PNAV

PROGRAMA NACIONAL
DE ALGORITMOS
VERDES

Execution Configuration Guide

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

accenture

Funded by the European Union - NextGenerationEU. However, the views and opinions expressed are solely those of the author(s) and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them

Index

Introduction

The aim of this guide is to provide a clear and practical methodological framework for configuring, training, deploying and operating artificial intelligence (AI) models efficiently, reproducibly, safely, and with minimal environmental impact. It targets developers, ML engineers, DevOps teams, infrastructure managers, and AI project managers. The idea is that they can apply specific good practices in installation, versioning, hardware, software, distributed execution, and consider technologies such as Blockchain in their design, always with energy sustainability criteria.

1. Principles of sustainability and efficiency in AI execution

Some fundamental principles that should guide the configuration and operation of AI:

- **Energy transparency**: documenting, measuring and reporting the energy consumption of the models ("energy footprint"). In the European Union, the recently enacted EU AI Act requires technical documentation that includes a breakdown of energy consumption for General-Purpose AI Models.
- **Reproducibility and traceability**: training is not enough - you must be able to replicate the results, including exact versions of libraries, environments, and hardware/software configuration.
- **Proactive optimization**: Apply model optimization techniques (pruning, quantization, distillation, sparsity) before deploying to reduce consumption without significantly degrading performance.
- **Use of clean energy**: choose suppliers or locations with renewable sources, prefer times of the day when the electricity grid has a better renewable mix.
- **Efficiency over scale**: more is not always better; distribute loads, parallelize only when it contributes clearly, avoid overcapacity.
- **Regulatory and ethical compliance**: comply with European directives, emerging standards, local energy efficiency and emissions regulations.

2. AI Model Execution Configuration Methodologies

This block addresses the technical mechanisms by which it is ensured that the execution of models is reproducible, safe, optimized and with controlled version.

2.1. Installation and versioning of libraries

Virtual environments and containers (conda, venv, Docker)

- Using virtual environments (venv, conda) or containers (Docker) allows you to isolate dependencies, avoid conflicts, ensure that specific versions are maintained.
- Docker also makes it easy to pack not only libraries, but also operating system versions, drivers, etc., which improves reproducibility.

Versions control (pip freeze, poetry, requirements.txt, lockfiles)

- Registering exact versions of libraries is essential: a requirements.txt with fixed versions or using poetry.lock or Pipfile.lock to make all developers/servers run with identical versions.
- Hashes of package files (as pip-freeze or poetry do) can also be used to prevent unexpected changes.

Compatibility and reproducibility

- Check compatibility between dependent libraries (e.g. CUDA versions, GPU drivers, tensor library versions) to avoid consumption spikes due to incompatibilities that trigger recalculations or inefficient use.
- Maintain reproducible test environments for benchmarking consumption, performance and verify that optimizations do not degrade accuracy or introduce bugs.

Software Lifecycle Management

CI/CD Practices

- Integrate automatic tests that verify that changes in code or libraries do not unduly increase consumption: for example, performance tests, GPU/CPU usage tests, energy tests if possible.
- Automate reproducible deployments: model versions, binary artifacts, containers, etc.

Integration with repositories and library registration

- Use internal package repositories where approval of new versions is controlled (e.g. an internal PyPI repository or mirror).

- Record libraries in use, approved versions, update history, so that you can revert if a new version introduces energy regression or failures.

Security and traceability of dependencies

- Dependency scanning for vulnerabilities.
- Package signature verification.
- Traceability: knowing which version of which library was used to generate a particular model, so that it can be reproduced or audited later.

2.2. Planning and distribution of tasks in algorithms

Parallelization strategies (multi-threading, multi-processing)

- Adequate parallelization to take advantage of multiple CPUs, cores, threads, without generating excessive overhead that consumes more energy than profit.
- In training models, use data parallelism when feasible to amortize cost between nodes, but be careful with inter-node communication that can introduce latencies and extra consumption.

Use of accelerators (GPU, TPU). Execution in hours with the best sustainable energy mix

- Selection of appropriate accelerator types: modern GPUs, TPUs, specialized hardware if available. Verify energy efficiency per flops/watt.
- Plan training or inference at times when the electricity grid has a higher proportion of renewable sources (for example, mornings/evenings depending on the region).
- In the EU, some regulations also call for transparency in energy consumed and energy sources, which can incentivize this practice. E.g.: Regulation (EU) 2024/1689, which establishes harmonized standards in the field of artificial intelligence, requires a breakdown of energy consumption.

Load balancing and efficient scheduling

- Distribute tasks so that no node is under- or overused.
- Use scheduling techniques that consider efficiency: group similar tasks, avoid frequent context changes, minimize data movements.
- Leverage batch processing in inference to reduce overhead.

Infrastructure as Code (IaC)

Using **Infrastructure as Code (IaC)** allows you to define and manage compute infrastructures—instances, networks, storage, hardware accelerators, and scaling policies—using versioned, replicable, and auditable code. This practice is especially relevant in the execution of AI models, where small variations in infrastructure can affect performance, energy consumption and results obtained.

The addition of IaC brings the following key benefits:

- **Execution reproducibility**: Allows you to accurately recreate the infrastructure used for training or inference, including instance types, number of nodes, accelerators (GPU/TPU), and network configurations.
- **Infrastructure version control**: As with libraries and code, the infrastructure is versioned, facilitating audits, rollback, and comparison between configurations.
- **Energy and cost optimization**: IaC makes it easy to define policies for autoscaling, automatic shutdown of idle resources, and explicit selection of energy-optimized instances.
- **Security and traceability**: Configurations are documented as code, allowing for systematic reviews, access controls, and compliance.

Among the tools commonly used for IaC in AI environments are **Terraform**, **AWS CloudFormation**, **Azure Bicep**, **Pulumi** or **Ansible**, which allow the integration of infrastructure provision within CI/CD pipelines.

3. Software and hardware configuration for energy efficiency

3.1. Hardware optimization

Cloud vs. on-premises instance selection

- Key factors: local energy cost, electricity generation mix in the area, data center efficiency, latency, maintenance cost.
- The choice between cloud infrastructures and on-premises systems should be based on the usage pattern and the requirements of the use case. The cloud is best suited when the workload is variable or scalable, as it allows resources to be dynamically adjusted and oversizing avoided, reducing energy consumption associated with underutilized infrastructures. In addition, it is especially advantageous for projects in early phases, pilots or use cases with intermittent peaks in demand.

- Conversely, an on-premises system can be more efficient when workloads are stable, intensive, and predictable over time, and when the organization has its own energy-optimized data center. In these scenarios, hardware payback and direct control over infrastructure can translate into more consistent and potentially more efficient power consumption per unit of compute.

GPU/TPU and CPU settings for efficient consumption

- Use low-power modes when full performance is not needed; adjust clock frequency, memory usage.
- Avoid overspecifying hardware: Better to use hardware with enough features to meet the requirements rather than oversized hardware that is underutilized.
- Update drivers/firmware that allow power optimizations.

Dynamic voltage and frequency scaling techniques (DVFS)

- DVFS allows you to lower or raise the CPU/GPU frequency and its associated voltage depending on the workload, reducing power consumption when the load is low or moderate.
- On modern hardware it can be set so that the frequency is adjusted automatically/manually. Impact must be measured in latency or throughput.

3.2. Software optimization

Optimized libraries (cuDNN, TensorRT, ONNX Runtime)

- Use libraries that take advantage of hardware capabilities to speed up calculation, optimize convolution operations, etc., so that calculation time is reduced and therefore energy consumed.
- ONNX Runtime, TensorRT, etc., offer optimizations for inference, operation merging, memory usage.

Monitoring and efficiency metrics (GPU utilization, FLOPs, kWh)

- Have metrics on hardware utilization, energy usage, number of floating-point operations (FLOPs), to evaluate how much energy is consumed per unit of task (e.g. per batch),
- Use profiling tools: e.g. monitor GPU/CPU usage, temperatures, memory usage; record power consumption if the hardware supports it.

- The CTN-UNE 71/SC 42/GT 1 specification "Evaluation of the energy efficiency of artificial intelligence systems", developed as part of the PNAV, provides an exhaustive list of profiling metrics and tools.

4. Distributed AI Model Execution

4.1. Distributed execution architectures (data parallelism, model parallelism, federated learning)

- **Data Parallelism**: consists of replicating the entire model on multiple nodes (or GPUs), distributing the data among them, forward+backward and finally synchronizing the gradients. It is simple to implement; it has good scaling if communication between nodes does not become a bottleneck. A recent study comparing frameworks such as Horovod, PyTorch-DDP and DeepSpeed shows that for convolutional networks (ResNet50, ResNet101, ResNet152) trained on ImageNet, a parallelization efficiency of more than 0.85 can be obtained using up to 256 GPUs, and around 0.75-0.80 with 1024 GPUs, depending on the size of the model and the data loader.
- **Model Parallelism / Mixture of Experts (MoE):** When the model is too large to fit on a GPU, or when certain parts can be parallelized in a specialized way. For example, the *DeepSpeed-MoE* work shows that much larger MoE models can be trained with improvements in inference and training cost compared to dense models of equivalent quality: improvements of up to 4.5× faster and 9× cheaper in inference for certain large models. Another paper, *Hybrid Tensor-Expert-Data Parallelism in MoE Training*, presents a hybrid algorithm (combining data, tensor, and expert parallelism) for training giant MoE models, with communication optimizations that reduce unnecessary data movement.

Federated Learning (FL): allows models to be trained in a distributed way on multiple end devices (edge, IoT, mobile), avoiding data centralization. This approach has clear advantages in terms of privacy and compliance, but **it is not universally efficient or recommended in all scenarios**. Its suitability depends on multiple technical, operational and energy factors.

When Federated Learning is recommended

FL is suitable primarily when one or more of the following conditions are met:

1. Data that is highly sensitive or subject to regulatory restrictions

FL is especially useful when:

- Data cannot leave the device or local domain (e.g., healthcare, financial, industrial data).

- There are legal or contractual restrictions that limit data transfer (GDPR, data sovereignty).

In these cases, the additional energy cost of distributed training may be justified by the privacy and compliance benefits.

2. Large volume of distributed data and high transfer cost

FL is efficient when:

- The volume of raw data is high.
- Sending data to the central server would have a higher energy and network cost than periodically sending parameters or gradients of the model.

This scenario is typical in mobile devices or sensors with continuous data generation.

3. Relatively light or partially upgraded models

FL is most viable when:

- The models are compact (few parameters).
- Techniques such as partial layer updating, gradient compression, or incremental learning are used.

This reduces computational cost and power consumption at participating nodes.

4. Non-Critical Real-Time Training

FL works best when:

- No rapid convergence is required.
- Latency is tolerated in model synchronization.
- Training takes place in wide-time windows (e.g. overnight).

This allows for scheduling runs in periods of lower energy load or greater availability of renewable energy.

When Federated Learning is not recommended

There are scenarios where FL can be clearly inefficient or counterproductive:

1. **High device heterogeneity**

FL becomes problematic when:

- The participating devices have very different capacities (CPU, memory, battery).
- Some nodes become *stragglers*, slowing down the overall process.

This increases the training time, the number of rounds needed and the total energy consumption.

2. Large or very deep models

FL is not efficient when:

- The model has millions or billions of parameters.
- The cost of transmitting gradients or weights exceeds the cost of moving the data.

In these cases, communication becomes the main energy bottleneck.

3. Unstable or expensive communication networks

FL is not recommended if:

- Connectivity is intermittent.
- Bandwidth is limited.
- The energy cost of communication is high (e.g. low-efficiency mobile networks).

Communication overhead can negate any advantage of avoiding data centralization.

4. Need for strict control or full traceability

FL complicates:

- The complete audit of the training process.
- The exact reproducibility of results.
- Fine control of data quality.

In environments where traceability and reproducibility are critical (e.g., strict regulatory validations), FL can introduce excessive complexity.

4.2. Tools and frameworks (Horovod, DeepSpeed, Ray, PyTorch DDP)

- **DeepSpeed**, from its MoE module, has demonstrated not only scalability but also reduced inference and training costs compared to equivalent dense models, thanks to its use of expertise-capabilities, sparsity and memory optimization.
- **Comparative Frameworks**: As mentioned in *Large scale performance analysis of distributed deep learning frameworks...* Horovod, DeepSpeed and PyTorch-DDP have been compared in training large networks, measuring throughput, scaling efficiency, performance in validation vs batch size, etc.
- Some federated learning implementations integrate communication compression, adaptive client selection, offloading, etc., to improve energy efficiency. Example: *AutoFL* optimizes convergence time and energy efficiency by considering system heterogeneity.

4.3. Use cases and best practices

Before scaling the execution of AI models in distributed environments, it is essential to carry out **systematic benchmarking and profiling** of the system, in order to identify bottlenecks and evaluate the real efficiency of the parallelism introduced.

The literature on parallel systems highlights that efficient scaling does not depend only on the number of nodes or accelerators available, but also on a correct **prior characterization of the computational and communication behavior of** the algorithm and its execution environment. In particular, it is recommended to measure:

- CPU, GPU, and memory usage on each node.
- Execution latency and synchronization times.
- Effective bandwidth and volume of data exchanged between nodes.
- Impact of communication vs. computation on overall performance.

As described in classic studies on task planning in parallel systems, the absence of this prior analysis can lead to **inefficient scaling**, where the increase in resources does not translate into proportional performance improvements, and even increases total energy consumption due to synchronization and communication cost overruns.

Therefore, good practices recommend:

- Analyze the model's parallelism pattern (data parallelism, model parallelism, hybrid).
- Identify phases dominated by computation versus phases dominated by communication.

- Adjust the granularity of tasks and planning strategy before increasing the number of nodes.

This approach enables informed decisions about scaling, maximizing performance and avoiding configurations that are energy-inefficient.

5. Practical recommendations and application cases

5.1. Checklist for setting up efficient execution environments

Here is a checklist that can be used before training/deploying:

- Check library versions, dependencies, hardware compatibility.
- Use reproducible containers or virtual environments.
- Measure base energy consumption with standard configuration.
- Apply model optimization techniques (quantization, pruning, distillation).
- Select appropriate hardware, avoid underutilization.
- Plan execution in periods of greater renewable energy.
- Use optimized frameworks.
- In case of distributed or blockchain use, choose low-power consensus mechanisms.
- Document all configurations, metrics, versions.

5.2. Model Optimization Quick Guide

- Analyze model profile: computationally expensive parts, memory usage, operations that can be optimized (convolutions, attention, etc.).
- Apply pruning: eliminate neurons or connections without much impact on accuracy.
- Quantization: Reduce accuracy (e.g. from FP32 to FP16, INT8) if tolerated.
- Distillation: training a lighter model that imitates a heavier one.
- Sparsity: using models that exploit partial structures of zeros.
- Validate the loss of accuracy vs. energy savings in real scenarios.

5.3. Case Study: Efficient Distributed Deployment with Sustainable Blockchain

A possible use case could be:

- Federated training of a model (e.g. for health or IoT) where local nodes train on their own data, using optimized versions of the lightweight model.
- Use of permissioned Blockchain to coordinate model updates, verify integrity, log audit.
- Execution of training/inference at times of the day with a high percentage of electrical renewables.
- Constant monitoring of energy consumption, performance metrics.
- Benchmarking: Energy cost and performance vs. conventional centralized solution.