Guía de configuración de la ejecución

Financiado por la Unión Europea
NextGenerationEU
MINISTERIO PARA LA TRANSFORMACIÓN DIGITAL Y DE LA FUNCIÓN PÚBLICA
Plan de Recuperación, Transformación y Resiliencia
ENIA
España | digital 2026
accenture

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 objetivo de esta guía es proporcionar un marco metodológico claro y práctico para configurar, entrenar, desplegar y operar modelos de inteligencia artificial (IA) de forma eficiente, reproducible, segura, y con un mínimo impacto ambiental. Se dirige a desarrolladores, ingenieros de ML, equipos DevOps, responsables de infraestructura, y gestores de proyectos IA. La idea es que puedan aplicar buenas prácticas concretas en instalación, versionado, hardware, software, ejecución distribuida, y considerar tecnologías como Blockchain en su diseño, siempre con criterios de sostenibilidad energética.

1. Principios de sostenibilidad y eficiencia en la ejecución de IA

Algunos principios fundamentales que deben guiar la configuración y operación de IA:

- **Transparencia energética**: documentar, medir y reportar el consumo energético de los modelos ("energy footprint"). En la Unión Europea, el recién entrado en vigor EU AI Act exige documentación técnica que incluya desglose del consumo energético para modelos generales ("General-Purpose AI Models").
- **Reproducibilidad y trazabilidad**: no basta con entrenar - se deben poder replicar los resultados, incluyendo versiones exactas de librerías, entornos, y configuración de hardware/software.
- **Optimización proactiva**: aplicar técnicas de optimización de modelos (pruning, cuantización, distillation, sparsity) antes de desplegar para reducir consumo sin degradar significativamente el rendimiento.
- **Uso de energía limpia**: elegir proveedores o ubicaciones con fuentes renovables, preferir momentos del día en que la red eléctrica tenga mejor mix renovable.
- **Eficiencia sobre escala**: más no siempre es mejor; distribuir cargas, paralelizar solo cuando aporte netamente, evitar sobrecapacidad.
- **Cumplimiento normativo y ético**: cumplir con directivas europeas, estándares emergentes, regulaciones locales de eficiencia energética y emisiones.

2. Metodologías de configuración de la ejecución de modelos IA

Este bloque aborda los mecanismos técnicos mediante los cuales se asegura que la ejecución de modelos sea reproducible, segura, optimizada y con versión controlada.

2.1. Instalación y versionado de librerías

Entornos virtuales y contenedores (conda, venv, Docker)

- Uso de entornos virtuales (venv, conda) o contenedores (Docker) permite aislar dependencias, evitar conflictos, asegurar que versiones concretas se mantengan.
- Docker facilita además empacar no solo librerías, sino versiones de sistema operativo, drivers, etc., lo que mejora reproducibilidad.

Control de versiones (pip freeze, poetry, requirements.txt, lockfiles)

- Registrar versiones exactas de librerías es esencial: un requirements.txt con versiones fijas, o usar poetry.lock o Pipfile.lock para que todos los desarrolladores / servidores corran con versiones idénticas.
- También se pueden usar hashes de archivos de paquetes (como hace pip-freeze o poetry) para evitar cambios inesperados.

Compatibilidad y reproducibilidad

- Verificar compatibilidad entre librerías dependientes (por ejemplo versiones de CUDA, drivers GPU, versiones de tensor libraries) para evitar picos de consumo por incompatibilidades que desencadenen recálculos o uso ineficiente.
- Mantener entornos de prueba reproducibles para benchmarking de consumo, rendimiento y verificar que las optimizaciones no degradan la precisión o introduce bugs.

Gestión del ciclo de vida del software

Prácticas de CI/CD

- Integrar pruebas automáticas que verifiquen que los cambios en código o en librerías no incrementen indebidamente el consumo: por ejemplo tests de rendimiento, tests de uso de GPU/CPU, tests energéticos si posible.
- Automatizar despliegues reproducibles: versiones de modelos, artefactos binarios, containers, etc.

Integración con repositorios y registro de librerías

- Usar repositorios internos de paquetes donde se controle la aprobación de versiones nuevas (por ejemplo un repositorio PyPI interno o mirror).
- Registrar librerías en uso, versiones aprobadas, historial de actualizaciones, para poder revertir si una nueva versión introduce regresión energética o fallos.

Seguridad y trazabilidad de dependencias

- Escaneo de dependencias para vulnerabilidades.
- Verificación de firma de paquetes.
- Trazabilidad: saber qué versión de qué librería fue usada para generar un modelo concreto, para poder reproducir o auditar después.

Planificación y distribución de tareas en algoritmos
Estrategias de paralelización (multi-threading, multi-processing)

- Paralelización adecuada para aprovechar CPUs múltiples, núcleos, hilos, sin generar overhead excesivo que consuma más energía que beneficio.
- En modelos de entrenamiento, usar paralelismo de datos (data parallelism) cuando sea viable para amortizar coste entre nodos, pero tener cuidado con la comunicación inter-nodos que puede introducir latencias y consumo extra.

Uso de aceleradores (GPU, TPU). Ejecución en horas con mejor mix energético sostenible

- Selección de tipos de aceleradores apropiados: GPUs modernas, TPUs, hardware especializado si disponible. Verificar eficiencia energética por flops / watt.
- Planificar entrenamiento o inferencia en horarios en los que la red eléctrica tenga mayor proporción de fuentes renovables (por ejemplo mañanas/noches según región).
- En la UE, algunas regulaciones también piden transparencia sobre energía consumida y origen energético, lo que puede incentivar esta práctica. Ej: Reglamento (UE) 2024/1689, por el que se establecen normas armonizadas en materia de inteligencia artificial, exige desglose del consumo energético.

Balanceo de cargas y programación eficiente

- Distribuir tareas de modo que ningún nodo esté infra-o sobreutilizado.
- Uso de técnicas de scheduling que consideren eficiencia: agrupar tareas similares, evitar cambios frecuentes de contexto, minimizar movimientos de datos.

- Aprovechar batch processing en inferencia para reducir overhead.

Infraestructura como código (Infrastructure as Code – IaC)

El uso de **Infrastructure as Code (IaC)** permite definir y gestionar infraestructuras de computación —instancias, redes, almacenamiento, aceleradores hardware y políticas de escalado— mediante código versionado, replicable y auditable. Esta práctica resulta especialmente relevante en la ejecución de modelos de IA, donde pequeñas variaciones en la infraestructura pueden afectar al rendimiento, consumo energético y resultados obtenidos.

La incorporación de IaC aporta los siguientes beneficios clave:

- **Reproducibilidad de la ejecución**: Permite recrear de forma exacta la infraestructura utilizada para el entrenamiento o inferencia, incluyendo tipos de instancias, número de nodos, aceleradores (GPU/TPU) y configuraciones de red.
- **Control de versiones de la infraestructura**: Al igual que ocurre con las librerías y el código, la infraestructura queda versionada, facilitando auditorías, rollback y comparación entre configuraciones.
- **Optimización energética y de costes**: IaC facilita la definición de políticas de autoescalado, apagado automático de recursos inactivos y selección explícita de instancias optimizadas para eficiencia energética.
- **Seguridad y trazabilidad**: Las configuraciones quedan documentadas como código, permitiendo revisiones, controles de acceso y cumplimiento normativo de forma sistemática.

Entre las herramientas comúnmente utilizadas para IaC en entornos de IA se encuentran **Terraform**, **AWS CloudFormation**, **Azure Bicep**, **Pulumi** o **Ansible**, que permiten integrar la provisión de infraestructura dentro de pipelines de CI/CD.

3. Configuración de software y hardware para la eficiencia energética

3.1. Optimización en hardware

Selección de instancias en la nube vs. on-premise

- Factores clave: coste energético local, mix de generación eléctrica de la zona, eficiencia del centro de datos, latencia, coste de mantenimiento.

- La elección entre infraestructuras en la nube y sistemas on-premise debe basarse en el patrón de uso y en los requisitos del caso de uso. La nube resulta más adecuada cuando la carga de trabajo es variable o escalable, ya que permite ajustar dinámicamente los recursos y evitar el sobredimensionamiento, lo que reduce el consumo energético asociado a infraestructuras infrautilizadas. Además, es especialmente ventajosa para proyectos en fases tempranas, pilotos o casos de uso con picos de demanda intermitentes.
- Por el contrario, un sistema on-premise puede ser más eficiente cuando las cargas de trabajo son estables, intensivas y predecibles en el tiempo, y cuando la organización dispone de un centro de datos propio optimizado energéticamente. En estos escenarios, la amortización del hardware y el control directo sobre la infraestructura pueden traducirse en un consumo energético más constante y, potencialmente, más eficiente por unidad de cómputo.

Ajustes de GPU/TPU y CPU para consumo eficiente

- Usar modos de bajo consumo cuando no sea necesario todo el rendimiento; ajustar frecuencia de reloj, uso de memoria.
- Evitar sobreespecificar hardware: mejor usar hardware con las características suficientes para cumplir los requisitos en vez de hardware sobredimensionado que esté subutilizado.
- Actualizar drivers / firmware que permiten optimizaciones energéticas.

Técnicas de dynamic voltage and frequency scaling (DVFS)

- DVFS permite bajar o subir la frecuencia de CPU/GPU y su voltaje asociado según carga de trabajo, reduciendo consumo energético cuando la carga es baja o moderada.
- En hardware moderno se puede configurar para que la frecuencia se ajuste automáticamente / manualmente. Se debe medir impacto en latencia o throughput.

3.2. Optimización en software

Bibliotecas optimizadas (cuDNN, TensorRT, ONNX Runtime)

- Usar bibliotecas que aprovechan las capacidades hardware para acelerar cálculo, optimizar operaciones de convolución, etc., de forma que se reduce tiempo de cálculo y por tanto energía consumida.
- ONNX Runtime, TensorRT, etc., ofrecen optimizaciones de inferencia, de fusión de operaciones, de uso de memoria.

Monitoreo y métricas de eficiencia (GPU utilization, FLOPs, kWh)

- Tener métricas de utilización de hardware, uso de energía, número de operaciones en punto flotante (FLOPs), para evaluar cuánta energía se consume por unidad de tarea (por ejemplo por batch),
- Usar herramientas de perfilado: por ejemplo monitorear el uso de GPU/CPU, temperaturas, uso de memoria; registrar consumo energético si el hardware lo soporta.
- En la especificación CTN-UNE 71/SC 42/GT 1 "Evaluación de la eficiencia energética de los sistemas de inteligencia artificial" , desarrollada como parte del PNAV, se da un listado exhaustivo de métricas y de herramientas de perfilado.

4. Ejecución distribuida de modelos IA

4.1. Arquitecturas de ejecución distribuida (data parallelism, model parallelism, federated learning)

- **Data Parallelism**: consiste en replicar todo el modelo en múltiples nodos (o GPUs), repartir los datos entre ellos, hacer forward+backward y al final sincronizar los gradientes. Es sencillo de implementar, tiene buen escalado si la comunicación entre nodos no se convierte en cuello de botella.
  Un estudio reciente que compara frameworks como Horovod, PyTorch-DDP y DeepSpeed muestra que para redes convolucionales (ResNet50, ResNet101, ResNet152) entrenadas sobre ImageNet, se puede obtener una eficiencia de paralelización (“parallel efficiency”) de más del 0.85 usando hasta 256 GPUs, y alrededor de 0.75-0.80 con 1024 GPUs, dependiendo del tamaño del modelo y del loader de datos.
- **Model Parallelism / Mixture of Experts (MoE)**: cuando el modelo es demasiado grande para caber en una GPU, o cuando ciertas partes pueden paralelizarse de manera especializada. Por ejemplo, el trabajo *DeepSpeed-MoE* muestra que se puede entrenar modelos MoE mucho más grandes con mejoras en costo de inferencia y de entrenamiento comparado a modelos densos de equivalente calidad: mejoras de hasta 4.5× más rápidos y 9× más económicos en inferencia para ciertos modelos grandes.
  Otro artículo, *Hybrid Tensor-Expert-Data Parallelism in MoE Training*, presenta un algoritmo híbrido (combinando paralelismo de datos, de tensores y de expertos) para entrenar modelos MoE gigantes, con optimizaciones de comunicación que reducen el movimiento de datos innecesario.

Federated Learning (FL): permite entrenar modelos de forma distribuida en múltiples dispositivos finales (edge, IoT, móviles), evitando la centralización de los datos. Este enfoque

presenta ventajas claras en términos de privacidad y cumplimiento normativo, pero **no es universalmente eficiente ni recomendable en todos los escenarios**. Su idoneidad depende de múltiples factores técnicos, operativos y energéticos.

Cuándo Federated Learning sí es recomendable

FL resulta adecuado principalmente cuando se cumplen una o varias de las siguientes condiciones:

1. Datos altamente sensibles o sujetos a restricciones regulatorias

FL es especialmente útil cuando:

- Los datos no pueden abandonar el dispositivo o dominio local (por ejemplo, datos sanitarios, financieros, industriales).
- Existen restricciones legales o contractuales que limitan la transferencia de datos (GDPR, soberanía del dato).

En estos casos, el coste energético adicional del entrenamiento distribuido puede estar justificado por los beneficios en privacidad y cumplimiento.

2. Gran volumen de datos distribuidos y alto coste de transferencia

FL es eficiente cuando:

- El volumen de datos crudos es elevado.
- El envío de datos al servidor central tendría un coste energético y de red superior al envío periódico de parámetros o gradientes del modelo.

Este escenario es típico en dispositivos móviles o sensores con generación continua de datos.

3. Modelos relativamente ligeros o con actualizaciones parciales

FL es más viable cuando:

- Los modelos son compactos (pocos parámetros).
- Se utilizan técnicas como actualización de capas parciales, compresión de gradientes o aprendizaje incremental.

Esto reduce el coste computacional y el consumo energético en los nodos participantes.

4. Entrenamiento no crítico en tiempo real

FL funciona mejor cuando:

- No se requiere convergencia rápida.
- Se tolera latencia en la sincronización de modelos.
- El entrenamiento se realiza en ventanas temporales amplias (por ejemplo, nocturnas).

Esto permite programar ejecuciones en periodos de menor carga energética o mayor disponibilidad de energía renovable.

Cuándo Federated Learning no es recomendable

Existen escenarios donde FL puede resultar claramente ineficiente o contraproducente:

1. **Alta heterogeneidad de dispositivos**

FL se vuelve problemático cuando:

- Los dispositivos participantes tienen capacidades muy dispares (CPU, memoria, batería).
- Algunos nodos se convierten en *stragglers*, ralentizando el proceso global.

Esto incrementa el tiempo de entrenamiento, el número de rondas necesarias y el consumo energético total.

2. **Modelos grandes o muy profundos**

FL no es eficiente cuando:

- El modelo tiene millones o miles de millones de parámetros.
- El coste de transmitir gradientes o pesos supera al coste de mover los datos.

En estos casos, la comunicación se convierte en el principal cuello de botella energético.

3. **Redes de comunicación inestables o costosas**

FL es poco recomendable si:

- La conectividad es intermitente.
- El ancho de banda es limitado.
- El coste energético de la comunicación es alto (por ejemplo, redes móviles de baja eficiencia).

La sobrecarga de comunicación puede anular cualquier ventaja de evitar la centralización de datos.

4. Necesidad de control estricto o trazabilidad completa

FL complica:

- La auditoría completa del proceso de entrenamiento.
- La reproducibilidad exacta de resultados.
- El control fino de la calidad de los datos.

En entornos donde la trazabilidad y la reproducibilidad son críticas (por ejemplo, validaciones regulatorias estrictas), FL puede introducir complejidad excesiva.

4.2. Herramientas y frameworks (Horovod, DeepSpeed, Ray, PyTorch DDP)

- **DeepSpeed**, desde su módulo MoE, ha demostrado no sólo escalabilidad sino también reducción del coste de inferencia y de entrenamiento comparado con modelos densos equivalentes, gracias a su uso de experto-capacidades, sparsidad y optimización de memoria.
- **Frameworks comparativos**: como se menciona en *Large scale performance analysis of distributed deep learning frameworks …* se ha comparado Horovod, DeepSpeed y PyTorch-DDP en entrenamiento de redes grandes, midiendo throughput, eficiencia de escalado, rendimiento en validación vs tamaño de batch, etc.
- Algunas implementaciones de federated learning integran compresión de comunicación, selección adaptativa de clientes, offloading, etc., para mejorar la eficiencia energética Ejemplo: *AutoFL* optimiza tiempo de convergencia y eficiencia energética considerando heterogeneidad del sistema.

4.3. Casos de uso y buenas prácticas

Antes de escalar la ejecución de modelos de IA en entornos distribuidos, es fundamental realizar **benchmarking y perfilado sistemático** del sistema, con el objetivo de identificar cuellos de botella y evaluar la eficiencia real del paralelismo introducido.

La literatura sobre sistemas paralelos destaca que un escalado eficiente no depende únicamente del número de nodos o aceleradores disponibles, sino de una correcta **caracterización previa del comportamiento computacional y de comunicación** del algoritmo y su entorno de ejecución. En particular, se recomienda medir:

- Uso de CPU, GPU y memoria en cada nodo.

- Latencia de ejecución y tiempos de sincronización.
- Ancho de banda efectivo y volumen de datos intercambiados entre nodos.
- Impacto de la comunicación frente al cómputo en el rendimiento global.

Tal como se describe en estudios clásicos sobre planificación de tareas en sistemas paralelos, la ausencia de este análisis previo puede provocar **escalados ineficientes**, donde el incremento de recursos no se traduce en mejoras proporcionales de rendimiento, e incluso incrementa el consumo energético total debido a sobrecostes de sincronización y comunicación.

Por tanto, las buenas prácticas recomiendan:

- Analizar el patrón de paralelismo del modelo (data parallelism, model parallelism, híbrido).
- Identificar fases dominadas por cómputo frente a fases dominadas por comunicación.
- Ajustar la granularidad de las tareas y la estrategia de planificación antes de aumentar el número de nodos.

Este enfoque permite tomar decisiones informadas sobre el escalado, maximizando el rendimiento y evitando configuraciones que resulten energéticamente ineficientes.

5. Recomendaciones prácticas y casos de aplicación

5.1. Checklist para configurar entornos de ejecución eficientes

Aquí una lista de control que puede usarse antes de entrenar / desplegar:

- Verificar versiones de librerías, dependencias, compatibilidad con hardware.
- Usar contenedores o entornos virtuales reproducibles.
- Medir consumo energético base con configuración estándar.
- Aplicar técnicas de optimización del modelo (cuantización, pruning, distillation).
- Seleccionar hardware acorde, evitar infrautilización.
- Planificar ejecución en periodos de mayor energía renovable.
- Usar frameworks optimizados.
- En caso de uso distribuido o blockchain, elegir mecanismos de consenso de bajo consumo.
- Documentar todas las configuraciones, métricas, versiones.

5.2. Guía rápida de optimización de modelos

- Analizar perfil del modelo: partes costosas computacionalmente, uso de memoria, operaciones que se puedan optimizar (convoluciones, atención, etc.).
- Aplicar pruning: eliminar neuronas o conexiones sin mucho impacto en precisión.
- Cuantización: reducir precisión (por ejemplo de FP32 a FP16, INT8) si se tolera.
- Distillation: entrenar un modelo más ligero que imite a otro más pesado.
- Sparsity: usar modelos que exploten estructuras parciales de ceros.
- Validar la pérdida de precisión vs ahorro energético en escenarios reales.

5.3. Caso práctico: despliegue distribuido eficiente con Blockchain sostenible

Un posible caso práctico podría ser:

- Entrenamiento federado de un modelo (por ejemplo para salud o IoT) donde los nodos locales entrenan con sus propios datos, usando versiones optimizadas de modelo ligero.
- Uso de Blockchain permissionada para coordinar las actualizaciones del modelo, verificar integridad, registrar auditoría.
- Ejecución del entrenamiento/inferencia en momentos del día con alto porcentaje renovable eléctrico.
- Monitorización constante de consumo energético, métricas de rendimiento.
- Evaluación comparativa: costo energético y rendimiento vs solución centralizada convencional.