AI

MDPI

Article

Resource-Aware Deep Learning Deployment for IoT–Fog Environments: A Novel BSIR and RAG-Enhanced Approach

Mostafa Atlam [1,*], Gamal Attiya [1] and Mohamed Elrashidy [1,2]

[1] Department of Computer Science and Engineering, Faculty of Electronic Engineering, Menoufia University, Menouf 32951, Egypt; gamal.mahrous@el-eng.menofia.edu.eg (G.A.); mohamed.elrashedy@el-eng.menofia.edu.eg or maelrashidy@arabeast.edu.sa (M.E.)
[2] Department of Computer, Arab East Colleges, Riyadh 13531, Saudi Arabia
* Correspondence: mostafa.atlam@el-eng.menofia.edu.eg

Abstract

The proliferation of Internet of Things (IoT) devices challenges deep learning (DL) deployment due to their limited computational power, while cloud offloading introduces high latency and network strain. Fog computing provides a viable middle ground. We present a resource-aware framework that intelligently partitions DL tasks between fog nodes and the cloud using a novel Binary Search-Inspired Recursive (BSIR) optimization algorithm for rapid, low-overhead decision-making. This is enhanced by a novel module that fine-tunes deployment by analyzing memory at a per-layer level. For true adaptability, a Retrieval-Augmented Generation (RAG) technique consults a knowledge base to dynamically select the best optimization strategy. Our experiments demonstrate dramatic improvements over established metaheuristics. The complete framework boosts memory utilization in fog environments to a remarkable 99%, a substantial leap from the 85.25% achieved by standard algorithms like Genetic Algorithms (GA), Simulated Annealing (SA), and Particle Swarm Optimization (PSO). The enhancement module alone improves these traditional methods by over 13% without added computational cost. Our system consistently operates with a CPU footprint under 3% and makes decisions in fractions of a second, significantly outperforming recent methods in speed and resource efficiency. In contrast, recent DL methods may use 51% CPU and take over 90 s for the same task. This framework effectively reduces cloud dependency, offering a scalable solution for DL in the IoT landscape.

Keywords: internet of things; deep learning; fog computing; cloud computing; retrieval-augmented generation; heuristic algorithms

Check for updates

Academic Editor: Miguel Angel Cazorla

Received: 17 December 2025
Revised: 22 January 2026
Accepted: 24 January 2026
Published: 30 January 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

The Internet of Things (IoT) is unexpectedly expanding, and vast amounts of multimedia data are generated from connected devices [1,2]. Efficient processing of this data plays a vital role in extracting significant insights that can help real-time applications. But the limited computational resources of IoT devices pose a huge challenge when handling such a large-scale data stream. Traditional cloud-based services are supportive but often contribute to congestion and latency problems in the network. In response, new techniques such as fog computing are coming to the forefront in providing processing nearer to the sources of data, which can improve performance and responsiveness [3,4].

Deep learning (DL) is a significant part of IoT data analysis and insight extraction, helping reveal patterns, make predictions, and facilitate automation in smart systems [5].

Due to the massive and complex data generated by IoT devices, traditional analytical methods often fail, making DL essential for extracting valuable insights. However, deploying these models directly on IoT hardware is constrained by limited computational power, memory, and energy reserves. While cloud computing offers a potential solution, it frequently introduces prohibitive latency and bandwidth bottlenecks. To address these trade-offs, researchers are increasingly turning to decentralized paradigms, such as edge and fog computing, to position DL inference closer to the data source and optimize system performance [6,7].

To optimize the placement of deep learning layers among fog and cloud nodes, researchers have explored various strategies [8,9], such as metaheuristics and deep learning-based efficient fog node (DLEFN) methods [10]. Although cloud computing provides the computational power required to offload complex IoT tasks to remote servers [1,2], the resulting transmission latency and network congestion often degrade overall system efficiency [11,12]. To address these challenges, metaheuristic algorithms such as Genetic Algorithm (GA), Simulated Annealing (SA), and Particle Swarm Optimization (PSO) are frequently used to configure these layers according to available resources [13,14].

Although these metaheuristic algorithms can improve workload efficiency by iteratively evaluating multiple solutions, they are often computationally expensive and do not guarantee global optimality. On the other hand, the DLEFN strategy [10] takes a different approach by aligning application needs with the resources available in fog nodes. This method starts by maximizing the number of layers and then smartly reduces workloads if a node becomes overloaded. The limitation of this strategy is that prioritizing early tasks at layer maximization negatively affects the real-time performance of later task applications and is also resource-consuming.

In this study, we propose a hybrid system designed to optimize how neural network layers are distributed between fog nodes and cloud infrastructure in Internet of Things (IoT) environments. The primary objectives of this system are to minimize the amount of data sent to the cloud, thereby reducing network congestion, and to speed up data analysis. The proposed system incorporates a range of advanced methodologies, including deep learning techniques, metaheuristic algorithms, a retrieval augmented generation (RAG) framework, and a novel optimization method called Binary Search Inspired Recursive (BSIR), along with our newly proposed enhancement algorithm. Together, these components provide an integrated architecture that strengthens resource allocation and enhances the deployment of deep learning models in distributed settings across fog nodes and cloud infrastructure. The main contributions of this paper are outlined below:

- **BSIR Optimization Technique:** We propose the BSIR technique, which optimizes the distribution of deep learning layers across fog nodes and the central cloud to maximize throughput and minimize latency.
- **RAG-Based Adaptive Framework:** We introduce a RAG-based system that integrates established metaheuristic algorithms (such as GA, SA, and PSO), the DLEFN model, and the BSIR technique. Unlike static scheduling models, this system dynamically identifies the most suitable optimization strategy by jointly considering user-defined performance criteria and the inherent characteristics of the input IoT data.
- **Allocation Enhancement Algorithm:** We develop an enhancement algorithm that iteratively adjusts initial layer assignments, improving resource utilization and responsiveness beyond the baseline allocation.
- **Comprehensive Performance Assessment:** We conduct an extensive end-to-end evaluation of the proposed system. Our findings demonstrate that this integrated approach substantially strengthens real-time processing capabilities, enhances scala-

bility, and increases adaptability for distributed deep learning workloads in complex fog–cloud environments.

The proposed system is structured into three principal subsystems. The first subsystem is dedicated to acquiring media data from IoT devices, in addition to processing a user-defined prompt that specifies particular performance criteria. The second subsystem evaluates the collected inputs by analyzing the input characteristics received from the first subsystem and integrating a RAG model that chooses between the metaheuristic optimization methods and the novel optimization technique referred to as BSIR. Once the technique is chosen, it starts working, followed by the proposed enhancement algorithm, to derive the optimal layer-wise allocation of each application across fog nodes and cloud servers. After this optimal distribution is established, the third subsystem transitions to the execution stage. During this phase, the first layers of the application are processed on the selected fog nodes, and the resulting intermediate representations are transmitted to the cloud for computation by the remaining layers of the application.

Comparative studies were performed to assess the structural efficiency of the proposed system in comparison to the recent state-of-the-art algorithms. The assessments focused on measuring the improvements achieved in the distribution and migration of neural network layers across fog nodes and cloud resources in IoT environments. The experimental results demonstrate that the proposed system offers better architectural efficiency than the existing systems.

Furthermore, integrating the proposed enhancement significantly improved the utilization of available memory resources across the tested applications while remaining within safe operational limits. In the fog environment, standard implementations of GA, SA, and PSO reached a maximum memory utilization of 85.25%, whereas the enhanced versions achieved 98.75% of all available memory. The BSIR system demonstrated the highest efficiency, with memory utilization peaking at 99%. Beyond these performance gains, the enhancement standardized resource consumption patterns; GA, SA, and PSO converged at similar levels of CPU, time, and memory demands. This stabilization highlights the algorithm's effectiveness in balancing competing system resource requirements.

The paper is organized as follows: The related work is presented in Section 2. Then, Section 3 illustrates the proposed system design and its phases. Section 4 discusses and analyzes the proposed system implementation and experimental results. Section 5 provides a detailed discussion of the findings. Finally, the conclusion and future work are presented in Section 6.

2. Related Work

Many strategies have been developed to deploy deep learning models efficiently across fog and cloud nodes and address resource allocation challenges. In this section, we will cover the most recent of these developed strategies. The main purpose of these methods is to improve service by reducing execution time, maximizing the utilization of fog node resources, and enhancing workload balancing.

One of the most common strategies to address these challenges and handle resource allocation is metaheuristic algorithms. For instance, Salem et al. (2022) [15] conducted a comprehensive survey of many metaheuristic algorithms and found that they can improve performance by reducing costs, processing time, and energy consumption while maximizing resource utilization. Guerrero et al. (2022) [16] investigated resource optimization techniques, finding that GAs are employed for their dynamic ability to adapt to changing scenarios. In a comparative analysis, Saad-Eddine et al. (2023) [17] examined GA and PSO. They concluded that while PSO is more cost-effective and faster than the energy-intensive GA, the integration of both algorithms is recommended for better overall efficiency.

Specific implementations of GAs have shown promising results. Jawad et al. (2022) [18] proposed a scheduling method called Genetic Algorithm-based Scheduling (GA-IRACE) to address cloud limitations in real-time IoT applications. This method optimizes execution time, cost, and bandwidth usage, showing a 15 to 40% performance gain over conventional approaches. Similarly, Attalah et al. (2025) [13] proposed the GA Hybrid-Fog algorithm for Internet of Drones (IoD) networks. By employing GA, this approach uses global search features to adapt to fluctuating task loads and network conditions, substantially decreasing offloading delays compared to PSO and MILP. However, it is noted that GA requires more energy and longer execution times than PSO. Additionally, Mahjoubi et al. (2024) [14] proposed Simulated Annealing Task Scheduling (SATS). Although efficient in well-managed conditions, its effectiveness heavily depends on the accuracy of service request predictions. Also, SA may not be optimally timed in dynamic environments with unpredictable traffic.

Beyond genetic algorithms, other researchers have introduced distinct techniques. Focusing on deep learning layers, Lee et al. (2020) [10] introduced the DLEFN algorithm, which dynamically assigns layers to fog nodes based on capacity and bandwidth. It dynamically scales task layers, offloading the most resource-intensive tasks only when necessary. While this favors existing applications, it may hurt the real-time performance of newer ones. Singh et al. (2022) [19] introduced a cluster-oriented technique for load balancing where resource clusters are categorized into free, working, and busy states to decide task assignment. Table 1 shows a comparison of the most recent research in resource allocation using different optimization algorithms.

Table 1. Comparison between recent technologies in resource allocation.

Category	Paper	Used Methods	Disadvantages
Metaheuristic Approaches	Salem et al. (2022) [15]	Survey of various metaheuristic algorithms (e.g., GA, PSO, etc.)	High computational and time resource requirements; no guarantee of global optimum.
	Guerrero et al. (2022) [16]	GA for resource optimization	Slower convergence rates, high energy consumption, and no guarantee of global optimum.
	Saad-Eddine et al. (2023) [17]	Comparative analysis of GA and PSO	Reliance on extensive sampling for optimal results; energy intensive nature of GA.
	Jawad et al. (2022) [18]	GA-IRACE (Genetic Algorithm-based Scheduling)	Performance effectiveness may decline in dynamic or unpredictable environments. Also, GA is energy intensive.
	Attalah et al. (2025) [13]	GA Hybrid-Fog for IoD networks	High energy consumption; longer execution times; requires careful tuning; ignores energy limits of UAVs.
	Mahjoubi et al. (2024) [14]	SATS (Simulated Annealing Task Scheduling)	Heavily dependent on service request prediction accuracy; potential for suboptimal timing in dynamic traffic. Additionally, large sample sizes are required to ensure reliable and consistent results.
Structural & Load Balancing Approaches	K. Lee et al. (2020) [10]	DLEFN (Deep Learning Entrusted to Fog Nodes)	Degrade performance of new real-time applications; strict limits can cause layer underutilization.
	Singh et al. (2022) [19]	Cluster-oriented load balancing with resource state tracking	Not explicitly mentioned; potential overhead in maintaining and tracking cluster states could be a concern.

3. Materials and Methods

In this study we proposed a hybrid system that enhances the displacement of neural network layers among fog nodes and the cloud infrastructure in IoT settings. The primary objectives of this system are to minimize data transmission to the cloud, thereby alleviating network congestion, and to expedite data analysis.

The proposed system is architected with three principal subsystems as shown in Figure 1. The first subsystem is responsible for collecting media data from IoT devices and temporarily storing it while evaluating the optimal partitioning of neural network layers across fog nodes and cloud infrastructure. Once a user submits a prompt outlining specific performance requirements such as low-latency responsiveness, memory constraints, or other operational criteria the system gathers additional contextual information. This includes a structured dictionary that maps each deep learning model to its associated applications, along with detailed memory consumption data up to each model layer.

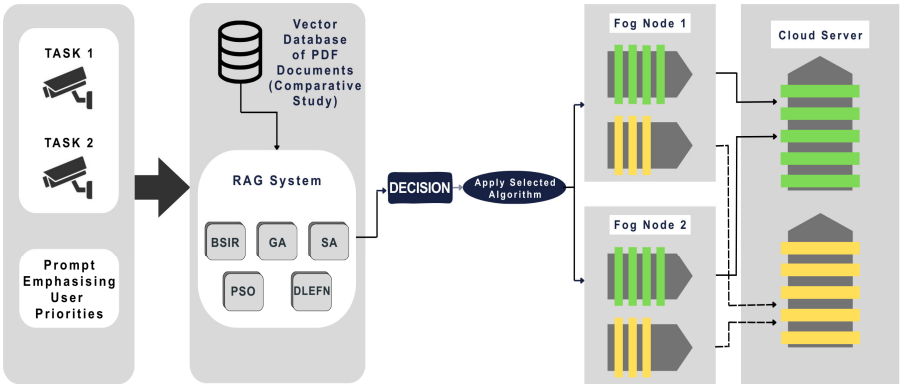


Figure 1. System overview.

The input characteristics received from the first subsystem, are subsequently analyzed by the second subsystem "Retrieval-Augmented Generation (RAG) model". This subsystem leverages metaheuristic algorithms, a novel optimization technique designated as BSIR, and a novel enhancement algorithm. Its operational mechanism involves querying a vector database to extract the most relevant knowledge to select the most effective optimization strategy for distributing model layers between fog and cloud resources. Following this selection, the chosen algorithm is employed to determine the best layer-wise distribution of each application between the fog nodes and the cloud servers.

Once the ideal distribution is determined, the third subsystem starts its execution phase: its initial layers run on the aforementioned fog nodes, whereas their intermediate outputs are additionally sent to the cloud to be computed by the rest of the following layers. Layered execution ensures effective utilization of resources while meeting user performance requirements.

3.1. The Proposed Retrieval-Augmented Generation (RAG) Subsystem

The initial subsystem orchestrates the collection of media data from IoT devices while simultaneously processing user-defined performance requirements. These requirements, which typically center on latency thresholds, memory ceilings, or energy constraints, form the basis of the optimization objective. To provide the necessary depth for decision-making, the system retrieves contextual metadata, including a structured dictionary mapping deep learning models to their respective applications and granular, layer-wise memory consumption profiles.

As shown in Figure 2, the RAG module serves as the primary decision engine, interfacing between the raw system constraints and the suite of available optimization algorithms.

This suite includes standard metaheuristics and our proposed BSIR technique. The knowledge base for this module was constructed by indexing comparative studies and empirical performance data into a locally hosted Chroma vector database. These documents are partitioned into overlapping segments and embedded to allow for semantic retrieval based on the current system state. The interaction between the RAG module and the optimization selection follows a structured logic. When a query is initiated, the system extracts the current operational parameters: the number of active model instances, total available memory, bandwidth availability, and peak layer-wise memory usage. These metrics are injected into the prompt context alongside retrieved documentation regarding the performance characteristics of each algorithm.

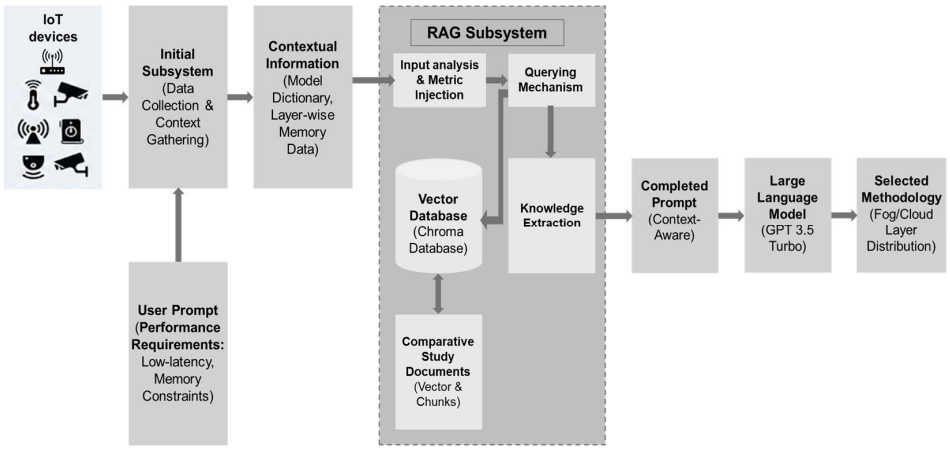


Figure 2. Proposed RAG subsystem architecture.

The selection process is determined based on a multi-objective evaluation performed by the LLM (GPT-3.5 Turbo), which dynamically shifts the decision boundary based on immediate operational needs [20]. In practice, when network volatility demands rapid convergence, the RAG module identifies heuristics with lower computational costs. In contrast, if the prompt focuses on achieving the highest efficiency of resources throughout complex fog-cloud hierarchies (e.g., memory < 0.4 GB) or requires complex layer splitting, the system favors the proposed BSIR-enhanced technique due to its demonstrated efficiency in constrained environments found in the retrieved literature. By weighing these historical benchmarks against live constraints, the LLM functions as a high-level orchestrator, determining the best technique that can handle the displacement of neural network layers across the fog nodes and cloud infrastructure.

3.2. Proposed Optimization Strategy

In this study, a new optimization strategy is proposed, termed **Binary Search-Inspired Recursive (BSIR)**, to find the best deep learning layer-wise distribution of the assigned application between the fog nodes and the cloud servers. It uses a divide-and-conquer approach that is inspired by binary search to recursively prune the search space by partitioning layer configurations, as detailed in Algorithm 1. Unlike traditional binary search, which requires sorted values, BSIR partitions the sequential layer indices and treats the model architecture as a linear sequence of potential split points. This approach allows it to accommodate the irregular or non-monotonic memory distributions inherent to deep learning models. This divide-and-conquer strategy progressively refines potential solutions by partitioning the search space and discarding less effective configurations. This iterative process gradually converges toward an optimal setup, minimizing overall execution time while maximizing efficient memory allocation. Figure 3 shows the steps of BSIR.

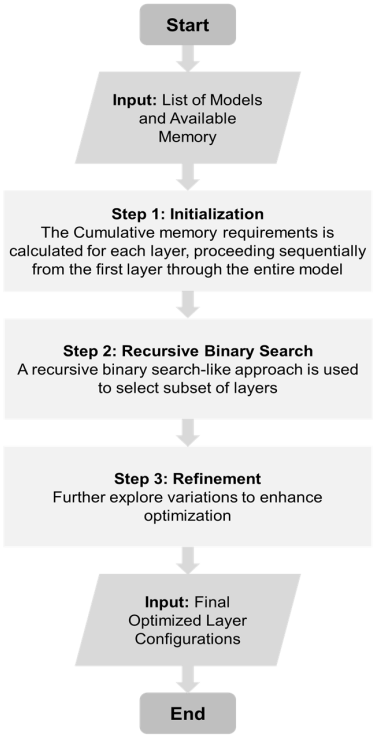


Figure 3. BSIR technique workflow.

Figure 3 illustrates the workflow of the BSIR technique. The process takes as input a list of incoming models along with the available memory on the fog node. An initialization phase then computes the cumulative memory requirements for each layer, proceeding sequentially from the first layer through the entire model. Next, a recursive binary search-like approach is employed to narrow the search space and efficiently identify optimal intermediate layer. Finally, a refinement phase evaluates additional variations to enhance performance and resource utilization, resulting in the generation of the final layer configurations.

Algorithm 1 presents the pseudocode for the proposed optimization strategy, BSIR, which enhances layer displacement between fog nodes and the cloud server. The first step of the algorithm involves the calculation and structuring of the total memory consumed by each model, encompassing all of its intermediate layers.

Memory Profiling and Initialization: It initiates with the **compute_memory_consumption** function, which profiles each model to determine its frequency $I(m_i)$, total memory footprint, and the specific memory requirements for each intermediate layer $L(m_i)$. These metrics are organized into a dictionary, dict1, and the models are sorted in descending order based on their total memory usage to prioritize high-impact models during optimization.

Recursive Binary Selection: The core optimization is driven by the **get_middle_element** function, which utilizes a binary search-like approach to narrow the search space efficiently. It accomplishes this by progressively assessing the central layer, or a nearby layer, in each model being examined:

- For each model, the system selects a specific layer index x_i based on its architectural position.
- The **check_memory_constraint_satisfaction** function then calculates the cumulative memory usage of these selections and verifies if they remain within the available memory limit A.

- If the configuration is valid, it is flagged as a potential optimum; if not, the function recursively explores a reduced subset of layers until the constraints are met.

Refinement and Finalization: Once a preliminary layer set is established, the **get_the_last_layers** function performs a final refinement. This step assesses various index combinations to improve memory efficiency and sustainability. The function concludes by merging the results from the recursive selection and the refinement phase to finalize the optimized deployment configuration.

To ensure clarity in the description of the BSIR technique, the following notations are used:

- Let M = $\{m_1, m_2, \ldots, m_n\}$ be the set of unique model types.
- Let I(m_i) be the number of instances of model m_i.
- Let L(m_i) = $\{l_{i1}, l_{i2}, \ldots, l_{ik}\}$ be the list of memory usages for each possible layer output of model m_i.
- Let $x_i \in \{0, 1, \ldots, k-1\}$ be the selected layer index for model m_i.
- Let A be the available memory (e.g., 0.4 GB).

The system fitness function can be formulated as follows:

$$f(x) = \begin{cases} \sum_{i=1}^{n} I(m_i), & \text{if } \sum_{i=1}^{n} I(m_i).\, L(m_i)[x_i] \leq A \\ -1, & \text{otherwise} \end{cases}. \quad (1)$$

Equation (1) presents the fitness function utilized in our system. Its objective is to determine the highest layer each application can reach on the fog nodes while concurrently maximizing the number of deployable applications within the constraints of available memory.

Algorithm 1: The pseudo code for the proposed optimization strategy (BSIR)

Input: List of models (with layers and memory details), available memory A, and layer configurations for each model.
Output: Optimized layer configuration that fits within the given memory constraints.

START
Step 1: Call function **compute_memory_consumption** to compute memory consumption for Each Model.
Step 2: Check if Selected Layers Fit Within Available Memory.
Step 3: Recursively Select the Middle Layer (**Following Recursive Binary Search**).
Step 4: Compute Optimized Layer Configuration.
Step 5: Call function **main** to execute the recursive layer selection and memory optimization.
Function **compute_memory_consumption**(list of Models):
for Each Model

1. Initialize an empty dictionary dict1.
2. For each unique model in models:
   a. Count occurrences of the model → I(m_i).
   b. Compute total memory consumption.
   c. Compute memory usage for each intermediate layer → L(m_i).
   d. Store results in dict1.
3. Sort dict1 by memory consumption in descending order.

Algorithm 1: *Cont.*

Function **check_memory_constraint_satisfaction**(selected_layers, current_list):

1. Initialize total_memory = 0.
2. For each model in current_list:
   a. Multiply $I(m_i) \times$ memory of selected layer $L(m_i)[x_i]$.
   b. Accumulate in total_memory.
3. If total_memory $\leq$ A, return "Acceptable".
4. Else, return "Exceeded".

Function **get_middle_element**(current_list):

1. Initialize an empty list selected_layers.
2. For each model in current_list:
   a. If only one layer exists, select index 0.
   b. If in first half, choose ceil(mid).
   c. If in second half, choose floor(mid).
   d. Store selection x_i in selected_layers.
3. Call check_memory_constraint_satisfaction (selected_layers, current_list).
   a. If "Acceptable", return selected_layers.
   b. Else, reduce current_list (keep first half layers).
4. If all selected layers are 0, stop recursion.
5. Recursively call get_middle_element(next_list).

Function **get_the_last_layers**(selected_layers):

1. If selected_layers are valid:
   a. Compute new_layers = 2 $\times$ (each_layer) (except 0 remains 1).
   b. Generate layer index combinations.
   c. Evaluate memory consumption for each combination.
   d. If a valid combination exists, return it.
   e. Else, return selected_layers.
2. Else, return None.

Function **main**():

1. Call get_middle_element(dict1).
2. Store the selected layers.
3. Call get_the_last_layers(selected_layers).
4. Store the optimized layer configuration.

END

3.3. The Proposed Enhancement Algorithm

Our proposed enhancement algorithm improves performance in multi-model environments by offering a smart and adaptive way to manage resources and explore models. Additionally, by analyzing memory use at the individual layer level and identifying usage patterns across different models, the algorithm ensures memory is distributed fairly, preventing any one model from taking over system resources. This, in turn, enhances system stability and greatly improves scalability, manageability, and overall reliability, making the environment well-suited for deploying complex model architectures.

Algorithm 2 provides a detailed description of the specific steps of the enhancement algorithm. The algorithm begins by indexing the current layers of each model to assess their individual contributions to memory consumption. It then calculates the total memory

usage across all models and dynamically redistributes resources across the model suite. An initial footprint is established by multiplying the number of model instances by the memory requirements of their current layers; this value is subtracted from the total capacity to determine the remaining overhead available for updates.

Based on this estimate, the system evaluates whether each model can transition to a deeper architectural layer. Models remain at their current depth if the memory threshold is exceeded; otherwise, they progress. Adjustment ratios are applied to scale this progression proportionally. If the predefined depth parameter exceeds one, the process iterates with updated layer configurations to balance optimization gains against runtime overhead. Finally, this logic is integrated into the BSIR method and the metaheuristic algorithms (GA, SA, and PSO) to refine decision-making and computational efficiency.

Algorithm 2: The pseudo code for Enhancer algorithm

```
Function Enhancer(dict1, available_memory, depth):
    If depth == 0:
        Return dict1  # Base case: Stop recursion
    Compute layer indices for each model
    Normalize memory contributions
    Compute initial memory usage and remaining available memory
    Adjust layers dynamically based on memory constraints
    Compute ratios to refine allocation
    If depth > 1:
        # Recursive call
        Return Enhancer(updated_dict, available_memory, depth - 1)
    Return updated_dict  # Final optimized configuration
```

4. Results

The proposed system is evaluated compared to several recent models DLEFN, GA, SA and PSO [10,13,14,19]. The comparative assessment employs a comprehensive suite of evaluation metrics. These metrics encompass critical performance indicators such as memory and CPU utilization, the maximum concurrent application execution capacity on fog nodes, overall execution duration, and the total memory footprint of deployed applications within the fog infrastructure. To ensure a fair and standardized comparison, a consistent memory constraint of 0.4 GB was uniformly applied across all evaluated techniques. The experimental framework for this evaluation was constructed utilizing a combination of the LeNet-5 and AlexNet deep learning architectures.

The LeNet's architecture consists of two convolutional layers that use 5×5 kernels, paired with two max-pooling layers for downsampling. This is followed by dense (fully connected) layers and a final output layer. The concealed layers utilize Rectified Linear Unit (ReLU) activation functions, whereas the output layer uses a sigmoid activation function. A flattening step precedes the fully connected layers to prepare the feature maps. A complete specification of this model's architecture is provided in Table 2.

The AlexNet architecture comprises five convolutional layers with increasing filter sizes. Max-pooling operations are distinctively performed using 3×3 windows with a stride of 2. The network incorporates substantial fully connected layers, each with 4096 neurons, preceding the final output layer. The ReLU activation functions are consistently utilized throughout the network's layers to introduce non-linearity, while a sigmoid activation function is utilized in the output layer. A comprehensive description of this model's architecture is available in Table 2.

Table 2. Deep learning architectures.

Feature	LeNet Architecture	AlexNet Architecture
Convolutional Layers	Two Conv2D layers with 5 × 5 kernels	Five Conv2D layers with progressively larger filter sizes (11 × 11, 5 × 5, 3 × 3)
Pooling Layers	Two MaxPool2D layers for downsampling	MaxPooling2D layers with 3 × 3 pooling windows and a stride of 2
Fully Connected Layers	Dense(256) → Dense(84) → Dense(2)	Two Dense (4096) layers followed by the final Dense (2) output layer
Special Convolution Feature	No special or non-standard convolutional design choices—	The initial Conv2D layer employs an 11 × 11 kernel with a stride of 4 (strided convolution)
Activation (Hidden)	‘relu’ activation in all hidden layers	‘relu’ activation used consistently throughout
Activation (Output)	Dense (2, activation = ‘sigmoid’)	Dense (2, activation = ‘sigmoid’)
Flattening	Flatten() layer preceding the fully connected layers	Flatten() layer before the fully connected layers

The experimental setup comprised an AMD Ryzen 7 5800H processor (16 cores, approximately 3.2 GHz base clock) and 16 GB of RAM to ensure ample memory. The system used integrated Radeon Graphics alongside a separate NVIDIA GeForce RTX 3060 GPU with 6 GB of VRAM for graphics processing. This configuration resulted in an approximate total of 14.1 GB of GPU memory, drawing from both dedicated and shared resources.

In order to simulate the hardware limitations of industrial fog nodes, we restricted the memory setup to 0.4 GB. This reflects actual IoT deployments where edge gateways have limited RAM available for inference after the operating system and background services are running. By enforcing this 0.4 GB limit, we were able to check that the algorithm is still stable and efficient even when it is working at the edge of its safety margin.

4.1. First Experiment: Best Hyperparameters of Metaheuristic Algorithms

To improve resource utilization and overall system performance, a proposed enhancement algorithm is integrated into the optimization process. Each metaheuristic algorithm, Genetic Algorithm (GA) [13,19,21], Simulated Annealing (SA) [14,22,23], and Particle Swarm Optimization (PSO) [17,19] is executed iteratively. This iterative execution aims to determine the optimal layer configurations for each deep learning model deployed on fog computing systems, operating within the available memory constraints, while concurrently seeking to maximize the number of applications that can be executed simultaneously. The specific parameter values utilized for the GA, SA, and PSO models are detailed in Tables 3–5, respectively.

Table 3. GA algorithm parameters.

Parameter	Value
Population Size	50
Number of Generations	100 generations
Mutation Rate	0.1
Random Seeds	250

Table 4. SA algorithm parameters.

Parameter	Value
Initial Temperature	1000
Cooling Rate	0.99
Minimum Temperature	1
Random Seed	150

Table 5. PSO algorithm parameters.

Parameter	Value
Swarm Size	50 particles
Iterations	100
Inertia Weight (w)	0.5
Cognitive Constant (c1)	1.5
Social Constant (c2)	1.5
Random Seed	250

4.2. Second Experiment: Memory Consumption

Table 6 presents an evaluation of RAM consumption, measured in megabytes (MB), for the proposed system in comparison to several recent methodologies: GA, SA, PSO, and DLEFN [10]. This comparative analysis was conducted across varying numbers of concurrently executing applications.

Table 6. Memory consumption (MB) for the proposed system and recent approaches.

Number of Apps		Optimization Techniques						
		BSIR—Enhancer Depth (1)	**BSIR—Enhancer Depth (2)**	**GA [13,19]—Enhancer**	**SA [14]—Enhancer**	**PSO [14]—Enhancer**	**GA, SA, and PSO [13,14,19]**	**DLEFN [10]**
RAM Consumed (MB)	10	0.04	0.16	0.19	0.15	0.18	0.16	0.16
	20	0.03	0.16	0.19	0.16	0.18	0.16	0.17
	50	0.07	0.16	0.20	0.15	0.19	0.16	0.49
	100	0.10	0.16	0.19	0.16	0.18	0.16	1.57
	150	0.10	0.16	0.19	0.16	0.19	0.17	2.43
	200	0.10	0.16	0.19	0.16	0.19	0.16	3.32
	250	0.10	0.16	0.19	0.16	0.19	0.17	3.92
	300	0.10	0.16	0.20	0.16	0.19	0.17	4.78
	400	0.10	0.16	0.20	0.16	0.20	0.16	6.18
	500	0.16	0.16	0.18	0.17	0.20	0.16	7.68
	600	0.16	0.16	0.19	0.15	0.18	0.16	9.41
	800	0.16	0.16	0.21	0.17	0.18	0.16	12.30
	1000	0.16	0.16	0.18	0.16	0.19	0.16	15.40
	Max Apps	1193	1193	1193	1193	1193	1193	1185

Generally, memory consumption remains stable across the evaluated methods as the number of applications increases, with a few exceptions. However, DLEFN requires significantly more memory, particularly as the application count rises. For instance, when DLEFN is tasked with processing 1000 applications, it uses up to 15.40 MB of memory. That is a big jump compared to the maximum of just 0.21 MB seen with other methods.

This clearly demonstrates DLEFN's substantially greater memory requirements. Furthermore, all methods except DLEFN can handle a larger number of simultaneous applications, accommodating up to 1193 (597 from the initial model and 596 from the subsequent one) before encountering memory limitations. In contrast, DLEFN's capacity is somewhat reduced, processing a total of 1185 applications (specifically, 597 from the initial model and 588 from the subsequent one). This suggests an inherent trade-off between DLEFN's enhanced features and its increased memory consumption.

The table also illustrates the improvement in performance we achieve by merging our proposed enhancement algorithm with other optimization techniques. While RAM consumption stays roughly constant, as we will detail later in this study, incorporating

our enhancement algorithm results in noticeable performance improvements, particularly regarding resource utilization.

Figure 4 compares the minimum, maximum, and average memory consumption across all evaluated partitioning techniques. DLEFN exhibits the highest memory overhead, scaling from 0.16 MB to 15.40 MB (5.22 MB average) as the number of applications increases. This inefficiency results from its computationally intensive node-level resource availability checks and the recursive reallocation of existing tasks. Specifically, when a new DL application is initialized, DLEFN evaluates the capacity of each fog node; if no node satisfies the minimum layer requirements, the algorithm reallocates layers from existing tasks to ensure feasibility. While metaheuristics like GA, SA, and PSO maintain a low footprint (~0.16 MB), they fail to guarantee convergence to optimal solutions.

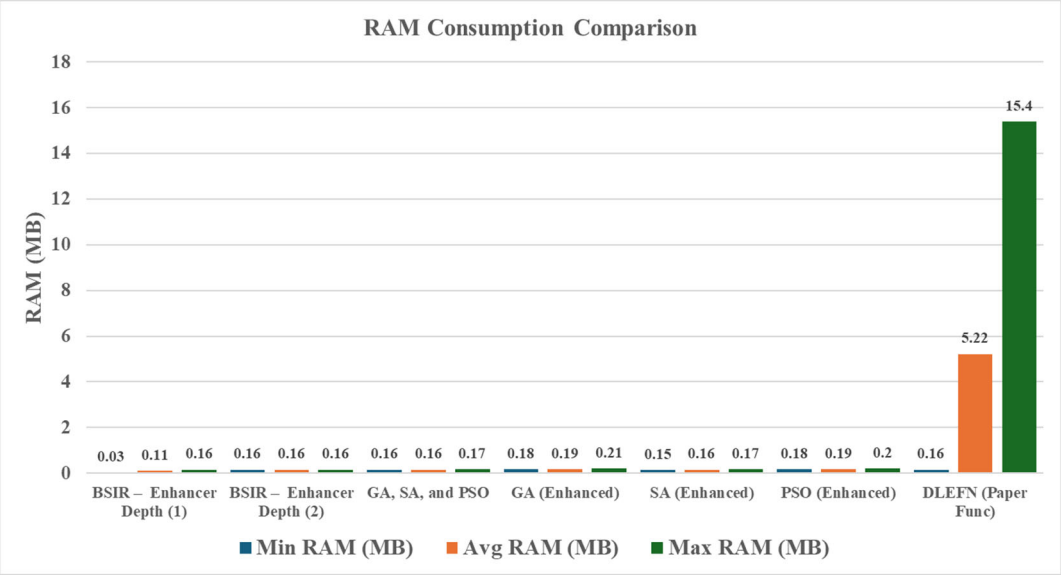


Figure 4. Comparative RAM consumption (MB) of the proposed BSIR approaches, standard metaheuristics (GA, SA, PSO), and the DLEFN baseline, showing minimum, average, and maximum memory usage across all simulated deep learning applications using LeNet and AlexNet architectures.

Our BSIR-enhanced framework bridges this gap, achieving superior partitioning with negligible overhead; even when integrated with metaheuristics, consumption peaks at only 0.21 MB. Specifically, BSIR at depth 1 offers the smallest footprint, ranging from 0.03 MB to 0.16 MB, making it ideal for memory-constrained environments. At depth 2, consumption stabilizes at 0.16 MB regardless of application count. This marginal increase between the two depths results from the additional refinement performed during the second recursion to optimize resource allocation within the safety margin. Crucially, such stability demonstrates that BSIR provides the necessary scalability for large-scale deployments without the exponential memory growth characteristic of DLEFN.

4.3. Third Experiment: CPU Utilization

Table 7 summarizes CPU utilization across varying concurrent application scales. The BSIR-Enhancer, at both Depth 1 and Depth 2, generally maintains CPU usage peaking at 2.77%, which suggests computational efficiency when managing larger workloads while providing better partitioning decisions. In contrast, DLEFN exhibits a non-linear increase in CPU demand, rising from 4.40% for 10 applications to 30.50% for 800 applications. Reaching 51% at 1000 applications is likely due to the complexity of its optimization mechanisms while making partitioning decisions.

Table 7. CPU consumption (%) by each approach.

Number of Apps		Optimization Techniques						
		BSIR—Enhancer Depth (1)	**BSIR—Enhancer Depth (2)**	**GA [13,19]—Enhancer**	**SA [14]—Enhancer**	**PSO [14]—Enhancer**	**GA, SA, and PSO [13,14,19]**	**DLEFN [10]**
CPU Consumption by the Functions (Assuming Equal from each model.) For ex, for 10 apps: 5 of model 1 and 5 of model 2	10	0.53%	0.57%	2.57%	1.50%	3.70%	3.10%	4.40%
	20	2.77%	0.53%	2.50%	2.50%	3.00%	0.90%	11.00%
	50	1.77%	0.63%	3.23%	1.77%	3.93%	5.30%	8.00%
	100	1.00%	0.53%	3.00%	2.50%	4.00%	4.40%	11.30%
	150	1.23%	0.63%	2.33%	1.87%	4.30%	6.20%	12.00%
	200	2.07%	0.95%	2.60%	3.00%	3.50%	3.80%	17.43%
	250	1.30%	0.93%	2.80%	4.07%	3.83%	5.40%	19.50%
	300	1.60%	1.13%	3.13%	4.70%	4.33%	7.00%	19.00%
	400	0.90%	1.13%	3.37%	5.35%	4.67%	5.10%	22.55%
	500	1.35%	1.13%	3.70%	3.06%	4.83%	4.70%	22.15%
	600	1.08%	1.60%	2.54%	2.40%	5.80%	2.30%	25.90%
	800	0.90%	1.63%	4.03%	5.10%	5.03%	2.30%	30.50%
	1000	1.70%	1.80%	4.90%	2.67%	3.17%	1.90%	51.00%

Baseline implementations of GA, SA, and PSO demonstrate CPU utilization ranging from 0.90% to 7%. Although these requirements are lower than those of DLEFN, they consistently exceed the consumption of our BSIR technique in some cases. While these standard metaheuristics do not guarantee optimal partitioning decisions, their integration with the proposed enhancement algorithm results in measurable improvements in metrics related to resource usage within the safety margin, such as memory utilization. These results suggest that the enhancement algorithm facilitates more effective resource management across different optimization frameworks without consuming more resources.

Figure 5 compares the minimum, maximum, and average CPU usage for each method across different number of applications. As shown in the figure, our models demonstrate much lower CPU consumption. For instance, DLEFN's highest CPU usage hits 51.00%, whereas the peak for the other methods is merely 7.00%. Notably, the enhanced versions incorporating our enhancement algorithm consume nearly the same amount of CPU as their non-enhanced counterparts. However, as will be demonstrated later, they achieve superior performance in aspects other than CPU usage compared to GA [13], SA [14], and PSO [19] without incorporating the enhancement algorithm.

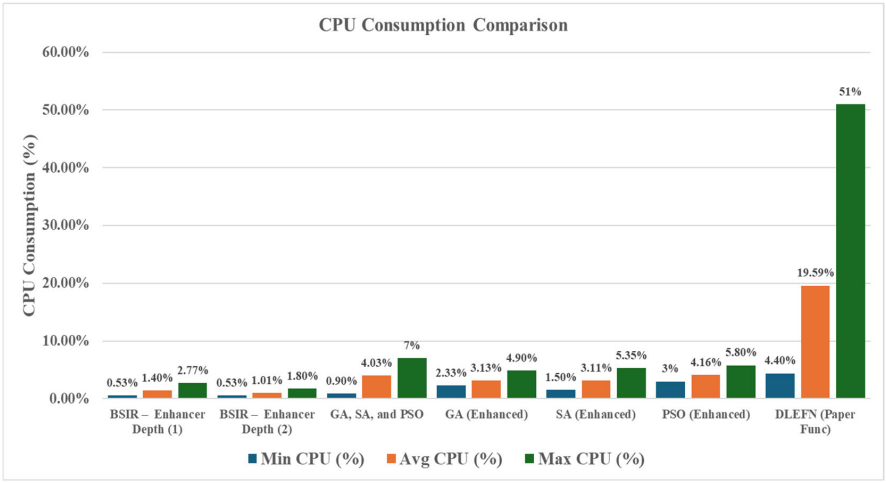


Figure 5. Comparative CPU consumption (%) of the proposed BSIR approaches, standard metaheuristics (GA, SA, PSO), and the DLEFN baseline, showing minimum, average, and maximum CPU usage across all simulated deep learning applications using LeNet and AlexNet architectures.

These results clearly indicate the effectiveness of the enhancement algorithm in optimizing performance and ensuring efficient utilization of available resources. This implies that including this method in the existing algorithms can offer considerable promise in improving performance overall without incurring a significant computation cost.

4.4. Fourth Experiment: Overall Execution Time

Table 8 compares the decision-making latency for the evaluated algorithms across varying application counts: BSIR with Depth = 1, BSIR with Depth = 2, GA, SA, PSO, and DLEFN. The evaluation was performed across a range of application counts. The proposed **BSIR** system demonstrates high efficiency and operational stability; at Depth 1, response times average 0.30–0.32 s, while Depth 2 achieves even greater efficiency at approximately 0.15 s. This consistent performance highlights the system's reliability for large-scale tasks while maximizing resource utilization within the safety margin.

Table 8. Execution time by each approach.

Number of Apps		Optimization Techniques						
		BSIR—Enhancer Depth (1)	**BSIR—Enhancer Depth (2)**	**GA [13,19]—Enhancer**	**SA [14]—Enhancer**	**PSO [14]—Enhancer**	**GA, SA, and PSO [13,14,19]**	**DLEFN [10]**
Time (Sec.) by the Functions to make the decision	10	0.30	0.15	0.14	0.10	0.11	0.13	0.11
	20	0.30	0.15	0.14	0.11	0.11	0.13	0.23
	50	0.31	0.15	0.14	0.10	0.11	0.13	0.78
	100	0.31	0.15	0.14	0.11	0.11	0.12	1.88
	150	0.31	0.14	0.14	0.11	0.11	0.12	3.30
	200	0.31	0.15	0.14	0.11	0.11	0.13	4.69
	250	0.31	0.15	0.14	0.10	0.11	0.13	6.67
	300	0.31	0.15	0.14	0.11	0.11	0.13	9.07
	400	0.32	0.15	0.14	0.11	0.11	0.13	22.89
	500	0.31	0.15	0.14	0.11	0.11	0.13	30.96
	600	0.30	0.15	0.14	0.11	0.11	0.13	42.87
	800	0.30	0.15	0.14	0.11	0.12	0.13	70.80
	1000	0.31	0.15	0.14	0.11	0.12	0.13	96.10

While the metaheuristics (GA, SA, and PSO) achieve comparable low latencies (~0.11–0.14 s), they do not guarantee convergence to optimal partitioning decisions. In contrast, DLEFN exhibits poor scalability, with its decision time increasing sharply from 0.11 s at low application counts to 96.10 s as the workload approaches 1000 applications. Consequently, the enhanced BSIR system proves superior for real-time environments, delivering the speed of heuristics without the computational bottlenecks observed in deep learning approaches, while improving resource utilization, as demonstrated in the next experiment.

4.5. Fifth Experiment: Resource Utilization

Figure 6 illustrates RAM consumption for the proposed BSIR configurations, composite metaheuristics (GA-SA-PSO), and the DLEFN baseline. The reported metrics aggregate performance across various partitioning scenarios and application counts. To ensure industrial relevance, the environment was restricted to a 409.6 MB (0.4 GB) memory ceiling, mirroring the constrained capacity of IoT gateways after accounting for essential system overhead. The comparative analysis of RAM consumption, illustrated in the figure, reveals distinct efficiency profiles:

- **Resource Maximization:** The DLEFN function exhibited a highly aggressive approach to utilizing the available safety margin. It achieved a high average consumption of 390.14 MB and reached the hardware limit of 409.6 MB. However, a significant limitation arises from the memory overhead of the partitioning process itself, which

requires 15.40 MB when managing 1000 applications. If this internal consumption is factored into the safety margin, the system likely fails to reach the 409.6 MB peak because it triggers the margin threshold prematurely. This differs from algorithms refined by the enhancement framework, which appear to maximize resource utilization while maintaining a much smaller computational footprint.

- Similarly, the GA-SA-PSO Enhanced and BSIR—Enhancer (Depth 2) models exhibited robust resource engagement, with average usage levels of 368.64 MB and 357.38 MB, respectively. Correspondingly, the BSIR–Enhancer (Depth 2) and GA-SA-PSO Enhanced models demonstrated robust engagement with the hardware, peaking at 405.5 MB and 404.48 MB, respectively. Compared to the DLFN function, these approaches appear more efficient, as they require considerably less memory for partitioning decisions.
- **Underutilization Risks:** The baseline GA-SA-PSO displayed significant underutilization, as it averaged only 229.38 MB with a peak of 349.18 MB and minimum usage of 86.02 MB.

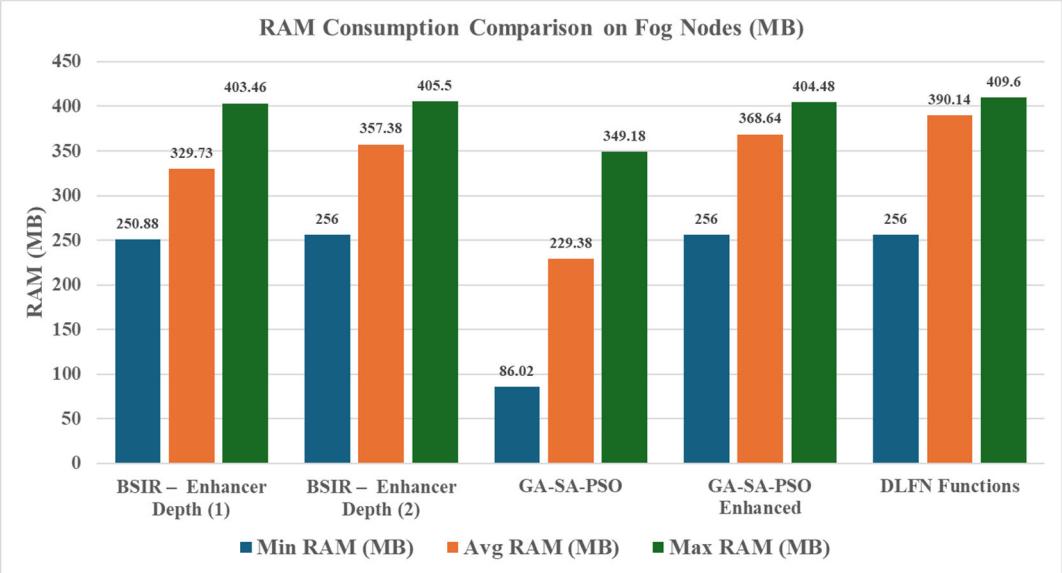


Figure 6. Comparative RAM consumption (MB) on fog nodes for the proposed BSIR approaches, composite metaheuristics (GA-SA-PSO), and the DLEFN baseline, reported as minimum, average, and maximum values over different numbers of simulated applications.

Based on these results, BSIR-enhanced methods, followed by GA, SA, and PSO refined by the enhancement algorithm, demonstrate the best solutions by maximizing resource utilization while minimizing the memory overhead of partitioning decisions.

4.6. Sixth Experiment: Demonstrating RAG-Driven Adaptive Decisions Impact

To highlight the importance of the RAG adaptive decision-making process, we conducted an ablation study comparing two distinct scenarios. The first involves a system that employs the same partitioning technique (e.g., DLEFN) for all scenarios. The second demonstrates the advantages of adding the RAG subsystem, which autonomously identifies the most effective partitioning method. This selection is based on real-time factors, including the number of applications, available resources, DL model architecture, and specific user performance requirements.

The importance of the RAG subsystem as a dynamic orchestrator is best demonstrated by comparing the performance outcomes of its adaptive selection logic against "fixed strategy" scenarios. The RAG module employs a knowledge base of empirical performance data to avoid suboptimal resource allocation.

Scenario 1: Latency-Sensitive Environments: In scenarios where the user-defined performance requirements prioritize faster response times, employing a dynamic resource allocation is essential to meet user and system requirements. In contrast, when using a static system that employs a fixed partitioning technique for all scenarios, it might not fit with user requirements and result in significant operational bottlenecks and increased latency, unlike user requirements.

- **Fixed-strategy Failure (DLEFN)**: If the system used a fixed strategy such as DLEFN, scaling to 1000 concurrently running applications leads to a huge increase in decision latency, with partitioning decisions requiring approximately 96.10 s.
- **RAG-Driven Selection:** The RAG module identifies this constraint and dynamically switches to the **SA-Enhanced** algorithm, which reduces partitioning decision-making time to approximately 0.11 s.
- **Impact:** In this context, RAG-driven selection provides a huge improvement in execution time compared to a fixed DLEFN deployment.

Scenario 2: Extreme Resource-Constrained Fog Nodes: When operating on industrial fog nodes with limited resources and restricted to a 0.4 GB memory safety margin, the choice of optimization algorithm becomes critical for system stability.

- **Fixed DLEFN Strategy:** Under heavy loads (1000 apps), DLEFN consumes 51.00% of CPU resources and requires 15.40 MB of RAM for the decision process alone.
- **RAG-Driven Selection:** The RAG module injects the 0.40 GB constraint into the prompt context and selects the **BSIR-Enhanced** algorithm. This selection maintains CPU utilization below 2% and RAM consumption at a stable 0.16 MB.
- **Impact:** The RAG module ensures that the system remains within its "Safety Margin," avoiding the non-linear CPU scaling and potential crashes associated with high-overhead algorithms in resource-limited environments.

By using the Chroma vector database to weigh historical benchmarks comparative studies of all partitioning techniques, the RAG subsystem transforms the framework from a static tool into an adaptive, context-aware orchestrator. Table 9 summarizes how the RAG subsystem shifts decision boundaries to optimize different performance metrics according to available resources and user requirements.

Table 9. Comparative Analysis of Fixed vs. RAG-Driven Resource Allocation Strategies.

Scenario	Metric	Fixed Strategy (DLEFN)	RAG-Driven Selection	Performance Impact
Latency-Sensitive (1000 Apps)	Decision Latency	96.10 s	0.11 s (via SA-Enhanced)	~874× reduction in latency
Resource-Constrained Fog Nodes	CPU Utilization	51.00%	<2.0% (via BSIR-Enhanced)	96.00% reduction in CPU overhead
	RAM Consumption	15.40 MB	0.16 MB (via BSIR-Enhanced)	99.00% reduction in memory footprint

5. Discussion

The BSIR technique demonstrates effective resource optimization, especially in environments where memory and processing power are limited. The recursive structure of the BSIR technique facilitates efficient partitioning decisions with minimal computational overhead, a capability further extended by the enhancement algorithm. This enhancement algorithm improves exploration of the search space without increasing hardware demands. Even with heavy application loads, the BSIR technique, especially when used with the enhancement algorithm, exhibits low memory consumption and low execution time. When

compared to other conventional metaheuristics such as GA, SA, and PSO, the BSIR technique offers a more resource-aware alternative that is suitable for real-time deployment.

In these deployment scenarios, a critical trade-off exists between efficiency and robustness. The framework aims to maximize memory usage, often achieving throughput levels close to 99% during the initial setup. To help manage this, it has a configurable "Safety Margin." This ensures that the system keeps a functional buffer to deal with workload changes and avoid crashes, addressing the potential risks that come with operating near saturation in real-time environments.

Applying the proposed enhancement algorithm to other optimization techniques, GA, SA, and PSO, significantly improved their performance, particularly in terms of memory usage and operational consistency, without increasing computational cost. While requiring similar CPU and memory resources as their original versions, these enhanced algorithms made better decisions. Integrating the enhancement algorithm with other algorithms is straightforward yet strategically powerful. Also, the performance improvement across all enhanced versions indicates that reducing the memory consumption of the partitioning techniques, along with intelligent exploration strategies, can enhance system scalability and responsiveness.

As shown in the results section, the main goal of the RAG framework is to allow adaptive selection of these optimization strategies. By using the vector store that contains historical performance data, the RAG subsystem can dynamically identify the best-suited approach that fits current deployment conditions. By incorporating operational parameters, such as memory limits and bandwidth availability, into the LLM prompt, the system determines the most appropriate partitioning technique. So, instead of relying on a single partitioning technique regardless of the existing resources and user's requirements, a dynamic and data-driven approach is adopted. As a result, the system selects the most appropriate optimization technique that is contextually met with current system requirements.

While this study primarily uses GPT-3.5, the dependency on cloud-based APIs may present challenges in offline fog environments. To address this, an alternative localized LLM can be employed. As quantized, edge-optimized LLMs can be used to facilitate autonomous decision-making directly on fog nodes, ensuring operational continuity in network-isolated scenarios.

6. Conclusions and Future Work

In summary, this study introduces a novel hybrid method that substantially enhances the practical deployment of deep learning across the fog–cloud continuum within IoT environments. The proposed system, based on the BSIR optimization method, has demonstrated exceptional efficacy, improving memory utilization as high as 99%, an exponential rise from the previous high of 85.25% seen in traditional methods. Furthermore, a new enhancer algorithm is proposed; it did not only improve BSIR's performance but also markedly enhanced the resource utilization of traditional metaheuristic algorithms such as GA, SA, and PSO, elevating it from a maximum of 85.25% to an impressive 98.75% in fog environments, all without increasing computational overhead. This improvement also helped create consistency, as GA, SA, and PSO all reached similar levels of CPU usage, time, and memory consumption. Furthermore, the inclusion of the RAG system significantly enhances the framework's intelligence by enabling dynamic, context-driven algorithm selection. This comprehensive approach synthesizes BSIR's memory efficiency, the adaptability of RAG, and the substantial resource improvements provided by the proposed enhancement algorithm (which averages a 13.5% gain for GA, SA, and PSO in fog environments). Collectively, these integrations represent a major advancement in overcoming resource constraints and optimizing deep learning performance within the

expanding IoT. While this study used LeNet and AlexNet to validate the framework across distinct parameter scales, the proposed system is model agnostic. Future research will focus on extending these evaluations to contemporary lightweight CNNs (e.g., MobileNet) and edge-adapted transformers (e.g., MobileViT) to further demonstrate the framework's scalability and practical relevance in next-generation IoT–Fog ecosystems.

Author Contributions: Conceptualization, G.A. and M.E.; Data curation, M.A.; Formal analysis, M.A., G.A. and M.E.; Investigation, M.A., G.A. and M.E.; Methodology, M.A., G.A. and M.E.; Software, M.A.; Supervision, G.A. and M.E.; Validation, M.A., G.A. and M.E.; Visualization, M.A.; Writing—original draft, M.A.; Writing—review and editing, M.A., G.A. and M.E. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data available on request from the authors.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

IoT	Internet of Things
DL	Deep Learning
BSIR	Binary Search-Inspired Recursive
RAG	Retrieval-Augmented Generation
GA	Genetic Algorithm
SA	Simulated Annealing
PSO	Particle Swarm Optimization
DLEFN	Deep Learning Execution Framework for Fog Nodes
IoD	Internet of Drones
SATS	Simulated Annealing Task Scheduling
MILP	Mixed Integer Linear Programming
UAV	Unmanned Aerial Vehicle
LLM	Large Language Model
GPT	Generative Pre-trained Transformer
ReLU	Rectified Linear Unit
CPU	Central Processing Unit
MB	Megabyte

References

1. Panagou, I.C.; Katsoulis, S.; Nannos, E.; Zantalis, F.; Koulouras, G. A Comprehensive Evaluation of IoT Cloud Platforms: A Feature-Driven Review with a Decision-Making Tool. *Sensors* **2025**, *25*, 5124. [CrossRef] [PubMed]
2. Hong, S.; Park, S.; Youn, H.; Lee, J.; Kwon, S. Implementation of Smart Farm Systems Based on Fog Computing in Artificial Intelligence of Things Environments. *Sensors* **2024**, *24*, 6689. [CrossRef] [PubMed]
3. Singh, J.; Singh, P.; Gill, S.S. Fog computing: A taxonomy, systematic review, current trends and research challenges. *J. Parallel Distrib. Comput.* **2021**, *157*, 56–85. [CrossRef]
4. Baccarelli, E.; Naranjo, P.G.V.; Scarpiniti, M.; Shojafar, M.; Abawajy, J.H. Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study. *IEEE Access* **2017**, *5*, 9882–9910. [CrossRef]
5. Ahmad, S.; Shakeel, I.; Mehfuz, S.; Ahmad, J. Deep learning models for cloud, edge, fog, and IoT computing paradigms: Survey, recent advances, and future directions. *Comput. Sci. Rev.* **2023**, *49*, 100568. [CrossRef]
6. Dastjerdi, A.V.; Buyya, R. Fog computing: Helping the Internet of Things realize its potential. *Computer* **2016**, *49*, 112–116. [CrossRef]

7. Zhang, K.; Ni, J.; Yang, K.; Liang, X.; Ren, J.; Shen, X.S. Security and privacy in smart city applications: Challenges and solutions. *IEEE Commun. Mag.* **2017**, *55*, 122–129. [CrossRef]
8. Yi, S.; Li, C.; Li, Q. A survey of fog computing: Concepts, applications, and issues. In Proceedings of the 2015 Workshop on Mobile Big Data, Hangzhou, China, 21 June 2015; pp. 37–42. [CrossRef]
9. Nagabushnam, G.; Choi, Y.; Kim, K.H. FODAS: A Novel Reinforcement Learning Approach for Efficient Task Scheduling in Fog Computing Network. In Proceedings of the 9th International Conference on Fog and Mobile Edge Computing (FMEC 2024), Malmo, Sweden, 2–5 September 2024; pp. 46–53. [CrossRef]
10. Lee, K.; Silva, B.N.; Han, K. Deep Learning Entrusted to Fog Nodes (DLEFN) Based Smart Agriculture. *Appl. Sci.* **2020**, *10*, 1544. [CrossRef]
11. Chiang, M.; Zhang, T. Fog and IoT: An overview of research opportunities. *IEEE Internet Things J.* **2016**, *3*, 854–864. [CrossRef]
12. Abirami, R.; Eswaran, P. HAWKFOG—An enhanced deep learning framework for the Fog–IoT environment. *Front. Artif. Intell.* **2024**, *7*, 1354742. [CrossRef] [PubMed]
13. Attalah, M.A.; Zaidi, S.; Mellal, N.; Calafate, C.T. Task-offloading optimization using a genetic algorithm in hybrid fog computing for the Internet of Drones. *Sensors* **2025**, *25*, 1383. [CrossRef] [PubMed]
14. Mahjoubi, A.; Ramaswamy, A.; Grinnemo, K.-J. An online simulated annealing-based task offloading strategy for a mobile edge architecture. *IEEE Access* **2024**, *12*, 70707–70718. [CrossRef]
15. Salem, A.; Al-Gaphari, G. Meta-heuristic Algorithms for Resource Allocation in Fog Computing. *Int. J. Mod. Trends Sci. Technol.* **2022**, *8*, 134–143.
16. Guerrero, C.; Lera, I.; Juiz, C. Genetic-based optimization in fog computing: Current trends and research opportunities. *Swarm Evol. Comput.* **2022**, *72*, 101094. [CrossRef]
17. Chafi, S.-E.; Balboul, Y.; Fattah, M.; Mazer, S.; El Bekkali, M. Enhancing resource allocation in edge and fog-cloud computing with genetic algorithm and particle swarm optimization. *Intell. Converg. Netw.* **2023**, *4*, 273–279. [CrossRef]
18. Arshed, J.U.; Ahmed, M.; Muhammad, T.; Afzal, M.; Arif, M.; Bazezew, B. GA-IRACE: Genetic algorithm-based improved resource aware cost-efficient scheduler for cloud fog computing environment. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 6355192. [CrossRef]
19. Singh, P.; Kaur, R.; Rashid, J.; Juneja, S.; Dhiman, G.; Kim, J.; Ouaissa, M. A fog-cluster based load-balancing technique. *Sustainability* **2022**, *14*, 7961. [CrossRef]
20. He, Y.; Fang, J.; Yu, F.R.; Leung, V.C. Large Language Models (LLMs) Inference Offloading and Resource Allocation in Cloud-Edge Computing: An Active Inference Approach. *IEEE Trans. Mob. Comput.* **2024**, *23*, 11253–11264. [CrossRef]
21. Mitchell, M. *An Introduction to Genetic Algorithms*; MIT Press: Cambridge, MA, USA, 1998.
22. Vergara, J.; Botero, J.; Fletscher, L. A comprehensive survey on resource allocation strategies in fog/cloud environments. *Sensors* **2023**, *23*, 4413. [CrossRef] [PubMed]
23. Mangalampalli, S.S.; Reddy, P.V.; Reddy Karri, G.; Tippani, G.; Kota, H. Priority-Aware Multi-Objective Task Scheduling in Fog Computing Using Simulated Annealing. *Sensors* **2025**, *25*, 5744. [CrossRef] [PubMed]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.