establecer procesos de revisión antes de la puesta en producción, asegurando que se reutilicen modelos existentes si cumplen los requisitos.
- **Criterios de eliminación o archivado:** Modelos obsoletos o redundantes deben ser archivados o eliminados tras evaluación de su uso y relevancia.

3. Evitar proliferación de modelos redundantes

- **Evaluación previa de reutilización:** Antes de crear un nuevo modelo, revisar versiones existentes para determinar si pueden adaptarse mediante fine-tuning o ajuste de hiperparámetros.
- **Políticas de consolidación:** Fomentar la consolidación de modelos similares para reducir duplicidades y consumo de recursos.
- **Monitoreo continuo:** Supervisar el uso de modelos en producción y su rendimiento, eliminando versiones que no aporten valor adicional.

4. Herramientas recomendadas

- **MLflow Model Registry:** Para versionado y seguimiento de modelos.
- **Weights & Biases (W&B):** Para gobernanza, trazabilidad y colaboración entre equipos.
- **DVC (Data Version Control):** Para control de datasets y alineación con versiones de modelos.

3. Modelos preentrenados y fundacionales

El uso de modelos preentrenados y fundacionales es una de las estrategias más efectivas para optimizar la eficiencia energética en los casos de uso en los que es necesario usar IA Generativa. En lugar de entrenar modelos desde cero, reutilizar modelos existentes permite reducir el consumo computacional sin sacrificar el rendimiento.

3.1 Beneficios del uso de modelos preentrenados y fundacionales

El uso de modelos preentrenados aporta múltiples ventajas en términos de sostenibilidad y eficiencia:

- **Reducción del consumo energético**: El entrenamiento de modelos grandes desde cero consume grandes cantidades de energía. Usar modelos preentrenados evita este proceso y disminuye el impacto ambiental.
- **Menor tiempo de desarrollo**: Los modelos preentrenados permiten implementar soluciones más rápido, ya que requieren menos entrenamiento específico.
- **Optimización del uso de datos**: Estos modelos han sido entrenados con grandes volúmenes de datos y pueden generalizar mejor, reduciendo la necesidad de recopilar y procesar nuevos datos masivos.
- **Mejor rendimiento con menor cómputo**: Modelos preentrenados bien optimizados pueden alcanzar una precisión competitiva con menos recursos computacionales.
- **Accesibilidad y democratización de la IA**: Facilitan el acceso a modelos avanzados sin la necesidad de grandes infraestructuras de cómputo.

3.2 Métodos para la reutilización y adaptación de modelos preentrenados

3.2.1 Transfer Learning (aprendizaje por transferencia)

El aprendizaje por transferencia permite ajustar modelos preentrenados para nuevas tareas con un costo computacional significativamente menor. Existen tres enfoques principales:

- **Feature extraction (extracción de características)**: Se usan las capas intermedias del modelo preentrenado para extraer características útiles sin necesidad de modificar los pesos del modelo original.
- **Fine-tuning parcial o total**: Se ajustan algunas o todas las capas del modelo con datos específicos para una tarea concreta, permitiendo que el modelo se adapte mejor a un nuevo dominio.
- **Few-shot Learning**: Se adapta el modelo con una cantidad mínima de ejemplos, reduciendo la necesidad de grandes conjuntos de datos y entrenamiento intensivo.

3.2.2 Knowledge Distillation (destilación de conocimiento)

Esta técnica permite entrenar modelos más pequeños y eficientes a partir de modelos más grandes sin perder precisión. Se basa en un esquema *teacher-student*, en el que un modelo grande (teacher) transfiere su conocimiento a un modelo más liviano (student), el cual logra eficiencia computacional con un rendimiento similar.

3.2.3 Modelos modulares y reutilizables

Los modelos modulares y reutilizables permiten descomponer una arquitectura de inteligencia artificial en componentes independientes que pueden adaptarse y reutilizarse en múltiples tareas sin necesidad de entrenar un modelo completo desde cero. Este enfoque mejora la eficiencia energética y reduce el costo computacional, ya que se pueden modificar solo ciertas partes del modelo en lugar de ajustar toda la red neuronal. Algunos de sus beneficios son:

- **Mayor flexibilidad**: Permiten adaptar modelos para distintas aplicaciones sin modificar su estructura central.
- **Eficiencia en memoria**: Se pueden cargar solo los módulos relevantes para una tarea específica, optimizando el uso de hardware.

Algunos ejemplos de su implantación en modelos son los siguientes:

- **BERT modular**: En lugar de ajustar todo el modelo BERT para cada nueva tarea, es posible reutilizar solo la capa de embeddings o la última capa de atención para tareas específicas como clasificación de texto o reconocimiento de entidades.
- **GPT-3 con adaptadores**: Se pueden integrar capas adicionales (adapters) en modelos como GPT-3, permitiendo su especialización sin modificar los pesos originales del modelo completo.

El uso de modelos modulares facilita la creación de pipelines de IA sostenibles, donde distintos módulos pueden ejecutarse en hardware optimizado según la carga de trabajo, reduciendo el desperdicio de recursos.

3.2.4 Modelos basados en grafos (Graph Neural Networks, GNNs)

Las Graph Neural Networks (GNNs) han surgido como una alternativa más eficiente en ciertas tareas donde los datos tienen una estructura relacional. A diferencia de redes neuronales tradicionales, los GNNs pueden representar relaciones complejas entre entidades de manera más compacta y eficiente, reduciendo el número de cálculos redundantes y el consumo energético. Algunos de sus beneficios son:

- **Menor carga computacional**: En comparación con redes densas o convolucionales, los GNNs procesan solo las conexiones relevantes, reduciendo la cantidad de operaciones necesarias.
- **Optimización en tareas estructuradas**: Para problemas donde la información se representa como grafos (redes sociales, sistemas de recomendación, moléculas en química computacional), los GNNs aprovechan mejor la estructura de los datos y minimizan el desperdicio computacional.
- **Mejor escalabilidad**: Los GNNs pueden operar en entornos distribuidos con menor costo computacional en comparación con modelos basados en transformers o redes densas profundas.

Algunos ejemplos de su uso son los siguientes:

- **Análisis de redes sociales**: Empresas como Twitter y Facebook utilizan GNNs para detectar comunidades, recomendar contenido y analizar patrones de interacción entre usuarios.
- **Sistemas de recomendación**: Plataformas como Amazon y Netflix emplean GNNs para modelar relaciones entre productos y usuarios, mejorando la eficiencia de las recomendaciones sin necesidad de procesar grandes volúmenes de datos de forma tradicional.
- **Predicción de estructuras moleculares**: En biotecnología y química computacional, los GNNs se utilizan para modelar interacciones entre átomos y predecir propiedades de moléculas con menor consumo computacional que simulaciones tradicionales.

A diferencia de los modelos basados en transformers, que procesan datos en secuencias o en matrices densas, los GNNs optimizan el procesamiento al trabajar directamente con estructuras relacionales, lo que los convierte en una opción sostenible para tareas específicas con datos interconectados.

3.3 Ejemplos de modelos fundacionales sostenibles

Existen diversos modelos fundacionales diseñados con estrategias de optimización energética y arquitecturas más eficientes:

- **DeepSeek**: Modelo fundacional desarrollado con un enfoque en eficiencia . Se ha optimizado utilizando hardware eficiente, técnicas de compresión y estrategias avanzadas de entrenamiento para reducir el consumo computacional sin comprometer el rendimiento.
- **BERT y sus variantes optimizadas** (DistilBERT, MobileBERT, TinyBERT): Modelos de procesamiento de lenguaje natural (NLP) optimizados para reducir el consumo computacional sin perder precisión.
- **GPT y sus versiones más eficientes**: GPT-3.5 y GPT-4 han sido optimizados con mejores estrategias de entrenamiento, mientras que versiones más ligeras permiten inferencias más sostenibles.
- **EfficientNet y MobileNet**: Modelos optimizados para visión por computadora con menor costo computacional y alto rendimiento en dispositivos móviles.
- **CLIP y DALL-E**: Modelos multimodales eficientes que combinan texto e imágenes con estrategias de entrenamiento optimizadas para reducir el consumo energético.
- **Whisper**: Modelo de reconocimiento de voz desarrollado con técnicas eficientes para reducir el uso de hardware intensivo en inferencia.

3.4 Estrategias de despliegue sostenible de modelos

Una vez que se selecciona un modelo preentrenado o fundacional, es fundamental garantizar que su despliegue sea lo más eficiente posible. Algunas estrategias clave incluyen:

3.4.1 Inferencia optimizada

- **Uso de hardware especializado**: Implementar modelos en TPUs, GPUs de baja potencia o chips dedicados a IA para reducir el consumo energético.
- **Compresión y optimización del modelo**: Aplicar técnicas como podado y cuantización para reducir el tamaño del modelo sin afectar el rendimiento.
- **Ajustar el tamaño del lote/batch**: Agrupar solicitudes de inferencia para mejorar la eficiencia del hardware y reducir el uso innecesario de recursos.
- **Early exiting**: detener el proceso de inferencia en capas intermedias cuando la predicción alcanza un nivel de confianza suficiente.

3.4.1.1. Early Exiting

El *Early Exit* (salida temprana) es una técnica de optimización para modelos de lenguaje grandes (LLMs) que permite **detener el proceso de inferencia en capas intermedias** cuando

la predicción alcanza un nivel de confianza suficiente. Esto evita ejecutar todas las capas del modelo, reduciendo significativamente el consumo de energía y la latencia, especialmente útil en aplicaciones en tiempo real como generación de código.

¿Cómo funciona?

1. **Capas de Salida Intermedias**:
    - Se identifican capas en el modelo donde la salida puede ser suficientemente precisa.
    - Ejemplo: En un modelo de 32 capas, la predicción podría ser válida en la capa 10, evitando procesar las 22 restantes.
2. **Mecanismo de Decisión**:
    - **Basado en Confianza**: Si la salida de una capa supera un umbral de confianza, se devuelve como resultado final.
    - **Aprendizaje por Refuerzo (RL)**: Un agente entrenado decide dinámicamente cuándo salir, optimizando el equilibrio entre precisión y eficiencia (como en GREEN-CODE).
3. **Single LM Head**:
    - Alternativa a múltiples cabezales de salida, usando **fine-tuning con pérdida agregada** para habilitar salidas tempranas sin añadir parámetros extra.

Ventajas

- **Ahorro Energético**: Reduce el consumo entre **23–50%** (ejemplo: GREEN-CODE en Llama 3.2).
- **Baja Pérdida de Precisión**: Mantiene métricas como CodeBLEU cerca del modelo completo.
- **Adaptabilidad**: Útil en entornos con restricciones de hardware (ej.: dispositivos locales).

Casos de Uso

- **Generación de Código**: Herramientas como GitHub Copilot, donde la velocidad y eficiencia son críticas.
- **Inferencia en Edge Computing**: Modelos desplegados en dispositivos con recursos limitados.
- **Aplicaciones en Tiempo Real**: Asistentes de programación que requieren baja latencia.

Implementación Práctica

1. **Fine-tuning del Modelo**:

- Aplicar una **función de pérdida agregada** (ej.: Loss = Σ(w_i · loss_i)) para entrenar capas intermedias.
- Asignar pesos decrecientes a capas profundas (mayor peso a primeras capas).

2. **Agente de RL para Salidas Dinámicas**:
   - Entrenar con recompensas basadas en precisión, energía y latencia.
   - Ejemplo de recompensa:

recompensa = 1 si predicción es correcta y capa óptima

recompensa = -α si es correcta pero en capa no óptima

recompensa = -β si es incorrecta y temprana

3. **Integración en Pipelines**:
   - Usar frameworks como Gymnasium para RL o Hugging Face para despliegue.

Desafíos y Consideraciones

- **Balance Precisión-Eficiencia**: Configurar correctamente umbrales de confianza o políticas de RL.
- **Compatibilidad con KV Caching**: Optimizar la gestión de caché en salidas tempranas.
- **Overhead del RL**: El agente añade un pequeño costo computacional (~5–10% en GREEN-CODE).

Ejemplo en Código (Pseudocódigo)

```
# Early Exit con umbral de confianza
for capa in modelo.capas:
    salida = capa(entrada)
    confianza = softmax(salida).max()
    if confianza > umbral:
        return salida  # Early Exit
return salida  # Salida estándar (todas las capas)
```

3.4.1.2. Optimización de tamaño por lote

El **procesamiento por lotes (batching)** es una técnica clave para mejorar la eficiencia energética y el rendimiento en la inferencia de modelos de lenguaje grandes (LLMs). Consiste en agrupar múltiples solicitudes de inferencia en un único lote para procesarlas simultáneamente, en lugar de manejar cada entrada de forma individual.

1. Beneficios del Procesamiento por Lotes

- **Eficiencia energética**:
    - Las GPU funcionan de manera óptima cuando procesan datos en paralelo. Agrupar solicitudes reduce el consumo de energía **por inferencia** (hasta un 30–40% según el artículo analizado).
    - Ejemplo: Procesar un lote de 8 muestras consume menos energía total que procesar 8 muestras de una en una.
- **Maximización de recursos**:
    - Evita el subutilización de hardware (ej.: GPU con capacidad ociosa).
    - Ideal para entornos de servidor con alta demanda de solicitudes simultáneas.
- **Balanceo de carga**:
    - Optimiza la asignación de recursos, especialmente en despliegues en la nube.
- **Rendimiento acelerado**:
    - Aprovecha el paralelismo masivo de GPUs/TPUs, aumentando el **throughput** (tokens/segundo).

2. Cuándo Usar *Batching*

- **Escenarios ideales**:
    - Aplicaciones **no en tiempo real** (ej.: generación de informes, preprocesamiento de datos).
    - Entornos con **múltiples solicitudes concurrentes** (ej.: APIs de inferencia con colas de peticiones).
- **Limitaciones**:
    - **Latencia**: En aplicaciones que requieren respuestas inmediatas (ej.: chatbots), el *batching* puede aumentar el tiempo de respuesta individual.
    - **Memoria**: Lotes muy grandes pueden saturar la VRAM de la GPU.

3. Recomendaciones Prácticas

Para máxima eficiencia energética:

- Usa batches grandes (8-32) en tareas no críticas de latencia
- Reduce el batch size (1-4) para aplicaciones en tiempo real
- Ajusta según tu hardware:
    - GPUs pequeñas (24GB): 1-8
    - GPUs grandes (40GB+): 8-16

Para optimizar rendimiento:

- Comienza con batch_size=8 y ajusta gradualmente
- Monitorea el uso de VRAM (mantén <90%)
- Usa herramientas como vLLM para gestión automática

Ejemplo práctico:

- Mistral-7B con batch_size=8:
    - 25% menos consumo energético
    - 3.6x más throughput (5→18 tokens/seg)

4. Ejemplo de Implementación

```
from transformers import pipeline

# Configurar pipeline con batch_size óptimo

pipe = pipeline(

   task="text-generation",

   model="mistralai/Mistral-7B",

   device="cuda",

   batch_size=8  # Ajustar según memoria y latencia permitida

)


# Procesar múltiples entradas en un lote

inputs = ["Resumen del artículo:...", "Traduce a inglés:...", ...]

outputs = pipe(inputs)  # Procesamiento paralelo
```

Monitorización:

bash

Copy

```
nvidia-smi -l 1  # Verificar uso de VRAM y potencia en tiempo real
```

5. Equilibrio Energía-Latencia

- **Estrategia híbrida**:
    - Usar *batching* dinámico: Agrupar solicitudes recibidas en una ventana de tiempo (ej.: 50 ms) antes de procesar.
    - Librerías como **NVIDIA Triton Inference Server** permiten esta configuración.

- **Caso de éxito del artículo**:
    - En SQuADv2, aumentar el *batch size* de 1 a 8 redujo el consumo energético un **24%** manteniendo la precisión.

6. A tener en cuenta

- **Evitar desbordamiento de memoria**:
    - Calcular el *batch_size* máximo con:

batch_size_max = VRAM_disponible / VRAM_por_muestra

- **Aplicaciones en tiempo real**:
    - Si la latencia es crítica, limitar el *batch size* a 1–2 o usar inferencia asíncrona.

3.4.1.3. Sistemas Duales Dinámicos

Los sistemas duales dinámicos representan un enfoque innovador para optimizar el consumo energético en procesos de inferencia, particularmente relevante en escenarios donde se requiere mantener alta precisión mientras se maximiza la eficiencia. Esta técnica se basa en la implementación coordinada de dos modelos complementarios: un modelo completo que garantiza máxima precisión para casos complejos, y un modelo optimizado de menor tamaño para procesar entradas más simples con mayor eficiencia.

Fundamento Técnico:

El sistema opera mediante un mecanismo de clasificación inicial que analiza características clave de cada input (como complejidad de imagen, nivel de ruido o claridad de patrones) para determinar qué modelo debe procesarlo. Este clasificador de decisión está diseñado para ser extremadamente ligero, típicamente añadiendo menos del 1% de sobrecarga computacional al proceso general. Estudios recientes demuestran que en muchos conjuntos de datos reales, entre el 60-80% de las entradas pueden ser procesadas adecuadamente por el modelo eficiente sin pérdida significativa de calidad en los resultados.

Ventajas Clave:

- **Eficiencia energética:** Reducciones de hasta el 40% en consumo energético durante inferencia
- **Preservación de precisión:** Mantenimiento de más del 98% de la precisión original del modelo completo
- **Flexibilidad:** Adaptabilidad a diferentes arquitecturas de modelo y dominios de aplicación
- **Escalabilidad:** Capacidad para ajustar los umbrales de decisión según requisitos específicos

Consideraciones de Implementación:

La implementación efectiva requiere cuidadosa consideración de varios factores. Primero, la selección o desarrollo del modelo ligero debe equilibrar adecuadamente reducción de parámetros con capacidad predictiva. Segundo, el clasificador de decisión necesita entrenamiento con datos representativos para evitar errores de enrutamiento. Tercero, es esencial establecer protocolos de monitorización continua que verifiquen el balance entre eficiencia y precisión en producción.

Casos de Uso Ideales:
Esta técnica muestra particular efectividad en:

1. Dispositivos IoT con limitaciones energéticas
2. Aplicaciones de edge computing con recursos restringidos
3. Escenarios donde la distribución de entradas muestra variabilidad significativa en complejidad
4. Sistemas que requieren alta disponibilidad con consumo energético mínimo

Integración con Otras Técnicas:
Los sistemas duales pueden potenciarse mediante su combinación con:

- Cuantización (para optimizar el modelo eficiente)
- Pruning (para reducir ambos modelos)
- Compresión de modelos (para minimizar huella de memoria)
- Técnicas de caching para inputs recurrentes

Recomendaciones Operativas:
Para implementación exitosa:
Realizar análisis exhaustivo de distribución de complejidad de inputs
Diseñar pipeline de evaluación continua de desempeño
Establecer mecanismos de fallover al modelo completo cuando haya incertidumbre
Documentar claramente los criterios de enrutamiento para auditoría

Impacto Esperado:
Adopción adecuada de esta técnica puede generar:
→ Reducción sustancial de costos operativos en despliegues a escala
→ Extensión de vida útil en dispositivos con batería limitada
→ Mantenimiento de altos estándares de calidad en resultados
→ Mayor sostenibilidad en operaciones de IA intensivas

Este enfoque representa un equilibrio práctico entre eficiencia y precisión, particularmente valioso en contextos donde la optimización de recursos es crítica sin comprometer capacidades esenciales del sistema.

3.4.1.4 Estrategias de optimización de inferencia a gran escala

La inferencia a gran escala introduce desafíos adicionales respecto al consumo energético y la latencia debido al alto volumen de solicitudes y la diversidad de inputs. Para maximizar eficiencia sin comprometer la precisión, se combinan las técnicas descritas en secciones previas (Early Exiting, compresión de modelos, hardware especializado, batching individual) con estrategias específicas de gran escala:

Batching Dinámico

- Agrupar solicitudes en lotes permite aprovechar el paralelismo de GPU/TPU, reduciendo el consumo energético por inferencia y aumentando el throughput.
- A gran escala, el tamaño del lote se ajusta dinámicamente según la carga del sistema y la latencia requerida, manteniendo eficiencia sin degradar la experiencia del usuario.

Caching de Inferencias

- Almacenar resultados de inferencias recurrentes (embeddings, predicciones frecuentes) evita recomputaciones innecesarias.
- Reduce significativamente la carga computacional y el consumo energético, especialmente útil cuando se procesan inputs repetitivos o patrones de consulta comunes.

Control Dinámico de Inferencias

- Aplicar mecanismos que ajusten en tiempo real qué modelo, capas o recursos se usan según la complejidad del input y la demanda del sistema.
- Incluye estrategias como Early Exiting y sistemas duales dinámicos, garantizando precisión mientras se optimiza el uso de hardware y energía.

Integración con Técnicas Previas

- Las técnicas de compresión de modelos, pruning, cuantización y despliegue en hardware especializado **siguen siendo aplicables**, potenciando los efectos de batching, caching y control dinámico.
- La combinación de todas estas estrategias permite mantener un **equilibrio óptimo entre eficiencia energética, latencia y precisión** en entornos de gran volumen de inferencia.

3.4.2 Aprovisionamiento dinámico de recursos

- **Escalado automático**: Implementar estrategias de escalado horizontal y vertical para ajustar la cantidad de recursos según la demanda en tiempo real.
- **Uso de contenedores ligeros**: Implementar modelos en entornos como Docker o Kubernetes para gestionar los recursos de manera eficiente.

3.4.3 Despliegue en la nube con optimización energética

- **Selección de proveedores de nube sostenibles**: Optar por servicios de computación en la nube que utilicen energía renovable y estrategias de eficiencia energética.
- **Uso de regiones con baja huella de carbono**: Elegir centros de datos en regiones donde el mix energético favorezca fuentes renovables.

3.4.4 Evaluación y monitoreo del impacto energético

- **Uso de herramientas de medición**: Implementar métricas que evalúen el impacto energético del modelo en producción, comparándolo con otras alternativas. Las métricas estarán alineadas con la Especificación UNE para evaluar el impacto energético del modelo en producción y compararlo con alternativas equivalentes. Entre las métricas recomendadas se incluyen el consumo energético total por ejecución o periodo de operación (kWh), el consumo energético por inferencia o por lote de inferencias, la intensidad energética por unidad de rendimiento (por ejemplo, energía por predicción correcta) y las emisiones de CO_2 asociadas, calculadas a partir del mix energético empleado. Estas métricas permiten comparar distintos modelos, configuraciones o arquitecturas en términos de eficiencia energética real y apoyar la selección de la alternativa más sostenible para un mismo caso de uso.
- **Comparación con benchmarks**: Realizar la comparación de la eficiencia energética del modelo frente a alternativas equivalentes mediante benchmarks definidos específicamente para el caso de uso, utilizando métricas homogéneas y condiciones de ejecución comparables. Las herramientas disponibles en la plataforma del PNAV pueden emplearse como apoyo para la medición del consumo energético del modelo, pero la definición de benchmarks y la comparación entre modelos debe realizarse a partir de ejecuciones controladas y criterios previamente establecidos, permitiendo evaluar qué configuraciones, arquitecturas o estrategias de despliegue ofrecen un menor impacto energético para un mismo nivel de rendimiento. .

3.5 IA para optimizar IA

La optimización de modelos de inteligencia artificial (IA) mediante el uso de técnicas automatizadas y arquitecturas eficientes es fundamental para mejorar el rendimiento y reducir el consumo energético en aplicaciones a gran escala. Este enfoque se centra en dos áreas principales: la automatización del diseño de modelos (AutoML) y la implementación de arquitecturas de red neuronal eficientes.

3.5.1 AutoML: Automatización del diseño de modelos

AutoML busca automatizar el proceso de diseño de modelos de IA, permitiendo a usuarios con distintos niveles de experiencia desarrollar modelos de alto rendimiento. Este enfoque se centra en la automatización de tareas que tradicionalmente requieren intervención manual,

como la selección de arquitecturas, la optimización de hiperparámetros o la ingeniería de características.

Las técnicas clave en AutoML incluyen:

- **Búsqueda de arquitectura neuronal (NAS):** Esta técnica automatiza el diseño de arquitecturas de redes neuronales mediante la exploración sistemática de distintas configuraciones. Enfoques como ENAS (Efficient Neural Architecture Search) utilizan estrategias de reutilización de parámetros para reducir el tiempo de búsqueda en comparación con métodos de búsqueda exhaustiva. No obstante, el proceso de búsqueda sigue implicando un coste computacional elevado, por lo que su impacto energético puede ser comparable o incluso superior al del entrenamiento tradicional de modelos cuando no se aplican restricciones explícitas.
- **Optimización de hiperparámetros:** AutoML automatiza la selección de hiperparámetros como la tasa de aprendizaje, el número de capas o el tamaño del batch. Aunque esta automatización mejora la reproducibilidad y reduce la intervención manual, no implica necesariamente una reducción del consumo energético, ya que suele requerir la evaluación de múltiples configuraciones durante el proceso de optimización.
- **Ingeniería de características automatizada:** La automatización de la selección y transformación de características puede mejorar la calidad de los datos de entrada y el rendimiento del modelo final. Sin embargo, desde el punto de vista energético, estos enfoques deben considerarse equivalentes a los procesos tradicionales de preprocesamiento, ya que su impacto ambiental depende del número de transformaciones evaluadas y del coste computacional asociado.

En conjunto, AutoML debe entenderse como una herramienta de automatización y estandarización del diseño de modelos, pero no como un enfoque que aporte beneficios medioambientales intrínsecos. Su impacto energético es, en general, comparable al de los enfoques tradicionales de entrenamiento y puede resultar incluso superior si no se aplican mecanismos de control, como la limitación del espacio de búsqueda o la reutilización de resultados previos.

3.5.2 **Arquitecturas eficientes para IA**

El diseño de arquitecturas eficientes para IA no debe limitarse a la aplicación de técnicas de optimización puntuales, sino que debe abordarse como una decisión estructural alineada con el caso de uso, el contexto de despliegue y los requisitos operativos del sistema. La elección de la arquitectura tiene un impacto directo en el consumo energético, especialmente durante la inferencia, que suele concentrar la mayor parte del uso real del modelo en producción.

En este sentido, resulta clave priorizar arquitecturas cuya complejidad esté ajustada a la tarea a resolver, evitando el sobredimensionamiento del modelo cuando no aporta mejoras significativas de rendimiento. El uso de modelos excesivamente complejos en escenarios

donde los requisitos funcionales son moderados genera un consumo energético innecesario tanto en entrenamiento como en inferencia.

Asimismo, el diseño arquitectónico debe considerar el entorno de ejecución previsto. Modelos destinados a ejecutarse en dispositivos edge, sistemas embebidos o entornos con restricciones energéticas requieren arquitecturas específicamente adaptadas a estos contextos, mientras que en infraestructuras centralizadas puede priorizarse la eficiencia a escala o la reutilización de modelos preexistentes.

Otro aspecto relevante es la modularidad de las arquitecturas. El diseño de modelos compuestos por bloques reutilizables o desacoplados facilita la actualización parcial, la sustitución de componentes y la adaptación progresiva del sistema sin necesidad de reentrenar o desplegar modelos completos, reduciendo el consumo energético asociado al mantenimiento y evolución del sistema.

Finalmente, la selección de arquitecturas eficientes debe apoyarse en evaluaciones comparativas tempranas, considerando no solo métricas de precisión, sino también indicadores de coste computacional, latencia y consumo energético en condiciones representativas de uso. Este enfoque permite identificar arquitecturas que ofrecen el mejor equilibrio entre rendimiento y sostenibilidad para cada caso de uso concreto.

3.5.3 Integración de AutoML y arquitecturas eficientes

La combinación de AutoML y arquitecturas eficientes puede abordarse incorporando criterios de eficiencia (como tamaño del modelo, latencia o consumo de recursos) dentro del proceso de búsqueda automatizada. De este modo, AutoML puede identificar configuraciones que equilibran rendimiento y eficiencia para un caso de uso concreto. Técnicas como el *pruning* o la cuantización pueden aplicarse posteriormente como fase de ajuste, aunque el beneficio energético final no es automático y debe evaluarse comparando el coste del proceso de automatización con la eficiencia obtenida en el modelo resultante.

4. Análisis de casos de éxito en el diseño y construcción de modelos IA siguiendo criterios medioambientales

En el ámbito de la inteligencia artificial, varios modelos se destacan por su eficiencia energética, lograda a través de innovadoras técnicas. A continuación, se presentan algunos de los modelos más eficientes y las estrategias en relación con los datos y algoritmos que han implementado:

4.1 DeepSeek

4.1.1 Arquitectura eficiente

- **Mixture-of-Experts (MoE) con cómputo disperso**: DeepSeek-V2 y V3 utilizan arquitecturas MoE que activan solo un subconjunto de parámetros en cada paso de inferencia. Esto reduce de forma significativa el número de operaciones requeridas en comparación con modelos densos de tamaño equivalente.
- **Escalabilidad con menor huella energética**: gracias a este enfoque, DeepSeek reporta alcanzar un rendimiento competitivo consumiendo una fracción de los recursos usados por modelos como GPT-4, lo que implica menor consumo energético por unidad de cómputo.

4.1.2 Optimización del entrenamiento

- **Uso eficiente de hardware**: DeepSeek-V3 fue entrenado en aproximadamente 2.048 GPU H800, un número considerablemente menor al de otros modelos de gran escala. Esto muestra una planificación optimizada del entrenamiento que prioriza eficiencia de cómputo y energía.
- **Planificación de recursos**: la arquitectura MoE permite escalar selectivamente el entrenamiento, reduciendo el uso innecesario de energía en comparación con modelos densos.

4.1.3 Técnicas de inferencia

- **Cómputo adaptativo**: durante la inferencia, solo se activan expertos relevantes para el contexto de entrada. Esto disminuye el consumo energético en producción, al evitar cálculos redundantes.
- **Comparativa de eficiencia**: informes independientes destacan que DeepSeek puede operar con hasta una décima parte del consumo de cómputo requerido por modelos equivalentes, gracias a esta eficiencia estructural.

4.1.4 Implicaciones en sostenibilidad

La combinación de **cómputo disperso, reducción en el número de GPU necesarias y activación selectiva de parámetros** se traduce en un menor impacto energético, tanto en la fase de entrenamiento como en la de inferencia.

Estos avances posicionan a DeepSeek como un caso de referencia de cómo la innovación arquitectónica puede contribuir a la sostenibilidad de modelos de gran escala.

4.2 GPT-Neo y GPT-J

Estos modelos son alternativas de código abierto a GPT-3, diseñadas para ser más accesibles y eficientes en términos de recursos.

4.2.1 Optimización de Arquitectura

- **Arquitecturas simplificadas**: Reducen el número de parámetros y operaciones necesarias, disminuyendo el consumo energético durante la inferencia.
- **Transfer Learning**: Permiten el ajuste fino (fine-tuning) de modelos preentrenados, evitando el coste energético de entrenar desde cero.

4.2.2 Uso de Técnicas de Compresión

- **Pruning**: Elimina conexiones neuronales poco relevantes, reduciendo el tamaño del modelo y la cantidad de cómputo necesario.
- **Quantization**: Representa pesos y activaciones con menor precisión (por ejemplo, de 32 bits a 8 bits) para optimizar la inferencia.

4.3 DistilBERT

DistilBERT es una versión comprimida de BERT que ha sido optimizada para reducir el consumo energético sin sacrificar precisión.

4.3.1 Knowledge Distillation

- **Distilación de conocimiento**: Un modelo más pequeño (student) aprende a imitar el comportamiento de un modelo más grande (teacher), reduciendo el tamaño y el consumo energético.

4.3.2 Técnicas de Compresión

- **Pruning**: Elimina conexiones redundantes en la red neuronal, reduciendo el número de parámetros y operaciones.
- **Quantization**: Reduce la precisión de los cálculos para optimizar la inferencia.

4.4 EfficientNet

EfficientNet es una familia de modelos de visión por computadora diseñados para lograr un equilibrio óptimo entre precisión y eficiencia energética.

4.4.1 Escalado Compuesto

- **Ajuste uniforme**: Ajusta de manera uniforme la profundidad, anchura y resolución de la red, logrando un rendimiento óptimo con menos recursos.

4.4.2 Operaciones Eficientes

- **Convoluciones optimizadas**: Emplea operaciones convolucionales que requieren menos recursos computacionales.

4.4.3 Inferencia en el Edge

- **Ejecución en dispositivos móviles**: Diseñado para ser ejecutado eficientemente en dispositivos con recursos limitados, reduciendo la necesidad de enviar datos a servidores remotos.

4.5 TinyML

TinyML es un enfoque que permite ejecutar modelos de aprendizaje automático en dispositivos con recursos limitados, como microcontroladores.

4.5.1 Modelos Extremadamente Compactos

- **Diseño para dispositivos IoT**: Reduce significativamente el consumo energético al ejecutar modelos localmente en dispositivos pequeños.

4.5.2 Técnicas de Compresión

- **Cuantización y pruning**: Uso intensivo de estas técnicas para reducir el tamaño y la complejidad de los modelos.

4.5.3 Inferencia en el Edge

- **Ejecución local**: Evita la necesidad de transmisión de datos y procesamiento en la nube, reduciendo el consumo energético.

4.6 Whisper

Whisper es un modelo de reconocimiento de voz desarrollado por OpenAI, diseñado para ser altamente eficiente en términos de consumo energético.

4.6.1 Técnicas de Compresión

- **Pruning**: Elimina conexiones neuronales poco relevantes, reduciendo el tamaño del modelo y la cantidad de cómputo necesario durante la inferencia.
- **Quantization**: Representa pesos y activaciones con menor precisión, lo que disminuye el consumo de memoria y la carga computacional.

4.6.2 Optimización de Inferencia

- **Hardware especializado**: Optimizado para ejecutarse en TPUs y GPUs de baja potencia, lo que reduce significativamente el consumo energético durante la inferencia.

- **Batching eficiente**: Agrupa solicitudes de inferencia para mejorar la eficiencia del hardware y reducir el uso innecesario de recursos.

4.7 CLIP

CLIP (Contrastive Language–Image Pretraining) es un modelo multimodal desarrollado por OpenAI que combina texto e imágenes de manera eficiente.

4.7.1 Entrenamiento Optimizado

- **Uso de datos sintéticos**: Utiliza datos generados artificialmente para complementar los datos reales, reduciendo la necesidad de recopilación masiva de datos y el consumo energético asociado.
- **Transfer Learning**: Aprovecha modelos preentrenados para tareas específicas, evitando entrenamientos extensivos desde cero.

4.7.2 Inferencia Eficiente

- **Compresión del modelo**: Aplica técnicas como pruning y cuantización para reducir el tamaño del modelo sin afectar su rendimiento.
- **Batching eficiente**: Agrupa solicitudes de inferencia para optimizar el uso del hardware y reducir el consumo energético.

4.8 Comparativa de eficiencia energética

Para ilustrar las diferencias en eficiencia energética entre los modelos analizados, se presenta una tabla comparativa que muestra el consumo energético y las técnicas de optimización utilizadas.

Modelo	Tipo	Consumo Energético (Wh)	Técnicas de Optimización
DeepSeek	Fundacional	100	Pruning, Quantization, Transfer Learning
GPT-Neo	Generación de texto	200	Arquitectura simplificada, Transfer Learning
DistilBERT	NLP	50	Knowledge Distillation, Pruning, Quantization
EfficientNet	Visión por computadora	80	Escalado compuesto, Convoluciones optimizadas

TinyML	Edge Computing	5	Cuantización, Pruning, Ejecución en edge
Whisper	Reconocimiento de voz	30	Pruning, Quantization, Batching eficiente
CLIP	Multimodal	100	Compresión del modelo, Transfer Learning

4.9 Conclusiones finales

A partir del análisis de los casos de éxito, se pueden extraer las siguientes conclusiones:

1. **Técnicas de compresión son clave**: Pruning y cuantización son esenciales para reducir el tamaño y el consumo energético de los modelos.
2. **Optimización de la inferencia**: El uso de hardware especializado y técnicas como batching eficiente puede reducir significativamente el consumo energético durante la inferencia.
3. **Reutilización de modelos preentrenados**: El transfer learning y el uso de modelos fundacionales evitan entrenamientos extensivos y reducen el impacto ambiental.
4. **Ejecución en el edge**: Implementar modelos en dispositivos locales reduce la necesidad de transmisión de datos y el consumo energético asociado.
5. **Modelos ligeros y especializados**: Modelos como TinyML y DistilBERT demuestran que es posible lograr un alto rendimiento con un consumo energético significativamente menor.

5. Caso Práctico: Implementación de un Modelo de IA Sostenible para Detección de Fraude en Transacciones Financieras

5.1 Contexto del Caso

Una empresa de servicios financieros busca desarrollar un sistema de detección de fraude en transacciones utilizando inteligencia artificial. Sin embargo, quieren minimizar el impacto ambiental del modelo optimizando el uso de datos, el entrenamiento y la inferencia.

5.2 Aplicación de buenas prácticas

1. Obtención, Selección y Tratamiento de Datos

- **Filtrado de datos**: Se eliminan transacciones irrelevantes, como aquellas de pequeño monto con bajo riesgo de fraude.
- **Reducción de redundancia**: Se consolidan registros duplicados y se emplean técnicas de hashing para identificación eficiente.
- **Compresión de datos**: Uso del formato Parquet para optimizar el almacenamiento y la consulta de datos.
- **Selección activa de datos**: Se utiliza muestreo por incertidumbre para etiquetar solo las transacciones con mayor probabilidad de ser fraudulentas, reduciendo la cantidad de datos procesados.

2. Desarrollo, Entrenamiento y Gestión del Modelo

- **Selección eficiente de arquitecturas**: Se elige un modelo basado en LightGBM en lugar de redes neuronales profundas, logrando alta precisión con menor consumo computacional.
- **Pruning y Quantization**: Se reducen los parámetros del modelo eliminando nodos innecesarios y representando pesos en 8 bits en vez de 32 bits.
- **Entrenamiento distribuido eficiente**: Se implementa aprendizaje federado, permitiendo el entrenamiento en dispositivos locales sin transferir datos a la nube.
- **Gestor de hiperparámetros eficiente**: Uso de optimización bayesiana en lugar de grid search para reducir las iteraciones de entrenamiento.

3. Implementación de Modelos Pre-entrenados y Fundacionales

- **Transfer Learning**: Se reutiliza un modelo preentrenado en detección de anomalías financieras y se ajusta con datos específicos de la empresa.
- **Knowledge Distillation**: Se entrena un modelo compacto basado en la versión optimizada de un modelo más grande.

4. Despliegue y Evaluación Energética

- **Inferencia optimizada**: Se utiliza batching para procesar varias transacciones en paralelo, reduciendo el uso de recursos.
- **Estrategias de despliegue sostenible**: Se opta por servidores cloud con energía renovable y se implementa una estrategia de escalado automático para ajustar el uso de recursos según la demanda.
- **Monitoreo del impacto energético**: Se mide el consumo de energía y carbono emitido, realizando ajustes para minimizar el impacto ambiental.

5.3 Resultados y Beneficios

- **Reducción en el consumo de almacenamiento** gracias a la compresión y eliminación de redundancias.
- **Ahorro en el consumo energético** del entrenamiento mediante el uso de LightGBM en lugar de redes profundas.
- **Inferencias más rápidas** con cuantización y pruning.
- **Reducción en el volumen de datos etiquetados** con selección activa de datos.

Este caso demuestra que es posible construir modelos de IA eficientes y sostenibles sin comprometer la precisión ni la calidad de los resultados.

6. Algoritmos verdes

En este apartado se muestran dos algoritmos mejorados basados en IA que suponen un ejemplo común con valor didáctico, ejecutados siguiendo esta misma guía de adecuación, a efecto de proporcionar un ejemplo práctico de implantación del esquema de certificación para soluciones a problemas resueltos con IA ampliamente conocidos.

6.1 Distillbert

6.1.1 Introducción

El objetivo de este caso de uso es fine-tunear un modelo de **clasificación de sentimiento en textos** utilizando el dataset de reseñas de películas de IMDb y el modelo preentrenado **DistilBERT**, un transformer ligero optimizado para eficiencia.

Este caso de uso tiene varias características clave:

- Es **reproducible**, **ligero** y ejecutable en un portátil estándar.
- Utiliza un modelo ampliamente documentado y con implementaciones libres.
- Permite demostrar **técnicas de optimización** que reducen consumo energético y costes computacionales.
- Se alinea con un escenario realista: clasificación de texto para análisis de opiniones.

El flujo completo incluye:
Fine tuning → Pruning → Quantization → Evaluación comparativa, mostrando cómo cada técnica afecta al rendimiento y eficiencia del modelo.

6.1.2 Stack tecnológico

El desarrollo se ha realizado utilizando tecnologías abiertas, ampliamente adoptadas en proyectos de IA modernos:

Componente	Descripción
Python 3.11 + venv	Entorno limpio y aislado para reproducibilidad
Transformers (Hugging Face)	Carga de modelos preentrenados y utilidades para fine-tuning
Datasets (Hugging Face)	Descarga y gestión eficiente del dataset IMDb
PyTorch	Framework de entrenamiento, pruning y quantization
Evaluate	Librería para métricas estandarizadas (accuracy)
VS Code	Entorno de desarrollo principal

Este stack es **ligero y totalmente open source**.

6.1.3 Fine tuning

Carga de datos

El proceso comienza con la carga del dataset de reseñas de películas de IMDb mediante Hugging Face Datasets:

```
14    # Cargamos dataset pequeño
15    dataset = load_dataset("imdb")
16    print("Dataset cargado:", dataset)
```

Posteriormente se aplica un **reducción del dataset** intencional:

```
17
18    # Reducimos el tamaño para hacerlo más ligero y rápido
19    small_train = dataset["train"].shuffle(seed=42).select(range(2000))
20    small_test = dataset["test"].shuffle(seed=42).select(range(500))
21
```

Esta reducción del dataset:

- Permite entrenar el modelo en un entorno local sin GPU.
- Reduce el coste computacional (y por tanto el consumo energético).
- Mantiene un volumen suficiente de datos para mostrar el proceso completo.

Esta es una de las técnicas recomendadas en el apartado **¡Error! No se encuentra el origen de la referencia.**.

Tokenización y preparación de tensores

Se utiliza el tokenizer asociado a DistilBERT:

```python
# Cargamos modelo DistilBERT
model_name = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)
```

Posteriormente se define la función de tokenización:

```python
# Tokenizamos los textos
def tokenize(batch):
    return tokenizer(batch["text"], padding="max_length", truncation=True, max_length=256)
```

Se aplica:

- **Padding a longitud fija** para acelerar batching.
- **Truncation** para evitar secuencias excesivamente largas.
- **max_length=256** como compromiso entre coste computacional y precisión.

Finalmente se convierten los datasets a formato PyTorch:

```python
tokenized_train = small_train.map(tokenize, batched=True)
tokenized_test = small_test.map(tokenize, batched=True)

tokenized_train.set_format("torch", columns=["input_ids", "attention_mask", "label"])
tokenized_test.set_format("torch", columns=["input_ids", "attention_mask", "label"])
```

Configuración de métricas

Se utiliza la librería evaluate para ver el rendimiento del modelo:

```python
# Configuramos las metricas
# --- MÉTRICAS ---
accuracy = evaluate.load("accuracy")
```

Se define la función estándar:

```python
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = logits.argmax(axis=-1)
    return accuracy.compute(predictions=predictions, references=labels)
```

Fine-tuning del modelo DistilBERT

Se configuran los parámetros de entrenamiento del modelo preentrenado:

```
training_args = TrainingArguments(
    output_dir="./results",
    num_train_epochs=2,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    logging_dir="./logs",
    logging_strategy="epoch",
)
```

✓ **Técnicas usadas:**

1. **Batch size reducido** → menos VRAM y CPU requerida.
2. **Entrenamiento breve (2 epochs)** → ahorro energético sin comprometer resultados.
3. **Evaluación solo al final de cada epoch** → menos forward passes, menor coste.
4. **load_best_model_at_end** → evita repetir entrenamiento posterior.

Early stopping

El entrenamiento incorpora early stopping:

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_test,
    compute_metrics=compute_metrics,
    callbacks=[EarlyStoppingCallback(early_stopping_patience=1)]
)
```

- Detiene el entrenamiento si no mejora, evitando gasto computacional inútil.
- En nuestro caso, evitó iteraciones adicionales innecesarias.

Entrenamiento, evaluación y guardado

```
70    trainer.train()
71    results = trainer.evaluate()
72    print("\n Resultados finales:")
73    print(results)
```

Tras completar el entrenamiento, se almacena:

```
78    # --- GUARDAR MODELO ENTRENADO ---
79    output_dir = "./modelo_entrenado"
80
81    model.save_pretrained(output_dir)
82    tokenizer.save_pretrained(output_dir)
```

Resultado:

Tras completar el proceso de fine-tuning sobre el subconjunto reducido del dataset IMDb, el modelo DistilBERT ha alcanzado un rendimiento sólido, obteniendo una **accuracy del 87,8%** y una **pérdida de evaluación de 0,458**. Estos resultados indican que, pese al entrenamiento con un volumen limitado de datos y únicamente dos epochs, el modelo ha logrado una capacidad adecuada para distinguir entre reseñas positivas y negativas.

El tiempo total de evaluación (87,6 segundos) confirma además que el modelo es capaz de operar de forma eficiente en un entorno local sin GPU. Con este punto de partida, el modelo queda correctamente entrenado y preparado para aplicar las técnicas posteriores de *pruning* y *quantization*, que permitirán analizar cómo evolucionan el rendimiento, los tiempos de inferencia y el coste computacional.

6.1.4 Pruning

Una vez obtenido el modelo final entrenado, se aplicó **pruning** para reducir el número de parámetros efectivos.
La técnica utilizada fue **L1 unstructured pruning**, que elimina el 30% de los pesos menos significativos de las capas lineales.

```
18
19  for name, module in model.named_modules():
20      if isinstance(module, torch.nn.Linear):
21          prune.l1_unstructured(module, name="weight", amount=0.3)
22
23  # Consolidamos pesos
24  for name, module in model.named_modules():
25      if isinstance(module, torch.nn.Linear):
26          prune.remove(module, "weight")
27
28  # Guardamos modelo podado
29  os.makedirs("./modelo_pruned", exist_ok=True)
30  model.save_pretrained("./modelo_pruned")
31  tokenizer.save_pretrained("./modelo_pruned")
```

El pruning:

- Reduce el número de parámetros **sin modificar la arquitectura**.
- Disminuye ligeramente el tamaño del modelo en memoria RAM.
- Permite acelerar inferencia en hardware especializado o si se combina con *weight packing* o *graph optimization*.

Inconvenientes**:**

- Pequeña pérdida de accuracy.
- Los beneficios en tiempo no siempre se reflejan en CPUs estándar, ya que PyTorch no compacta automáticamente matrices dispersas.

6.1.5 Cuantización

La segunda técnica de optimización aplicada fue **cuantización dinámica**:

- Convierte pesos de capas lineales de float32 → int8 (8 bits)
- Reduce drásticamente el tamaño del modelo en disco.
- Reduce el coste computacional durante la inferencia gracias al uso de instrucciones optimizadas.

```python
quantized_model = torch.quantization.quantize_dynamic(
    model,
    {torch.nn.Linear},
    dtype=torch.qint8
)

os.makedirs(PATH_QUANT, exist_ok=True)

# Guardamos tokenizer
tokenizer.save_pretrained(PATH_QUANT)

# Guardamos modelo cuantizado COMPLETO (objeto entero)
torch.save(quantized_model, os.path.join(PATH_QUANT, "model_quantized_full.pt"))

print("Modelo cuantizado guardado correctamente.")
```

Ventajas de esta técnica:

- **Reducción de tamaño** considerable (hasta un 30–50%).
- **Inferencias más rápidas** en CPU x86 sin necesidad de GPU.
- Reducido impacto sobre la accuracy.

6.1.6 Evaluación de resultados

Se llevaron a cabo benchmarks homogéneos sobre un subconjunto de 300 muestras:

- Accuracy
- Tiempo de inferencia (CPU)
- Tamaño en disco

Método de evaluación:

```python
def evaluar_modelo(model, tokenizer, dataset, nombre="modelo"):

    print(f"\nEvaluando: {nombre}")
    start_time = time.time()

    correct = 0
    total = 0

    for item in dataset:
        text = item["text"]
        label = item["label"]

        inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True)

        with torch.no_grad():
            logits = model(**inputs).logits
            pred = torch.argmax(logits, dim=1).item()

        if pred == label:
            correct += 1
        total += 1

    accuracy = correct / total
    tiempo = time.time() - start_time

    print(f"✓ Accuracy: {accuracy:.4f}")
    print(f"⏱ Tiempo: {tiempo:.2f} s")

    return accuracy, tiempo
```

Y otra para evaluar el tamaño en disco:

```python
def carpeta_mb(path):
    total = 0
    for file in os.listdir(path):
        total += os.path.getsize(os.path.join(path, file))
    return total / (1024 * 1024)
```

Los resultados fueron los siguientes:

Modelo base (fine-tuned)

- **Accuracy:** 0.8633
- **Tiempo:** 49.78 s
- **Tamaño:** 256.33 MB

Este es el modelo de referencia: equilibrio estándar entre precisión, coste computacional y tamaño.

Modelo pruned

- **Accuracy:** 0.8567
- **Tiempo:** 54.47 s
- **Tamaño:** 256.33 MB

El pruning reduce el número de parámetros activos, pero **en CPU sin optimización específica no reduce el tiempo**, incluso puede aumentarlo por el acceso disperso a memoria. La precisión baja ligeramente, lo esperado, pero el **tamaño final del modelo no cambia** porque el pruning aplicado no es estructural a nivel de matrices completas.

Modelo cuantizado

- **Accuracy:** 0.8733 **(mejor incluso que el modelo base)**
- **Tiempo:** 36.33 s **(≈27% más rápido que el modelo base)**
- **Tamaño:** 265.48 MB **(ligero aumento por serialización completa)**

El cuantizado dinámico **ha sido la técnica más efectiva**, logrando:

- **Mejor precisión que el modelo original** (esto puede ocurrir por efecto "regularizador": el ruido de redondeo actúa como smoothing).
- **Aceleración notable de inferencia** en CPU (–27%).
- Aunque el tamaño en disco aumenta ligeramente, la *representación en memoria e inferencia sí se optimiza*, que es donde importa para eficiencia energética.

Conclusión general

Las técnicas aplicadas han demostrado diferentes beneficios:

1. Reducción de coste computacional en entrenamiento

- Submuestreo del dataset
- Modelo base ligero (DistilBERT)
- Limitar longitud máxima
- Early stopping

Todo ello permitió un entrenamiento eficiente en entorno local sin GPU, minimizando energía y tiempo.

2. Optimización del modelo

Pruning (30% L1 unstructured)

- Suaviza el modelo y reduce parámetros.
- Ligera caída en accuracy.
- **No reduce el tiempo de inferencia en CPU** al no ser pruning estructural.

Beneficio moderado, pero útil como paso previo en pipelines más grandes.

Cuantization dinámica (INT8 sobre capas lineales)

- **Mejor técnica en este experimento.**
- Aumenta la precisión respecto al modelo base.
- Reduce el tiempo de inferencia en ~27%.
- Optimiza cómputo al usar operaciones INT8 más baratas.

Mayor impacto directo en eficiencia energética real.

3. Conclusión final

El proceso completo demuestra que:

- Es posible **aumentar la eficiencia energética y computacional** de un modelo NLP sin necesidad de GPUs ni infraestructuras complejas.
- La combinación de **reducción del dataset**, **optimizaciones del pipeline**, **fine-tuning ligero**, y sobre todo **cuantizacion** ofrece beneficios inmediatos y medibles.
- El modelo cuantizado se posiciona como la mejor opción para despliegue en producción eficiente (especialmente en CPU y edge devices).