UNICO I+D Project

6G-INTEGRATION 04

6G-INTEGRATION-04-E22

Distributed processing through AI mechanisms

Financiado por la Unión Europea NextGenerationEU

GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES

Plan de Recuperación, Transformación y Resiliencia

Document properties

Document number	6G-INTEGRATION-04-E22
Document title	Distributed processing through AI mechanisms
Document responsible	Monica Villalobos, Irene Rubio, Yaima Fiallo (Capgemini Engineering)
Document editor	Yaima Fiallo, Ernesto Correa, Noel Ruiz, Miguel Juaniz, Jon Iriarte (Capgemini Engineering)
Target dissemination level	Private
Status of the document	Final
Version	1.0
Delivery date	31/12/2024

Production properties

Reviewers	Noel Ruiz (Capgemini Engineering)

Disclaimer

This document has been produced in the context of the 6G-INTEGRATION Project. The research leading to these results has received funding from the Spanish Ministry of Economic Affairs and Digital Transformation and the European Union-NextGenerationEU through the UNICO 5G I+D programme.

Contents

Financiado por la Unión Europea NextGenerationEU

GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES

Plan de Recuperación, Transformación y Resiliencia

UNICO I+D

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA
MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

List of Figures

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

List of Tables

Financiado por la Unión Europea NextGenerationEU

GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES

Plan de Recuperación, Transformación y Resiliencia

UNICO I+D

List of Acronyms

AI: Artificial Intelligence

ACO: Ant Colony Optimization

CC: Container Capacity

CPUs: Central Processing Units

DRL: Deep Reinforcement Learning

DQN: Deep Q-Network

D3QN: Dueling Double Deep Q-Learning Network

EO/IR: Electro-Optical/Infrared

GB: Gigabytes

GAs: Genetic Algorithms

GRU: Gated Recurrent Unit

HAPS High Altitude Platforms

IaaS: Infrastructure as a Service

IoT: Internet of Things

KPIs: Key Performance Indicators

LSTM: Long Short-Term Memory

Mbps: Megabits per second

ms: milliseconds

ML: Machine Learning

NTN: Non-Terrestrial Network

PER: Peripheral Error Rate

PoC: Proof of concept

RAM: Random Access Memory

RNN: Recurrent Neural Networks

SCOPs: Stochastic Combinatorial Optimization Problems

UAV: Unmanned Aerial Vehicle

UCs: Use Cases

Wh: Watt-hours

1 Introduction

In the era of distributed computing, managing computational resources efficiently presents a critical challenge, particularly within federated environments where multiple nodes must collaborate in a coordinated manner. The growing complexity of these systems necessitates the development of advanced strategies that optimize resource utilization while ensuring reliability and security in the distribution of computing tasks.

The main idea is to examine and develop Artificial Intelligence (AI) based strategies for computational distribution within federations, offering a state-of-the-art assessment and proposing innovative solutions. The focus is on exploring how AI can be used to design and refine distribution mechanisms that consider key criteria such as resource availability, security policies, task acceptance, node status, key performance indicators (KPIs), and other relevant constraints.

AI training is based on a deep analysis of the accumulated knowledge from the nodes and the federation, integrating all available control information to enable the self-management of computational task distribution. Additionally, a study and analysis of this control data is conducted, providing support to AI systems and ensuring that processing is optimally distributed in terms of reliability and energy efficiency.

An AI learning strategy is proposed, tailored to different use cases, demonstrating how this AI can learn and refine computing distribution. Finally, a theoretical proposal for a federated network architecture is presented, capable of implementing these strategies efficiently. This work concludes with comparative tests, evaluating the performance of the proposed AI strategy against traditional methods, thereby providing a clear understanding of the advantages and disadvantages of each approach in terms of computational distribution.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

2 State-of-the-Art of AI Strategies for Processing Distribution

The exponential growth in the volume of information and data generated by modern applications has significantly increased the complexity of decision-making and the allocation of computing resources. In particular, federations of nodes, which group multiple processing units distributed geographically or across different domains, face critical challenges related to the efficient distribution of tasks, taking into account aspects such as availability, security policies, and the maintenance of high performance levels. As noted in recent studies, the massive amount of data, combined with the variability in workloads, complicates real-time decision processes and requires new approaches to resource management. [1]

In this context, AI has become a key solution to address these challenges. AI-based strategies not only enable better computing allocation but also offer adaptive and autonomous mechanisms to manage resources efficiently. According to previous studies, machine learning and optimization algorithms have proven effective in handling the variability of resource demand in distributed environments, allowing decisions to be made in a more informed and autonomous way [2]. Additionally, AI's ability to process large volumes of data and adapt its models to real-time changes makes it an indispensable tool for optimizing computational infrastructure. [3]

This state-of-the-art review explores the main AI strategies that have been applied for computing distribution in federated environments, with a focus on those that facilitate self-management and the continuous adaptation of nodes. It will analyze three approaches that have demonstrated effectiveness in distributed systems: **Distributed Reinforcement Learning** [4], **Ant Colony Optimization** [5], and **Genetic Algorithms (GAs)** [6]. Each of these approaches offers distinct solutions to the problem of computing allocation, ranging from real-time adaptation to finding optimal solutions through evolutionary processes.

The article "*Deep Reinforcement Learning for Load-Balancing Aware Network Control in IoT Edge Systems*" [4], presents a Deep Reinforcement Learning (DRL) approach to control load balancing in Internet of Things (IoT) networks using edge computing systems. In distributed IoT systems, load balancing is critical to optimize both device communication and data processing on edge servers, particularly when IoT devices are dynamic and heterogeneous.

The main challenge lies in balancing communication and computing needs in an environment where IoT devices can generate large volumes of data, and these data vary depending on the movement of mobile objects within the network. To address this problem, the authors propose a reinforcement learning model with Long Short-Term Memory (LSTM) integrated into a Dueling Double Deep Q-Learning Network (D3QN). This approach enables dynamic control of IoT node clustering, balancing both the communication load on the network and the computing load on edge servers.

Experimental results, based on real data collected from a self-driving vehicle, demonstrate that the proposed approach significantly improves performance compared to traditional load-balancing

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

solutions. This model not only allows for network clustering adjustments according to the movement of mobile objects but also optimizes processing on edge servers, resulting in better network resource utilization and greater overall efficiency.

The article "*A Survey on Metaheuristics for Stochastic Combinatorial Optimization*" [5], offers a detailed review of metaheuristics applied to Stochastic Combinatorial Optimization Problems (SCOPs). SCOPs are problems in which part of the information is uncertain or dynamic, reflecting real-world situations, such as variable customer demand or unpredictable travel times.

Main points of the article:

1. **Metaheuristics**: Metaheuristics are high-level algorithmic frameworks that combine strategies to explore the solution space. They are useful in complex optimization problems, overcoming the limitations of classical methods in terms of scalability and computation time.
    - **Ant Colony Optimization (ACO)**: Based on ant behavior to find short paths by depositing pheromones.
    - **Simulated Annealing (SA)**: Inspired by the physical cooling process, with the ability to temporarily accept suboptimal solutions.
    - **Tabu Search (TS)**: An extension of local search that prevents cycles through the use of "taboo" move lists.
    - **Evolutionary Computation (EC)**: Inspired by biological evolution, it uses operators such as mutation and recombination.
2. **SCOPs**: include uncertainty in their data, such as random variables or cost functions with known probability distributions. They can be:
    - **Static**: Where decisions are made before knowing the random variables.
    - **Dynamic**: Where decisions are made continuously or in stages as random events unfold.
3. **Applications**: Metaheuristics are applied to SCOPs in a wide range of domains, such as information network management, transportation problems, and logistics, where optimization under uncertainty is crucial.
4. **Challenges and Future Research**:
    - The difficulty of obtaining exact solutions in problems with high uncertainty.
    - The use of ad hoc approximations or simulations to calculate objective functions, as in many cases there is no closed-form expression for these calculations.
    - Further studies are needed on how to tailor metaheuristics to specific SCOPs and how to handle the interactions between uncertainty and decision-making.

The article "*A Niched-Penalty Approach for Constraint Handling in Genetic Algorithms*" [6], explores a new approach for handling constraints in GAs using a technique called the niched-penalty, which

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

does not require the use of penalty parameters, making it more efficient in optimization problems with constraints.

Detailed Summary:

1. **Optimization Problems with Constraints**:
   - Real-world optimization problems often have constraints (equalities and inequalities) that complicate the search for optimal solutions.
   - Classical methods, such as penalty functions, convert these problems into unconstrained versions, but the appropriate choice of penalty parameters is critical and varies by problem, which can lead to inefficient results.
2. **Traditional Penalty Methods**:
   - Conventional approaches for handling constraints in GAs rely on penalty parameters that penalize infeasible solutions. However, finding the correct parameter values is challenging, and incorrect values can lead to premature convergence to suboptimal or infeasible solutions.
   - Some researchers have attempted to dynamically adjust these parameters or use specialized operators, but these solutions remain highly specific and are not generally applicable.
3. **New Approach: Niched-Penalty**:
   - The approach proposed by Deb and Agrawal avoids the explicit use of penalty parameters. Instead, it employs a three-phase selection mechanism:
     1. When two feasible solutions are compared, the one with a better objective function value is selected.
     2. When a feasible solution is compared to an infeasible one, the feasible one is chosen.
     3. When both solutions are infeasible, the one with the lower constraint violation is selected.
   - This approach is exclusively applicable to population-based search methods, such as GAs, and promotes diversity among feasible solutions, helping the algorithm explore new viable solutions.
4. **Experimental Results**:
   - GAs with this approach were tested on five constrained optimization problems, including engineering design problems.
   - The results show that GAs with the niched-penalty approach not only find feasible solutions more efficiently but also avoid premature convergence to non-optimal solutions, outperforming previous methods that used dynamic or multiple penalties.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

5. **Advantages of the Proposed Approach**:
   - **No Penalty Parameters**: Eliminating the need to adjust penalty parameters is a key advancement. This simplifies the algorithm's implementation and makes it more robust in different optimization problems
   - **Solution Diversity**: The use of niche techniques maintains diversity in the population, improving the algorithm's ability to explore the solution space and avoid getting stuck in local optima
   - **General Applicability**: The method applies to a wide range of constrained optimization problems without requiring specific adjustments for different problems

The following table presents a brief description of different AI techniques, some advantages, challenges and examples.

AI Techniques	Description	Advantages	Challenges	Example
DRL	Distributed reinforcement learning is a technique in which AI agents make decisions about resource allocation based on the reward obtained from previous actions. In a node federation, each node could learn to optimize its workload based on its own state (resource availability, current load, etc.) and interactions with other nodes	▪ Allows each node to dynamically adapt to changes in computing demand ▪ Scalability, as the system adapts well to large and distributed networks ▪ Can handle uncertain and dynamic environments, adjusting in real-time	▪ Requires large amounts of data to train the models ▪ The computational complexity of training can be high if the nodes are numerous. ▪ Balancing individual rewards with the global goal can be challenging	In cloud computing systems, DRL has been used to distribute tasks efficiently among servers, adapting to real-time demand fluctuations
ACO	Algorithm inspired by the	▪ Suitable for complex	▪ Convergence can be slow in	Has been used in telecommunications

Financiado por la Unión Europea NextGenerationEU · Gobierno de España · Ministerio de Asuntos Económicos y Transformación Digital · Secretaría de Estado de Telecomunicaciones e Infraestructuras Digitales · Plan de Recuperación, Transformación y Resiliencia · UNICO I+D

	behavior of ants to find optimal paths to food sources. Applied to computing distribution, this approach is used to find the best routes or task allocations among nodes, based on a heuristic that favors the most efficient routes	network optimization problems, such as federations. ▪ Its distributed structure fits well in decentralized environments, where nodes lack a full view of the system. ▪ Robust against failures, as it continuously seeks new optimal solutions when conditions change	very large networks, as the learning process is iterative ▪ Sensitive to parameter settings, such as pheromone evaporation and search intensity	network planning and distributed computing systems to balance load among nodes based on data traffic
Genetic Algorithms	Evolutionary methods that use principles of natural selection to generate optimal solutions. In the context of computing distribution, they can be used to evolve resource configurations that maximize the performance of a node federation, considering constraints such	▪ Capable of finding global optimal solutions from multiple possible combinations ▪ Suitable for scenarios with complex constraints, such as security policies and node availability ▪ Facilitates self-management through the	▪ Convergence times can be long, especially if the problem has many variables ▪ GA performance depends on the correct configuration of its operators (mutation, crossover, etc.) ▪ The computational cost of evaluating solutions can	Commonly used in cloud computing to allocate resources optimally across multiple data centers, ensuring efficient use of bandwidth and CPU

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA
MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

	as security and availability	continuous evolution of adaptive strategies	be high in federated environments with many nodes	

TABLE 1 DESCRIPTION OF AI TECHNIQUES, ADVANTAGES, CHALLENGES AND EXAMPLES

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA
MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

3 Proposal for a control data repository, search engine, and analytics capable of adequately supporting AI systems

The correct distribution of processing in a federation of computing nodes requires the collection and analysis of key data that enables artificial intelligence to make optimal decisions regarding task allocation, load balancing, and energy optimization. This data must be selected based on its relevance to the system's performance and reliability, as well as its ability to contribute to maximizing energy efficiency. The main types of data needed and their application in the AI system are defined below.

3.1. Types of Data Used

3.1.1 Node Availability Data

Real-time information about the available computing capacity of each node, including parameters such as CPU usage, memory, and storage. This data is essential for assigning processing loads in a balanced manner, avoiding overloading specific nodes. It allows AI to make informed decisions about how to distribute tasks, prioritizing nodes with the most available capacity. Data is continuously collected and used under high-demand scenarios to adjust assignments in real time, especially in situations of fluctuating workloads.

3.1.2 Network Status Data

Information about latency, available bandwidth, network congestion, and the overall state of connectivity between nodes. This type of data is crucial for avoiding bottlenecks in the network that could slow down the distribution of computing. AI algorithms use this data to avoid assigning tasks to nodes that, although they have computing resources, are located in regions of the network with high latency or low connectivity. This is more relevant in distributed and dynamic networks, such as those with mobile nodes or in heterogeneous systems where connectivity fluctuates.

3.1.3 Energy Consumption Indicators

Data on the energy consumption of each node and the energy cost associated with executing specific tasks. This data enables AI to make decisions that minimize overall energy consumption, contributing to the optimization of resource use. It is used in algorithms that seek to balance performance with energy efficiency, assigning tasks to nodes that can complete them more efficiently in energy terms. This is prioritized in contexts where energy efficiency is critical, such as in IoT infrastructures or edge computing, where energy resources may be limited.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

3.1.4 Historical Performance Data

Historical information about node performance, including success rates in task execution, response times, and overall reliability. This provides AI with a solid basis for predicting future performance and making proactive adjustments. It helps AI make decisions based on past trends, identifying nodes that consistently offer high performance and reliability. This data is valuable in scenarios where AI needs to make forecasts about task distribution or in systems with high variability in performance.

3.1.5 Security Constraints and Policies

Information about the security policies that must be met in task distribution, including restrictions on where certain data can be processed (e.g., sensitive data that can only be processed on certain nodes). AI uses this data to ensure that the distribution of workloads complies with regulations and security restrictions, avoiding the assignment of tasks to unauthorized nodes. These policies apply in federated systems operating under strict privacy regulations, such as GDPR, or in critical infrastructures.

3.2. Proposed Control Data Repository

To support AI systems, the use of a centralized control data repository is proposed, which efficiently stores and updates the aforementioned data. This repository should meet the following requirements:

- Scalability: Ability to handle large volumes of data, both in real-time and historical.
- High Availability: Ensure that data is available to AI algorithms at all times, minimizing downtime.
- Data Integrity: Guarantee the accuracy and consistency of stored data, with validation and error correction mechanisms.
- Secure Access: Implement robust security policies to protect access to the data, ensuring that only authorized users or systems can access it.

3.3. Search and Analytics Engine

To process and analyze data from the repository, a search and analytics engine capable of performing complex searches and executing real-time analyses is required. This engine should be able to:

- Search Data Based on Specific Contexts: Identify and filter relevant data based on the current condition of the nodes or the network.
- Generate Analytical Reports: Create real-time reports summarizing the current state of the system, providing key metrics such as energy consumption or network load.
- Support Real-Time Decisions: Assist AI algorithms in making decisions based on real-time analysis, optimizing computing distribution based on the most recent data.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

3.4. Data Analysis Focused on Reliability and Energy Optimization

Data analysis should focus on maximizing system reliability and energy efficiency. The collected data must be selected and analyzed based on its ability to:

- Improve Reliability: Identify failure patterns and predict potential performance issues, allowing for proactive system reconfiguration
- Energy Optimization: Provide insights that enable the reduction of energy consumption, such as identifying nodes that can execute tasks at a lower energy cost

The resulting report should include a selection of candidate data that follows these principles and should be continuously evaluated to adjust AI strategies based on the changing conditions of the system.

In the following table, the information is presented as follows:

- **Metrics**: These are the actual measurements considered for each data type.
- **DRL**: Examples of how DRL uses data to optimize computing allocation, prioritizing real-time efficiency and adaptability.
- **ACO**: Examples of how ACO adjusts routes and task allocation using available data, reinforcing decisions based on key metrics.
- **GA**: Examples of how GA selects and evolves optimized node configurations based on availability, network status, energy consumption, and historical reliability.

Data Type	Metrics	Distributed Reinforcement Learning	Ant Colony Optimization	Genetic Algorithms
Node Availability	▪ CPU Usage (%) ▪ RAM Used (GB) ▪ Storage Capacity (GB) ▪ Available CPU Cycles (GHz)	Adjusts task allocation based on the CPU, RAM, and storage availability of nodes. It assigns tasks to nodes with <70% CPU.	Reinforces paths to nodes with higher CPU and bandwidth availability. Nodes with available CPU cycles receive more "pheromones"	Uses availability data to evaluate configurations that optimize CPU and RAM usage, selecting nodes with faster processing times
Network Status	▪ Network Latency (ms) ▪ Packet Loss (%)	Assigns tasks to nodes with latencies <100 ms and less than 1%	Adjusts routes to avoid bottlenecks, assigning tasks	Optimizes configurations based on network status, minimizing

Financiado por la Unión Europea NextGenerationEU · Gobierno de España · Ministerio de Asuntos Económicos y Transformación Digital · Secretaría de Estado de Telecomunicaciones e Infraestructuras Digitales · Plan de Recuperación, Transformación y Resiliencia · UNICO I+D

	▪ Bandwidth (Mbps) ▪ Data Transfer Rate (Mbps)	packet loss, improving real-time efficiency	to nodes with lower latency and higher bandwidth	latency and maximizing data transfer rate
Energy Consumption Indicators	▪ Energy Consumption per Task (Watts) ▪ Energy Efficiency (GFlops/Watt) ▪ Energy Cost per Operation (Joules)	Optimizes the balance between performance and energy consumption, prioritizing nodes with lower consumption per task (e.g., 30 W versus 50 W)	Reinforces paths to nodes with lower cumulative energy consumption, enhancing long-term sustainability	Selects configurations that optimize energy efficiency (e.g., 150 J versus 200 J per operation), favouring more efficient nodes
Historical Performance Data	▪ Average Response Times (ms) ▪ Failure Rate (%) ▪ Success Rate (%) ▪ Reliability History (hours without failures)	Learns from historical data, assigning tasks to nodes with fewer failures and faster response times (e.g., 40 ms versus 80 ms)	Penalizes routes that include nodes with failure histories, reinforcing paths to nodes with a higher success rate in tasks (>95% success)	Evolves toward more reliable configurations based on reliability history (e.g., nodes with 500 operational hours without failures versus 200 hours)
Security Constraints and Policies	▪ Geographic Restrictions ▪ Security Permissions ▪ Regulatory Compliance (GDPR, etc.)	Avoids assigning tasks to nodes in restricted geographic locations, adapting distribution to comply with regulations such as GDPR	Adjusts routes to avoid nodes that do not comply with security policies, reinforcing paths to authorized nodes	Penalizes solutions that do not meet security regulations, evolving configurations that ensure legal and privacy compliance

TABLE 2 DATA TYPE FOR AI TECHNIQUES

Financiado por la Unión Europea NextGenerationEU · GOBIERNO DE ESPAÑA · MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL · SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES · Plan de Recuperación, Transformación y Resiliencia · UNICO I+D

4 Proposed AI learning strategy for processing distribution

To In this distributed computing environment, there are multiple compute nodes available for deploying various services, and each node's performance can vary significantly over time. Each node generates performance coefficients that reflect its capacity to run services at any given time, and these coefficients can change dynamically due to multiple factors, such as workload, resource availability, and energy efficiency. Additionally, the number of nodes in the network is not fixed: new nodes can be added or removed from the system at any moment, introducing additional complexity. The task in this scenario is to continuously and automatically identify the optimal node for deploying a service, maximizing performance and adapting to changes in the system.

To solve this problem, an AI strategy is proposed that allows the system to learn from the performance coefficients and select the ideal node for each deployment, leveraging advanced neural network techniques that can handle variable inputs and adapt to real-time changes. The proposed methods below take these dynamic characteristics into account and enable optimal and flexible evaluation of each node's performance:

1. **Recurrent Neural Networks (RNN), LSTM or Gated Recurrent Unit (GRU)**: These networks are ideal for capturing temporal dependencies in data. In the context of nodes' performance coefficients, an RNN or a more advanced variant like LSTM (Long Short-Term Memory) or GRU (Gated Recurrent Unit) can model how each node's historical performance affects its future performance. This approach is useful when patterns are observed over time, allowing the system to make informed predictions about the node with the best expected performance at a given moment.

   Advantages:

   - Temporal dependency capture: RNNs, and especially their LSTM and GRU variants, are designed to model temporal sequences, enabling the system to understand how a node's past performance impacts its future performance. This is beneficial when performance data has a strong temporal dependency.
   - Good prediction capacity for sequential data: LSTM and GRU are more effective than traditional RNNs at avoiding gradient vanishing problems, achieving better retention of relevant information over long data sequences.
   - Efficiency for moderate data streams: RNNs and their variants are generally less computationally expensive compared to more complex architectures like Transformers, especially in systems with a moderate number of nodes and performance coefficients.

   Disadvantages:

   - Challenges with variable inputs: RNNs are designed to process fixed-length sequences, which can present issues when the number of nodes changes dynamically, requiring adjustments or retraining with each significant configuration change.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

- Inferior parallel processing: RNNs process data sequentially, which limits their parallelization capability, making training slower compared to models like Transformers.
- Less adaptability to abrupt changes: Although they can adapt to temporal patterns, RNN, LSTM, and GRU models may struggle to adjust quickly to abrupt or dynamic changes in performance, such as sudden load spikes on nodes.

2. **Transformers with Attention Mechanism**: The variability in the number of nodes and their fluctuating performance levels require a flexible approach that doesn't depend on a fixed sequence of inputs. Transformers, with their attention mechanism, allow the system to dynamically prioritize nodes with the best performance conditions at any given time. Moreover, their ability to process all inputs in parallel makes them efficient and adaptable to scenarios with variable inputs, which is particularly valuable in this dynamic environment.

   Advantages:

   - Adaptability to variable inputs: The attention mechanism in Transformers allows processing a variable number of inputs, which is ideal in an environment where the number of compute nodes changes dynamically. This enables robust implementation without constant adjustments.
   - Efficient parallelization: Unlike RNNs, Transformers process all input data in parallel, significantly reducing training times and enabling quick responses to new performance data.
   - Flexible focus: The attention mechanism allows the model to "focus" on the most relevant nodes at any given moment, effectively identifying nodes with the best performance, making this process scalable and adaptive to system variations.

   Disadvantages:

   - Higher computational cost: Transformers typically require significant computational resources due to their complexity and high dimensionality, which can be a drawback in systems with limited resources.
   - Need for large training data sets: Transformers tend to be more effective with large data sets to adequately capture patterns, which can be a limitation if historical performance data is insufficient or highly variable.
   - Possible overfitting in changing environments: Due to their ability to capture complex patterns, Transformers may overfit specific training data, which could lead to suboptimal behaviour in environments with sudden or non-recurring performance changes.

3. **DRL**: In an environment where the system needs to adapt constantly and learn to autonomously select the optimal node, deep reinforcement learning is an effective option. This approach enables the system to learn an optimal node selection strategy by interacting with the environment, receiving feedback in the form of rewards or penalties. Models like DQN (Deep Q-Networks) or the Actor-Critic method are suitable for this type of task, allowing

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

the system to adjust its deployment decisions based on changes in node performance and resource availability over time.

Advantages:

- Experience-based learning: DRL models, such as DQN or Actor-Critic, learn directly from interacting with the environment and can develop adaptive policies based on rewards, adjusting autonomously to changes in node performance.
- Adaptability to dynamic changes: As the system experiences variations in node performance, DRL models can continuously adjust, optimizing deployment decisions in real time.
- Long-term optimization: DRL not only selects the optimal node instantaneously but also considers the long-term effects of its decisions, helping to avoid overloading recurrently used nodes and balancing resource utilization.

Disadvantages:

- Longer training time and complexity: Training DRL models is resource-intensive in terms of time and resources, as the system needs to perform many interactions with the environment to learn an effective policy.
- Dependence on an adequate simulation environment: Training a DRL model requires an accurate and reliable simulation environment. If this environment does not faithfully represent reality, the model may not generalize well.
- Potential convergence issues: In environments where conditions change abruptly, DRL models may take time to adjust, and in some cases, they may not converge to an optimal solution, which could affect consistency in decision-making.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

5 Theoretical proposal for a federated network architecture with the capability for intelligent processing distribution

This section describes the proposal of a federated network architecture with the capability for intelligent processing distribution, following the principles of maximum possible efficiency and reliability. It is illustrated with a Python simulation of each proposal, where, in an initial approach, **the number of nodes is assumed to be fixed**.

5.1. Model 1: LSTM in Keras (TensorFlow)

The architecture we'll use is an LSTM network that takes a time sequence of performance coefficients from multiple nodes and predicts the most suitable node for deploying a service. This model uses an LSTM layer to capture the temporal dependencies in the data, followed by dense layers for output refinement, and finally, a softmax layer that selects the optimal node from all available nodes.

Steps for the proposed architecture

1. LSTM Layer: Designed to capture the time sequence of each node's performance coefficients, which is crucial for predicting performance continuity or changes
2. Dense and Dropout Layers: Act as refinement layers, helping the model learn specific patterns and generalize better, reducing the risk of overfitting
3. Output Layer (Softmax): Assigns probabilities to each node, allowing the identification of the optimal node with the highest probability

Input Data: In this model, the input data consists of a time sequence representing the evolution of certain performance coefficients for each node. These coefficients may include metrics such as CPU usage, available memory, latency, and any other measure that reflects the node's performance status.

- **Input Structure:**
  - A matrix of shape (T, N, F):

    T: Number of time steps (temporal history) to be considered. For example, if we want the model to consider the last 10 minutes and have data for each minute, T will be 10

    N: Number of nodes in the system, each with its own set of performance coefficients

    F: Number of features or metrics for each node, such as CPU usage, memory, etc.
  - Example: If there are 5 nodes and each has two features (CPU usage and latency), and we analyse the last 10 minutes, the input would have a shape of (10, 5, 2)

Importance of Input in LSTM: Since LSTMs are well-suited for capturing patterns in sequences, this model can identify trends or patterns in node performance, such as consistently high latencies in a node before it becomes overloaded.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

Output Data: The model's output is a vector assigning a probability to each node of being the most suitable for deploying a new service or workload.

- **Output Structure:**
  - A vector of size N (number of nodes)
  - Each position in the vector represents the probability of selecting the corresponding node
  - Example: If the system has 5 nodes, the output vector could be something like [0.1, 0.3, 0.05, 0.45, 0.1], where the highest value indicates the node the model recommends

In the following table, the proposed architecture for Model 1 is explained.

Architecture	Description	Advantage	Justification
LSTM Layer	The LSTM layer with 128 units is used to capture temporal patterns in the nodes performance data	LSTMs are particularly effective for sequential data, as they have the ability to remember long-term information (important in the context of performance changes)	This approach allows the model to learn temporal trends in node performance, helping it predict the optimal node based on recent history
Dense and Dropout Layers	Dense (64 units): This layer helps reduce the dimensionality of the learned representation and focuses on key performance patterns Dropout: Adds regularization to the model, helping to reduce the risk of overfitting when training with performance data	Improve model performance by enhancing feature extraction, generalization, and adaptability. Dense layers help the model focus on key patterns, reducing complexity and enabling efficient training, while Dropout layers prevent overfitting by deactivating random neurons during training, making the model more resilient to new or changing data. This combination ensures the model can handle dynamic environments effectively while	It allows the model to generalize better, which is crucial since node availability may change over time, requiring the model to adapt

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA
MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

		maintaining controlled complexity	
Output Layer (Softmax)	The final layer uses a softmax activation function, which converts the output into probabilities for each node. Thus, the model selects the node with the highest probability as the optimal one for service deployment	Converting model outputs into probabilities for each possible class, making it ideal for selecting the optimal node in a multi-class problem. This allows the model to provide a clear, interpretable probability distribution across all available nodes, so it can choose the node with the highest probability confidently. Softmax is particularly effective in classification tasks where only one option (like a single node) needs to be chosen, ensuring a straightforward decision-making process	The categorical output is appropriate for this problem, as we aim to select a node rather than predict continuous performance values

TABLE 3 ARCHITECTURE EXPLANATION FOR MODEL 1

The following figure shows the proposed architecture for Model 1: LSTM in Keras (Tensorflow)

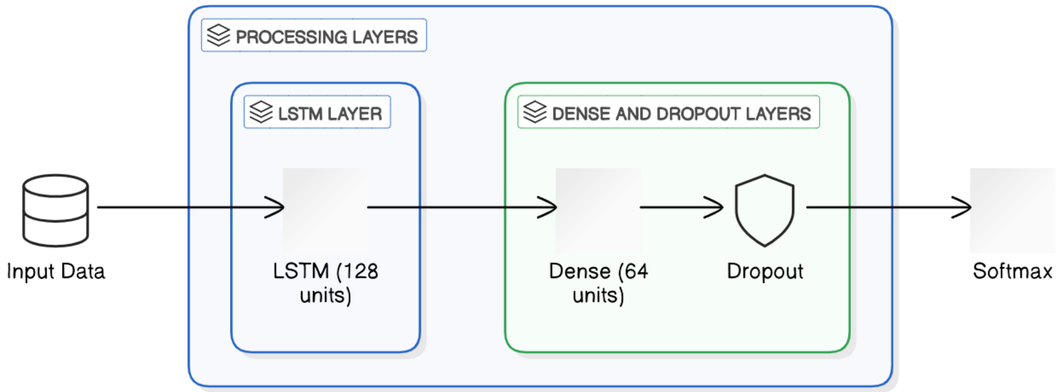


FIGURE 1 ARCHITECTURE FOR MODEL 1: LSTM IN KERAS (TENSORFLOW)

Model Training

1. Dataset Preparation:
   - Divide the data into sequences of length T
   - Normalize the performance coefficients to ensure a uniform range

- Create labels based on the optimal node observed in the next time step

2. Training Process:

   - Train using a training set that includes multiple temporal sequences
   - Optimize using a categorical cross-entropy loss function and an optimizer such as Adam

3. Evaluation:

   - Use a test set with unseen data
   - Measure metrics such as precision and recall for node selection

The implementation of these models can be found in **Annex A: Python Implementation for Model 1: LSTM in Keras (TensorFlow)**

5.2. Model 2: Transformers with Attention Mechanism

Model 2 is a basic Transformer applied to predict the optimal node based on a sequence of performance coefficients. Transformers are particularly effective at handling sequential data and can capture complex relationships between input data.

The architecture includes:

1. Transformer Encoder Layer: Learns patterns in the performance data sequence using attention mechanisms that enhance the learning of long-term dependencies
2. Average Layer: Takes the output of the encoder and averages it across the sequence, resulting in a fixed representation
3. Linear Output Layer: Predicts the optimal node based on the representation learned by the Transformer

Input Data: Like LSTM, the Transformer model receives a temporal sequence with the performance of each node but leverages its attention mechanism to focus on different aspects of the sequence based on relevance.

- **Input Structure:**
  - Like the LSTM input, a matrix of shape (T, N, F) representing node performance
  - Each entry represents the recent history of each node and its features

Importance of Input in Transformers: Transformers can handle longer sequences without losing context and can prioritize certain moments or values. For example, they can identify key instances when a node exhibits high latency.

Output Data: The output is also a probability vector indicating the suitability of each node.

- **Output Structure:**
  - A vector of size N, with a probability associated with each node

- Example Output: If there are 4 nodes, the model might produce a result like [0.2, 0.5, 0.1, 0.2], indicating that the second node is the best choice in this situation

In the following table, the proposed architecture for Model 2 is explained.

Architecture	Description	Advantage	Justification
Transformer Encoder Layer	This layer processes the data sequences through an attention mechanism, allowing the model to focus on different parts of the input based on their relevance	Transformers are effective at handling complex relationships and can capture long-term dependencies, which is crucial in contexts where node performance may vary significantly over time	The Transformer's ability to learn relationships between different points in time makes it suitable for analysing node performance sequences, optimizing the selection of the appropriate node
Average Layer	After processing the sequence, the output is averaged to obtain a unique representation that summarizes the information from all time steps	Including dimensionality reduction by summarizing the entire sequence into a compact representation, which enhances computational efficiency. It also promotes generalization by considering the overall trend across the sequence rather than focusing on individual time steps, making the model more robust to fluctuations or noisy data. Additionally, this approach helps improve model performance by smoothing out irregularities, ensuring that the model captures long-term patterns rather than being overly sensitive to short-term variations	This operation reduces dimensionality and allows the model to make a more generalized prediction about the best node, instead of focusing on a specific time instance

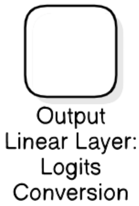
Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

Linear Output Layer	This layer converts the data representation into logits for each node, allowing a softmax activation function to be applied to obtain probabilities	Providing a direct mapping from the learned representation to the output logits, allowing the model to predict a score for each possible class (or node, in this case). This layer is computationally efficient, enabling easy integration with activation functions like softmax to convert the logits into probabilities. It simplifies the decision-making process by providing clear, interpretable outputs, making it ideal for classification tasks where selecting a single, optimal choice (e.g., the best node) is necessary. Additionally, it helps the model remain flexible and adaptable to various types of data	Provides a clear prediction of the most suitable node to deploy a service based on the observed performance coefficients

TABLE 4 ARCHITECTURE EXPLANATION FOR MODEL 2

The following figure shows the proposed architecture for Model 2: Transformers with Attention Mechanism.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D



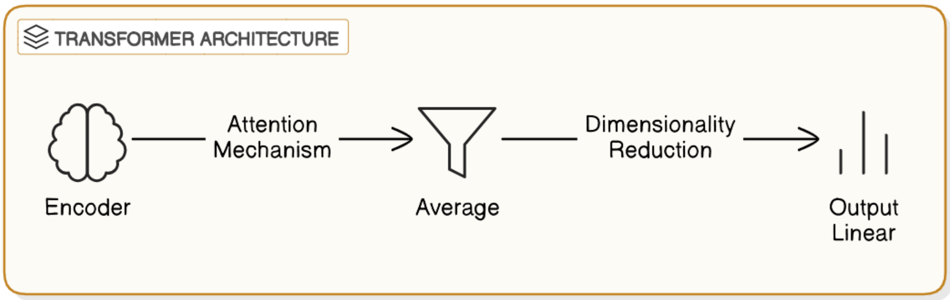


FIGURE 2 ARCHITECTURE FOR MODEL 2: TRANSFORMERS WITH ATTENTION MECHANISM

Model Training

1. Dataset Preparation:
    - Like Model 1, but a mask token can be added to handle sequences of variable length.
    - Normalize performance coefficients.
2. Training Process:
    - Train using a categorical cross-entropy loss function to maximize the probability of the optimal node.
    - Tune hyperparameters such as the number of attention heads and the model's dimension.
3. Evaluation:
    - Like Model 1, but with a focus on how the model handles long or complex sequences.

The implementation of these models can be found in **Annex B: Python Implementation for Model 2: Transformers with Attention Mechanism**

5.3. Model 3: Deep Reinforcement Learning

In this model, a DQN is trained in an environment where each node has an associated performance, and the agent must decide where to deploy a service. The goal of the agent is to maximize the total reward, which is based on the performance of the selected nodes.

Input Data: In this model, the input data represents the current state of each node at a single point in time. Unlike the previous models, which consider sequences, this model evaluates the current state

Financiado por la Unión Europea NextGenerationEU GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES Plan de Recuperación, Transformación y Resiliencia UNICO I+D

at each decision step and learns a strategy based on past experiences (rewards obtained by selecting nodes in the past).

- **Input Structure:**
  - A vector of size N × F, where N is the number of nodes and F is the number of features per node
  - This vector includes all current data for each node (e.g., current CPU usage, memory availability)
  - Example: If there are 6 nodes and each node has 3 features (CPU, memory, and latency), the input would be a vector of size 6×3=186 \times 3 = 186×3=18

Importance of Input in DQN: Since the model learns to select nodes that maximize long-term rewards, the agent uses this current information to make the optimal selection.

Output Data: the model's output is a Q-vector representing the expected reward values for each node if selected in the current state.

- **Output Structure:**
  - A vector of size N, where each position represents the expected reward value (Q-value) for selecting the corresponding node
  - Example: If there are 5 nodes, the output might look like [0.8, 0.5, 0.2, 0.9, 0.6], where the node with the highest Q-value (in this case, the fourth node) would be the optimal node to select at that moment

Architecture Explanation

1. Simulation Environment (NodeDeploymentEnv): this environment simulates a scenario where an agent selects from multiple nodes, each with a performance value represented in the state space.

   Rewards: The reward for the agent is equal to the performance of the node selected by the agent.

   State and Action Spaces: The state space is a vector that represents the performance of each node. The action space is discrete, meaning the agent can select any of the nodes.

2. DQN (Deep Q-Network): the neural network consists of fully connected layers that transform the input state into Q-values for each possible action.

   Activation: The ReLU activation function is used to add non-linearity to the model, enabling it to capture complex patterns in the data.

   Forward Propagation: The network is used to predict the Q-values given an input state, which represent the expected future rewards for taking each action (selecting a node).

3. DQN Training

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

Exploration and Exploitation: An ε-greedy strategy is employed to balance exploration (trying new actions) and exploitation (choosing the best-known actions) during the training process.

Memory: A memory buffer is used to store experiences, allowing training on mini-batches, which enhances the stability and efficiency of the learning process.

Optimization: The model is optimized by minimizing the Mean Squared Error (MSE) between the current Q-value and the target Q-value, which helps adjust the model's weights to better predict future rewards.

This architecture allows the agent to learn the optimal node selection policy through interaction with the environment, balancing exploration and exploitation to maximize the overall reward over time.

The following figure shows the proposed architecture for Model 3: DQN.

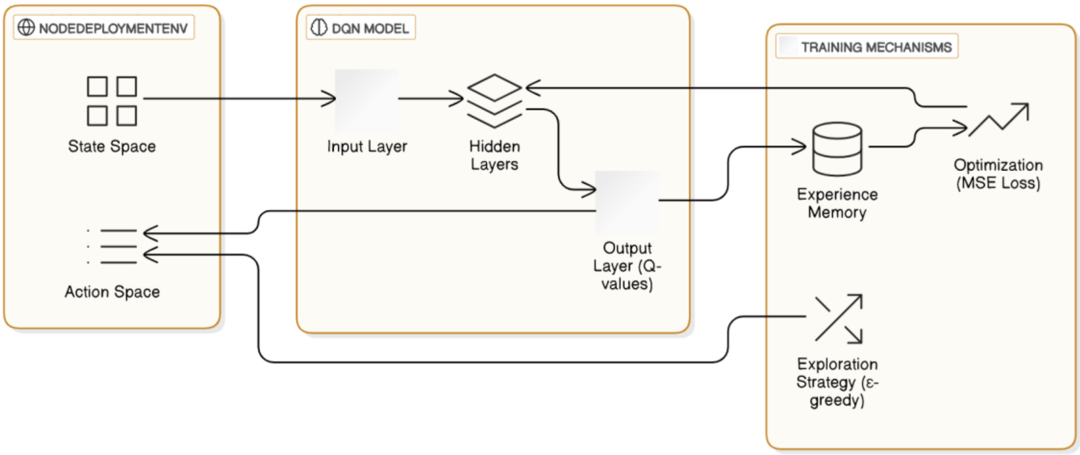


FIGURE 3 ARCHITECTURE FOR MODEL 3: DEEP Q-NETWORK

Model Training

1. Environment Preparation:
   - Design a simulated environment where the agent can interact with virtual nodes
   - Define the state space (node performance) and the action space (selecting a node)
   - Define the reward as the performance of the selected node
2. Training Process:
   - Train the agent using an ϵ-greedy strategy to balance exploration and exploitation
   - Update the model using an optimizer (e.g., Adam) and minimize the loss between the current Q-value and the target
3. Evaluation:
   - Test the agent in environments with both simulated and real data to measure its adaptability to changes.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

The implementation of these models can be found in **Annex C: Python Implementation for Model 3: DRL**

5.4. Proposed AI Strategy for Distributed Processing

The management of new policies, adaptation to node growth, and diversity are key factors influencing the choice of AI strategy for distributed processing systems. As networks grow and evolve, the need to handle dynamic policy changes, accommodate an increasing number of nodes, and manage a diverse range of features becomes crucial. Effective management of new policies ensures that the AI system remains flexible and capable of adapting to shifting network demands, such as prioritizing load balancing or penalizing overloaded nodes.

Furthermore, as the number of nodes increases, scalability becomes a critical consideration, requiring AI models that can efficiently filter and evaluate nodes without compromising performance. The diversity of node characteristics—such as varying CPU usage, memory availability, and latency—adds complexity to decision-making, necessitating an AI strategy that can adapt to different contexts and handle feature variations effectively. This dynamic interplay between policy management, node growth, and feature diversity directly shapes the AI strategy, prompting the selection of models that are both adaptable and scalable to ensure optimal performance in ever-changing environments.

Adaptability to New Policies

- The DQN model allows for the incorporation of additional rewards based on new policies. For example, if a policy favors equitable load balancing, nodes with repeated excessive loads could be penalized
- Each policy change translates into a modification of the reward, enabling the DQN to adjust its strategy without requiring complete retraining

Scalability with Node Growth

- Preselection with LSTM: As new nodes are added to the network, the LSTM efficiently filters out less suitable nodes, reducing the number of nodes the DQN needs to evaluate in detail
- DQN Adjustment: The DQN only needs to evaluate the preselected subset, maintaining efficiency even as the total number of nodes increases
- Additionally, the hybrid model can periodically update its weights to adapt to drastic changes in the network, allowing new nodes to join continuously with minimal impact on performance

Diversity of Features

- In the input phase, features can be added or removed without affecting the overall architecture. The LSTM can be trained with new temporal features, while the DQN can accept new features as part of each node's state

Financiado por la Unión Europea NextGenerationEU · GOBIERNO DE ESPAÑA · MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL · SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES · Plan de Recuperación, Transformación y Resiliencia · UNICO I+D

- If features change frequently, a periodic retraining process for the LSTM and continuous updates to the DQN rewards can be implemented to accommodate the most relevant features

Impact of Changes in the Number and Characteristics of Nodes

Model 1 and Model 2

- Adaptation to the number of nodes:
  - Both models can adjust by retraining with new data, but the computational cost may increase significantly with many nodes
  - Transformers are better suited to handle rapid variations due to their parallelization capabilities
- Change in characteristics:
  - New features would need to be added as additional dimensions in the input and retrained
  - Efficiency will depend on how many additional features are introduced and whether there is enough data

Model 3:

- Adaptation to the number of nodes:
  - The action space would change proportionally to the number of nodes
  - The agent can adjust without needing complete retraining if the simulation environment is updated dynamically
- Change in characteristics:
  - The state space can be extended without affecting the network structure, but the model will require additional learning to incorporate the new relationships

In a dynamic scenario, reinforcement learning-based models (Model 3) have an advantage, as they can adjust without restarting training completely, while supervised models would require additional data and retraining.

Given the complexity of managing dynamic environments with evolving policies, increasing numbers of nodes, and diverse feature sets, a **hybrid AI strategy combining LSTM and DQN is proposed**.

The proposal of a hybrid model can leverage the strengths of each to create a system better adapted to the dynamic and changing characteristics of the environment. This combination can be particularly useful for managing node variability, evolving policies, and the diversity of features, optimizing the distribution of computation.

This hybrid model leverages the strengths of both techniques to address the challenges of scalability, adaptability, and efficiency.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

Objective of the Hybrid Model

The hybrid model aims to leverage the sequential prediction capabilities of LSTM and the strategic adaptability of DQN to:

1. Predict performance patterns from historical sequences and pre-select the most suitable nodes
2. Optimize load distribution based on the current state and maximize long-term rewards by making adaptive decisions

Architecture and Operation of the Hybrid Model

Phase 1: Sequential Prediction with LSTM

- Input: A temporal sequence of performance data for each node, capturing CPU usage, memory, latency, bandwidth, and other relevant data
- LSTM Output: A score vector that ranks the nodes based on the probability of supporting new workloads, considering their historical performance
- Node Pre-selection: Based on the LSTM output, a subset of nodes with the highest scores is selected. This reduces the number of nodes that will proceed to the decision-making phase with the DQN and helps filter out unviable options early on

Phase 2: Optimal Selection with DQN

- Input: The current state of each pre-selected node (CPU, memory, latency, new features, or constraints)
- Decision-Making Process: The DQN uses the current state of the nodes and their past rewards to calculate the Q-value of each node based on future rewards. This enables dynamic adaptation to:
    - Changes in the environment: Such as nodes entering or exiting the network
    - New usage policies or restrictions: Changes in requirements or resource availability based on updated policies
- DQN Output: The node with the highest Q-value within the pre-selected nodes, indicating the best long-term choice in the current context

The following figure shows the proposed architecture for Hybrid Model.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

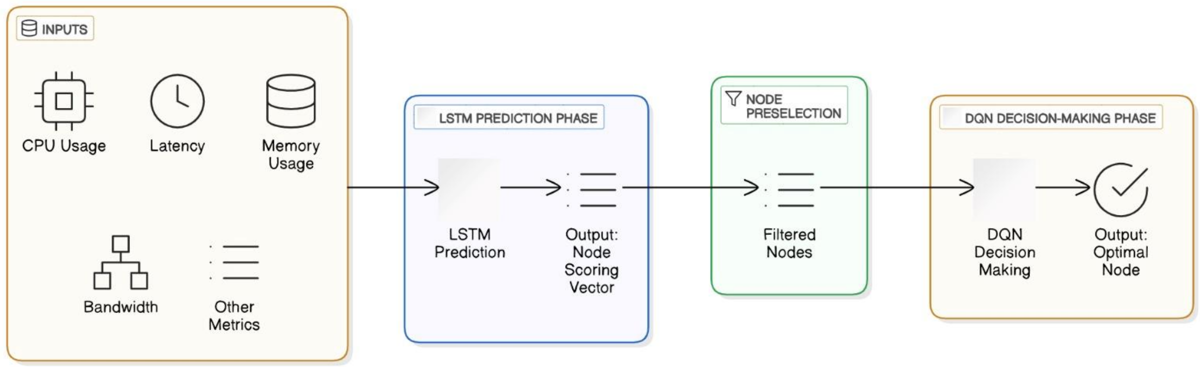


FIGURE 4 HYBRID MODEL ARCHITECTURE

Training and Optimization of the Hybrid Model

LSTM Training:

- Training Data: Use historical performance sequences of the nodes, with labels indicating whether a node successfully handled a workload based on those metrics
- Training Goal: Predict nodes with high future performance to make the pre-selection

DQN Training:

- Initial Training: Use simulated experiences where the agent receives rewards based on performance metrics such as response time, load balancing, and availability
- Continuous Learning: As more computing distributions are made, the model can use real system feedback to improve the calculation of Q-values, continuously adapting to the changing environment

Advantages of the Hybrid Model for Computing Distribution

- Decision-Making Efficiency: The combination of LSTM and DQN allows for efficient decision-making both in the short and long term, balancing prediction with adaptability
- Dynamic Adaptability: Thanks to the characteristics of DQN and continuous learning, the model can quickly adjust to the entry and exit of nodes, as well as changes in policies and features
- Scalability: Pre-selection using the LSTM ensures that the model scales well, maintaining processing speed even with an increasing number of nodes

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

6 Comparison between strategies that use AI and those that do not

Complexity Models for Task Assignment in a Federated Network of Nodes

In this type of federated computing network, each node (or machine) has unique characteristics, such as memory capacity, processing power, and power consumption, among others. These nodes are interconnected, with a central orchestrator that monitors their status in real-time to coordinate necessary tasks across the network. The orchestrator's goal is to distribute a set of services among the nodes in an optimal way, respecting specific policies and adapting to each machine's availability and load.

To optimize this process, the aim is to find a task assignment that minimizes the overall load and balances the workload among the nodes, all while meeting policy and resource constraints. The complexity of this assignment—meaning the computational or manual effort required to accomplish it—can vary considerably depending on whether it is performed by a human or by an artificial intelligence (AI) system. This analysis proposes two complexity models:

1. **Manual Assignment Model:** Assumes that a human operator performs the task assignment, evaluating and allocating each task to nodes while considering the policies and conditions of each machine.
2. **AI-Based Assignment Model:** Assumes that an AI algorithm automatically makes assignment decisions, evaluating the nodes' status and optimizing the task distribution.

Comparing these two models will allow us to analyze how the complexity of the process changes depending on the approach, helping to infer the advantages of an AI-based system in terms of scalability and efficiency in large networks.

6.1. Model 1: Manual Assignment

In the manual assignment model, a human agent distributes services among the nodes. This process may involve reviewing multiple parameters and constraints (policies and each node's status) as well as considering the specific capabilities of each machine before making the final assignment.

The complexity of this assignment can be modeled with the following formula:

Formula

$$C_m = S \times \sum_{i=1}^{N} \left(T_m \times C_i \times \left(1 + \frac{P}{N} \right) \right)$$

where:

- **S** is the number of services or tasks to be distributed
- **N** is the total number of nodes in the network

- $\mathbf{T_m}$ is the average time it takes for the operator to evaluate an assignment between a service and a node
- $\boldsymbol{C_i}$ represents a measure of the "evaluation complexity" of node i, depending on the specific characteristics of each node. This complexity could be calculated based on:
    - Memory and computing capacity: A node with higher memory or processing capacity may require more evaluation time due to the additional assignment options
    - Occupancy status: Nodes with high occupancy may increase the difficulty of finding an optimal assignment, thus increasing complexity
    - Power consumption: Depending on energy efficiency policies, power consumption may limit assignment options for certain nodes, increasing evaluation effort
- **P** represents the number of specific policies that must be met

Interpretation:

The summation within the model indicates that the operator must evaluate each node individually before assigning a task, taking into account the unique characteristics of each machine. As **N** and **S** grow, this process can become exponentially more complex, especially in networks with heterogeneous nodes. Complexity increases with a greater variety of nodes, as the operator needs to exert more effort to identify the appropriate node based on the task type and each node's requirements.

6.2. Model 2: AI-Based Allocation

In the AI-based allocation model, an algorithm autonomously makes assignment decisions. This system evaluates node parameters, computing capacity, and policy constraints more efficiently than a human operator by using optimization and prioritization techniques to reduce the number of combinations that need to be analyzed.

The complexity in this case is modeled by:

$$C_{IA} = S \times \sum_{i=1}^{N} \left(\log(N) \times T_{AI} \times \gamma \times C_i \times \left(1 + \frac{P}{N}\right)\right)$$

where:

- $\boldsymbol{T_{AI}}$ is the average time the algorithm needs to evaluate an assignment between a service and a node
- $\boldsymbol{\gamma}$ represents the efficiency factor introduced by the AI system, improving speed and accuracy in evaluating assignments
- $\boldsymbol{C_i}$ represents the evaluation complexity of node iii, which depends on individual node characteristics (such as memory, computing power, occupancy, and energy consumption, as in the manual model)

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

- The function $\boldsymbol{log(N)}$ reflects the efficiency of AI search and optimization methods, reducing complexity compared to the manual model

This model assumes that AI uses a scalable approach, making the allocation process less affected by the increase in the number of nodes. By including the term C_i, the model accounts for node diversity, allowing the AI to identify and assign tasks optimally based on each machine's capacities and current state.

6.3. Comparison between models

The following table presents a comparison between the manual model and the AI-based allocation model, highlighting the key differences in their functioning and efficiency.

Comparison variables	Model 1: Manual Assignment	Model 2: AI-Based Allocation
Scalability	Evaluation complexity increases significantly with the number of nodes and the diversity of their characteristics	Optimized algorithms keep complexity growth more manageable, enhancing scalability for large networks
Efficiency	It does not have the ability to analyse complex patterns or adapt dynamically to network needs, which limits its performance in balancing workload and meeting the network's specific requirements. This shortcoming makes the manual model insufficiently responsive to real-time demands, especially as network size and task complexity grow	Allows for more efficient task assignment by reducing evaluation time for each node, leveraging its ability to analyse patterns and optimize assignments based on individual node characteristics. This results not only in reduced workload but also in a more optimal assignment tailored to the network's needs
Policy Consideration	Both models incorporate policies in node evaluation	
	It is requiring manual intervention and making the process slower and more error-prone	Offers the advantage of automated and integrated policy processing, making verification faster and less complex

TABLE 5 COMPARISON BETWEEN THE MANUAL MODEL AND THE AI-BASED ALLOCATION MODEL

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

Interpretation and Visualization

The following graph illustrates the relative complexity of manual versus AI-based assignment for a node network, showing how each model responds to an increase in node count. Manual assignment has linear growth, reflecting a steady rise in complexity as the network expands. Meanwhile, AI-based assignment benefits from logarithmic complexity growth, highlighting its significant advantage for large networks and enabling a more efficient and effective resource management.

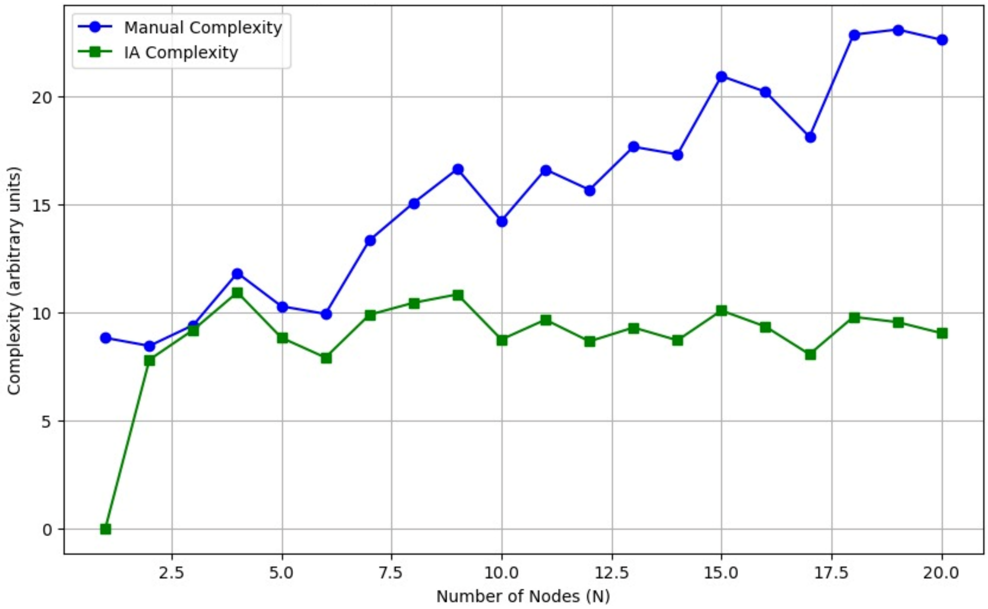


FIGURE 5 COMPARISON OF COMPLEXITY BETWEEN MANUAL ASSIGNMENT AND AI-BASED ASSIGNMENT

Conclusions Based on the Comparison Made

This analysis reveals how a manual allocation system becomes increasingly complex and less viable as the network grows, due to the linear nature of complexity in relation to the number of nodes and services. In contrast, an AI-based system allows for a more scalable and manageable allocation, thanks to its optimization capabilities and the reduced time required to evaluate each node-service. This comparison highlights the advantages of implementing AI for the management of federated networks, especially in terms of scalability and workload reduction, resulting in a more flexible and efficient system for large-scale networks.

The code related to the comparison can be found in **Annex D: Comparison of Complexity between Manual Assignment and AI-Based Assignment**

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA
MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

7 Conclusions

In this work, a hybrid model based on LSTM and DQN has been proposed for processing distribution in federated networks, a highly dynamic environment where nodes continuously vary in terms of characteristics and policies. This problem is particularly challenging, as each new node brings its own unique characteristics, and policies change constantly. Furthermore, training data frequently varies, necessitating constant modifications to the architecture and training process of neural networks. The proposed hybrid model can handle this variability and the continuous evolution of federated networks by combining the predictive capabilities of LSTM with the dynamic adaptability of DQN.

When comparing the use of artificial intelligence with traditional approaches, it has been demonstrated that AI-based solutions offer greater adaptability and scalability. While conventional methods require complete retraining and manual adjustments to accommodate changes in the network, AI models—especially those based on deep learning and reinforcement learning—enable more agile and continuous adaptation, optimizing decision-making without the need for exhaustive retraining. This real-time adaptability, combined with efficiency in managing nodes and changing characteristics, makes AI-based solutions a more robust and effective option for managing federated networks.

In summary, the combination of LSTM and DQN in a hybrid model provides an efficient and scalable solution for computing distribution in federated networks, highlighting the superiority of artificial intelligence models over traditional approaches in terms of adaptability, accuracy, and responsiveness to continuous changes in the environment.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

8 References

[1] C. Yang, Q. Huang, Z. Li, K. Liu and F. Hu, "Big Data and cloud computing: innovation opportunities and challenges," International Journal of Digital Earth, 10(1), 13–53, 2016. [Online]. Available: https://doi.org/10.1080/17538947.2016.1239771.

[2] C. Ghribi, M. Hadji and D. Zeghlache, “Energy Efficient VM Scheduling for Cloud Data Centers: Exact Allocation and Migration Algorithms,” *IEEE International Conference on Cloud Computing,* 2016.

[3] M. Al-Gumaei, M. Alrubaiee, S. Khorsandroo and M. Al-Khafajiy, “A Comparative Study of Resource Allocation Algorithms for Cloud Computing,” *Journal of Cloud Computing,* vol. 9, no. 1, pp. 1-22, 2020.

[4] Q. Liu, T. Xia, L. Cheng, M. v. Eijk, T. Ozcelebi and Y. Mao, "Deep Reinforcement Learning for Load-Balancing Aware Network Control in IoT Edge Systems," *IEEE Transactions on Parallel and Distributed Systems,* vol. 33, no. 6, doi: 10.1109/TPDS.2021.3116863, pp. 1491-1502, 2022.

[5] L. Bianchi, M. Dorigo, L. Gambardella and W. Gutjahr, "A Survey on Metaheuristics for Stochastic Combinatorial Optimization," *Natural Computing,* vol. 8, pp. 239-287, 2009.

[6] K. Deb and R. Agrawal, “A Niched-Penalty Approach for Constraint Handling in Genetic Algorithms,” *Complex Systems,* vol. 9, no. 2, pp. 111-138, 2005.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA
MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

9 Annexes

Annex A: Python Implementation for Model 1: LSTM in Keras (TensorFlow)

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# Problem definition
# We assume a system with several nodes, each with a performance sequence
num_nodos = 5          # Number of nodes
timesteps = 10         # Time steps (length of the time sequence)
features = 1           # Each node has a performance coefficient at each time point

# Simulated training data (should be replaced by real data in practice)
X_train = np.random.rand(100, timesteps, num_nodos)  # Inputs: performance sequence
y_train = np.random.randint(0, num_nodos, 100)       # Outputs: index of the node with best performance

# LSTM model construction
model = Sequential()

# LSTM layer to learn the temporal relationship between performance coefficients
model.add(LSTM(128, input_shape=(timesteps, num_nodos), return_sequences=False))

# LSTM with 128 units: processes the sequences and captures temporal patterns
```

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA
MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

```python
# Dropout layer to reduce overfitting
model.add(Dropout(0.2))

# Dense layer to refine the learned representation
model.add(Dense(64, activation='relu'))

# Another Dropout layer to improve generalization
model.add(Dropout(0.2))

# Output layer: predicts the best node with softmax function
model.add(Dense(num_nodos, activation='softmax'))

# Model compilation
# sparse_categorical_crossentropy is ideal for classification problems where classes are not one-hot encoded
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Model training
# Trained using a moderate number of epochs and a small batch size to allow convergence
model.fit(X_train, y_train, epochs=20, batch_size=16)

# Model testing
# We use test data to simulate a new sequence of performance coefficients for each node
X_test = np.random.rand(1, timesteps, num_nodos)
prediction = model.predict(X_test)
best_node = np.argmax(prediction)

print(f"The best node is: {best_node}")
```

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

Annex B: Python Implementation for Model 2: Transformers with Attention Mechanism

```
import torch
import torch.nn as nn
import numpy as np

# Definition of the Transformer model
class TransformerModel(nn.Module):
    def __init__(self, input_size, num_nodes, num_heads=2, num_layers=2, hidden_dim=128):
        super(TransformerModel, self).__init__()
        self.encoder_layer = nn.TransformerEncoderLayer(d_model=input_size, nhead=num_heads)
        self.transformer_encoder = nn.TransformerEncoder(self.encoder_layer, num_layers=num_layers)
        self.fc = nn.Linear(input_size, num_nodes)  # Final layer to predict the node

    def forward(self, x):
        x = self.transformer_encoder(x)
        x = x.mean(dim=1)  # Average across the sequence to obtain a single representation
        output = self.fc(x)  # Output layer
        return output

# Input configuration
timesteps = 10  # Length of the temporal sequence
num_nodes = 5  # Number of nodes
input_size = num_nodes  # Each node has a performance coefficient
```

```
# Model initialization
model = TransformerModel(input_size=input_size, num_nodes=num_nodes)
criterion = nn.CrossEntropyLoss()        # Loss for classification
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)      # Adam optimizer

# Simulated data for training
X_train = torch.randn(100, timesteps, input_size)   # Inputs: performance sequence
y_train = torch.randint(0, num_nodes, (100,))       # Outputs: index of the node with the best performance

# Basic model training
for epoch in range(20):
    optimizer.zero_grad()     # Reset gradients
    output = model(X_train)  # Forward pass
    loss = criterion(output, y_train)       # Calculate the loss
    loss.backward()  # Backpropagation
    optimizer.step()  # Update weights
    print(f'Epoch {epoch+1}, Loss: {loss.item()}')  # Print loss per epoch

# Prediction
X_test = torch.randn(1, timesteps, input_size)    # Simulated test data
with torch.no_grad():    # Deactivate gradients for inference
    output = model(X_test)    # Forward pass for prediction
    best_node = torch.argmax(output, dim=1).item()    # Get optimal node
    print(f"The best node is: {best_node}")     # Result
```

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

Annex C: Python Implementation for Model 3: DRL

```python
import gym
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import random
from collections import deque

# Simulation of an environment (Gym)
class NodeDeploymentEnv(gym.Env):
    def __init__(self, num_nodes=5):
        self.num_nodes = num_nodes
        self.action_space = gym.spaces.Discrete(self.num_nodes)  # Action space: select a node
        self.observation_space = gym.spaces.Box(low=0, high=1, shape=(self.num_nodes,), dtype=np.float32)
        self.state = np.random.rand(self.num_nodes)  # Initial random state

    def reset(self):
        self.state = np.random.rand(self.num_nodes)  # Reset the state
        return self.state

    def step(self, action):
        reward = self.state[action]  # Reward based on the performance of the selected node
        done = True  # Each step is terminal in this example
        return self.state, reward, done, {}
```

```python
# Deep Q-Network (DQN)
class DQN(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(DQN, self).__init__()
        self.fc = nn.Sequential(
            nn.Linear(input_dim, 128),  # Hidden layer
            nn.ReLU(),  # ReLU activation
            nn.Linear(128, output_dim)  # Output layer
        )

    def forward(self, x):
        return self.fc(x)  # Forward propagation
```

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA
MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

```python
# Training function
def train_dqn(env, model, episodes=1000, gamma=0.99, epsilon=0.1, lr=0.001):
    optimizer = optim.Adam(model.parameters(), lr=lr)  # Adam optimizer
    memory = deque(maxlen=10000)  # Memory to store experiences
    criterion = nn.MSELoss()  # MSE Loss
    batch_size = 32

    for episode in range(episodes):
        state = torch.FloatTensor(env.reset())
        done = False
        total_reward = 0

        while not done:
            if random.random() < epsilon:  # Exploration
                action = env.action_space.sample()  # Random action choice
            else:
                with torch.no_grad():
                    q_values = model(state)  # Q-values prediction
                    action = torch.argmax(q_values).item()  # Action selection based on Q-values

            next_state, reward, done, _ = env.step(action)  # Execute action
            memory.append((state, action, reward, torch.FloatTensor(next_state), done))

            if len(memory) > batch_size:  # Mini-batch training
                batch = random.sample(memory, batch_size)
                states, actions, rewards, next_states, dones = zip(*batch)

                states = torch.stack(states)
                actions = torch.LongTensor(actions)
                rewards = torch.FloatTensor(rewards)
                next_states = torch.stack(next_states)

                q_values = model(states)
                next_q_values = model(next_states)

                q_value = q_values.gather(1, actions.unsqueeze(1)).squeeze(1)
                next_q_value = rewards + gamma * torch.max(next_q_values, dim=1)[0] * (1 - torch.FloatTensor(dones))

                loss = criterion(q_value, next_q_value)  # Loss calculation
                optimizer.zero_grad()  # Reset gradient
                loss.backward()  # Backpropagation
                optimizer.step()  # Update weights

            state = torch.FloatTensor(next_state)  # Update states
            total_reward += reward  # Accumulate rewards

        print(f'Episode {episode+1}, Total Reward: {total_reward}')  # Print rewards
```

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA
MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D

```
# Training
env = NodeDeploymentEnv(num_nodos=5)
model = DQN(input_dim=env.observation_space.shape[0], output_dim=env.action_space.n)
train_dqn(env, model)

# Node selection
state = torch.FloatTensor(env.reset())
with torch.no_grad():
    q_values = model(state)  # Get Q-values for the current state
    best_node = torch.argmax(q_values).item()  # Select the node with the highest Q-value
    print(f"The best node is: {best_node}")  # Print the optimal node
```

Annex D: Comparison of Complexity between Manual Assignment and AI-Based Assignment

```
# Import the necessary libraries
import matplotlib.pyplot as plt
import numpy as np

# Parameters for complexity calculation
S = 5             # Total number of services to assign
N = np.arange(1, 21)  # Number of nodes (range from 1 to 20)
T_manual = 1.5    # Task complexity factor for manual assignment
T_IA = 0.8        # Task complexity factor for IA-based assignment
P = 10            # Number of policies to be respected
gamma = 0.5       # System efficiency factor with IA

# Individual node characteristics (capacity factors for each node)
C = np.random.uniform(0.8, 1.2, len(N))  # Simulating a range of capacities

# Calculate complexities
manual_complexity = S * T_manual * C * (1 + (P / N))
ia_complexity = S * np.log(N) * T_IA * gamma * C * (1 + (P / N))

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(N, manual_complexity, label="Manual Complexity", marker='o', color='blue')
plt.plot(N, ia_complexity, label="IA Complexity", marker='s', color='green')
plt.xlabel("Number of Nodes (N)")
plt.ylabel("Complexity (arbitrary units)")
plt.title("Comparison of Complexity between Manual Assignment and IA-Based Assignment")
plt.legend()
plt.grid(True)
plt.show()
```

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D