Future Generation Computer Systems 166 (2025) 107696

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Enhancing performance of machine learning tasks on edge-cloud infrastructures: A cross-domain Internet of Things based framework

Osama Almurshed [a,f,*], Ashish Kaushal [b], Souham Meshoul [c], Asmail Muftah [d], Osama Almoghamis [e], Ioan Petri [a], Nitin Auluck [b], Omer Rana [a]

[a] Cardiff University, United Kingdom
[b] Indian Institute of Technology Ropar, India
[c] Princess Nourah bint Abdulrahman University, Saudi Arabia
[d] Azzaytuna University, Libya
[e] King Saud University, Saudi Arabia
[f] Prince Sattam Bin Abdulaziz University, Saudi Arabia

ARTICLE INFO

Keywords:
Artificial intelligence
Distributed systems
Edge computing
Internet of things
Machine learning

ABSTRACT

The Internet of Things (IoT) and Edge-Cloud Computing have been trending technologies over the past few years. In this work, we introduce the Enhanced Optimized-Greedy Nominator Heuristic (EO-GNH), a framework designed to optimize machine learning (ML) and artificial intelligence (AI) application placement in edge environments, aiming to improve Quality of Service (QoS). Developed specifically for sectors such as smart agriculture, industry, and healthcare, EO-GNH integrates asynchronous MapReduce and parallel meta-heuristics to effectively manage AI applications, focusing on execution performance, resource utilization, and infrastructure resilience. The framework carefully addresses the distribution challenges of AI applications, especially Service Function Chains (SFCs), in edge-cloud infrastructures. It contains Data Flow Management, which covers aspects of data storage and data privacy, and also considers factors like regional adaptations, mobile access, and AI model refinement. EO-GNH ensures high availability for forecasting, prediction, and training AI models, operating efficiently within a geo-distributed infrastructure. The proposed strategies within EO-GNH emphasize concurrent multi-node execution, enhancing AI application placement by improving execution time, dependability, and cost-effectiveness. The efficiency of EO-GNH is demonstrated through its impact on QoS in real-time resource management across three application domains, highlighting its adaptability and potential in diverse cross-domain IoT-based environments.

1. Introduction

The Internet of Things (IoT) is rapidly becoming a transformative technology in various domains, reshaping how we approach everything from agricultural practices to healthcare delivery and industrial operations. This technological paradigm shift is primarily driven by the continuous flow and processing of data through interconnected devices. IoT's ability to gather, analyze, and act upon data in real-time offers immense potential for efficiency and innovation. However, this also presents unique challenges, particularly in ensuring timely data communication and maintaining privacy, especially in scenarios where resources such as computation, bandwidth, and energy are limited [1].

As shown in Fig. 1, cloud computing has offered substantial benefits in managing IoT data, but it also has some serious limitations, particularly related to latency and data privacy [2]. Edge computing emerges as a viable alternative, extending cloud capabilities closer to the data source through devices such as single-board computers or computational accelerators owned by end-users. This approach is particularly advantageous for IoT applications where real-time low-latency data processing is critical. Integrating edge and cloud models into a unified 'hybrid edge-cloud' structure also presents an efficient approach. This combined model offers the best of both the paradigms: speedy processing and augmented computational power, greatly benefiting IoT applications. The key benefits of this architecture include reduction in delays, improved dependability, and superior Quality of Service (QoS) [3].

As IoT environments are becoming increasingly complex, especially with the integration of artificial intelligence (AI) and machine learning (ML); there is a rising demand for platforms capable of handling sophisticated applications. AI and ML integration significantly enhances the capabilities of IoT systems, especially in optimizing processes using data collected at the network's edge. Moreover, in the edge-cloud

* Corresponding author at: Cardiff University, United Kingdom.

https://doi.org/10.1016/j.future.2024.107696
Received 10 December 2023; Received in revised form 20 May 2024; Accepted 22 December 2024
Available online 31 December 2024

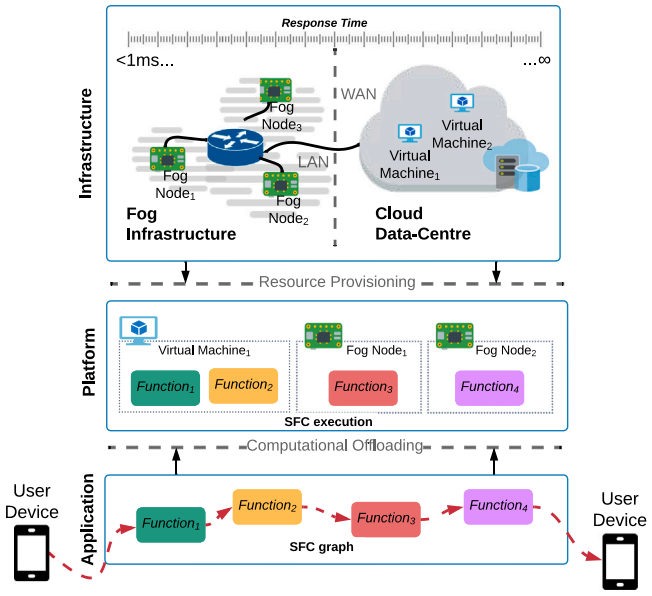


Fig. 1. The process of IoT computational offloading across application, platform, and infrastructure layer.

ecosystem, an application consists of smaller components known as virtual functions, which can be deployed across entire IoT-based infrastructures for execution. Deciding the optimal location for executing these virtual functions is also a complex task. This decision-making process requires focus on analyzing specific needs of the application, efficient use of available infrastructure, and the avoidance of congestion points that could slow down the entire IoT environment.

The Greedy Nominator Heuristic (GNH) [4] is a performance optimization algorithm specifically designed for infrastructure setups including IoT, fog, and cloud computing infrastructures. At its core, GNH integrates key components: a function for assessing similarity with the ideal solution for comparing solutions using a similarity function [5], MapReduce's mappers and reducers for data processing, and a master-worker mechanism with controllers and workers for task management [6]. The similarity function in GNH, inspired by a context-sensitive distance measurement approach, is critical in differentiating solutions from various optimization methods. Mappers in GNH are designated to specific nodes, identified as locations, responsible for processing workflow functions and creating decision variables. During a placement query, mappers suggest suitable nodes for deployment. Following this, a reducer assesses these suggestions and picks the most appropriate nodes; a process that continues until the entire workflow is efficiently deployed.

Both the mappers and reducers adopt a greedy strategy [7], utilizing a euclidean distance-based similarity function as a benchmark for comparing current solutions with an ideal one. GNH is adaptable and can work with various other similarity functions, like cosine similarity [8] and fuzzy measures [9], provided they can effectively gauge the likeness between the ideal solution and the ones being evaluated. The max-heap also plays a crucial role in GNH. It is a binary tree format that organizes results from mappers and the reducer, ensuring the most significant value always resides at the root. This structure is particularly beneficial for the quick and efficient retrieval and removal of stored solutions. Both mappers and reducers navigate through the search space, continually updating the max-heap with their findings. This approach in GNH allows for a more effective and streamlined process in selecting optimal nodes for deployment for complex IoT-based edge and cloud computing environments. The framework design is structured around a controller, serving the role of the reducer, and workers functioning as mappers. These workers are tasked with monitoring the network's performance and the available computational resources at various sites. The controller then uses this information to select the most suitable locations for deployment. This system is implemented using an asynchronous parallel programming library in Python (Parsl [10]), designed to facilitate execution on both high performance and edge computing resources.

GNH has shown its versatility and efficiency in intelligent IoT applications in varied settings. In smart city scenarios [11], GNH has proven its capability by autonomously deploying virtual functions across both edge and cloud environments. This deployment strategy led to more efficient resource utilization, enhanced execution performance, and a noticeable reduction in operational costs. Additionally, GNH has been adapted for use in federated learning (FL) based frameworks, particularly in rural areas with limited network connectivity [1]. In these applications, GNH efficiently balanced resource usage and performance across a range of IoT applications. Despite these successes, it has been observed that GNH sometimes struggles in ensuring resource availability, highlighting an area for potential enhancement. This observation points to the need for ongoing development and refinement to fully harness GNH's capabilities in varying IoT environments.

This paper introduces an advanced optimization algorithm, the Enhanced Optimized Greedy Nominator Heuristic (EO-GNH), designed to minimize execution time and optimize resource usage in IoT applications. EO-GNH is an updated version of the original GNH algorithm, offering several improvements. This framework incorporates asynchronous parallel computing alongside machine learning techniques for better heuristic selection. EO-GNH's development involves a simulation-based evaluation, enabling a thorough analysis of its performance across diverse IoT environments and potential failure situations.

While GNH is effective, its efficiency in generating pareto-optimal solutions is comparatively limited. EO-GNH addresses this limitation by utilizing asynchronous MapReduce and parallel meta-heuristics. It significantly reduces the execution time of the optimization algorithm and helps in avoiding local optima thus ensuring consistent service availability. EO-GNH enhances the jMetalPy framework [12] through the integration of Parsl [10], a tool that optimizes its functionality for real-time IoT applications. A key feature of EO-GNH is its capability to select optimal, non-dominanted solutions from the pareto front approximations produced by its mappers. At the reducer level, EO-GNH enhances optimization with a greedy approach that aggregates objectives as a scalarization method [13], leading to quicker, more dynamic results, ideal for complex and real-time IoT scenarios requiring efficient decision-making.

EO-GNH functions within a proposed framework for IoT application management, aiming to provide balanced solutions in terms of delay, cost, and risk. The algorithm's core principle is to improve one aspect without negatively impacting others. This paper examines whether EO-GNH can adapt to the varied characteristics and infrastructure needs of different IoT applications, a critical question for the evolving landscape of IoT technology. The investigation forms the central focus of our work, contributing significantly to the field of IoT application management. We validate the efficient and optimized deployment of IoT applications in edge-cloud infrastructure by leveraging various ML-based use-cases.

The main highlights of this work are as follows:

- **Innovative Asynchronous Optimization Model:** Development of EO-GNH as an advanced optimization algorithm using asynchronous MapReduce and meta-heuristics for efficient distributed computing.
- **Integration of Parsl for Dynamic Resource Utilization:** Implementing Parsl within EO-GNH for adaptive and scalable resource management, enhancing computational efficiency.

- **Superior Performance in Time-Critical Applications:** Demonstrating ability of EO-GNH to outperform traditional algorithms like dNSGAII in execution time and success rate, making it suitable for time-sensitive tasks.
- **Robustness Against Variability in Computational Environments:** EO-GNH maintains consistent performance across different mapper setups, proving its reliability in various computational contexts.
- **Effective Balance Between Resource Utilization and Operational Risk:** EO-GNH's framework adeptly balances resource usage while managing operational risk, a very crucial factor in real-time optimization problems.
- **Optimization Across Various ML Applications:** We tested our framework over multiple scenarios, including cloud computing, edge computing, and IoT; highlighting its versatility in solving various optimization challenges across wider range of domains.

The remainder of the paper is organized in this manner. Section 2 provides the description of other optimization approaches that have been developed for ML and AI tasks on edge and cloud environment. Section 3 delves into the design of framework and application use-case considered in this work. A mathematical description of the problem is provided in Section 4. Section 5, 6, 7, and 8 describes the EO-GNH framework and details the experimentation conducted for analyzing the designed framework. The results and evaluations are depicted in details in Section 9. Section 10 concludes the work and provides the future scope of work that can further improve the implemented approach.

2. Optimization algorithms for AI/ML applications

In the field of task and function placement algorithms, the use of informed approaches like heuristics and meta-heuristics is crucial for optimizing placement strategies [14,15]. Heuristic methods, customized for specific problems, leverage heuristic functions to develop algorithms that effectively tackle placement challenges. These approaches benefit from insights provided in the works of Pearl et al. [16] and Almurshed et al. [1]. Meta-heuristics, in contrast, offer problem-independent frameworks, requiring only that solutions be defined in a format understandable by the algorithm. This broadens their usability across various problem types.

The combination of heuristics and meta-heuristics enhances adaptability and effectiveness. For example, the A* search algorithm uses a genetic algorithm as a heuristic function, an approach explored by Yiu et al. [17].

In optimization, two key methods are scalarization and non-dominanted solutions, also known as the Pareto method. Scalarization [5] merges multiple objective functions into one. Non-dominanted solutions in contrast, seek outcomes that do not surpass others in every objective. Combining Meta-Heuristic and Hierarchical Heuristic methods, as described by Yiu et al. [18], offers a balanced solution, incorporating the benefits of both scalarization and non-dominanted solutions.

Integrating machine learning with heuristic functions can further enhance results. Machine learning models, acting as function approximators [19], provide valuable data that can guide heuristic algorithms. The efficiency of these combined methods can be increased through Parallel Meta-Heuristics. This approach, detailed by Alba et al. [20], involves the parallel operation of interconnected components like heuristic functions, function approximation, and various meta-heuristic elements. Parallel libraries such as Parsl [10], Apache Dask [21] and Apache Openwhisk [22]. Also, open-source meta-heuristic libraries like jMetalPy [12], facilitate this efficient planning process.

Utilizing parallel algorithms to deploy AI solutions has proven effective in various applications. A notable example includes image processing, where Apache OpenWhisk has been used to enhance processing across the edge-cloud continuum, as demonstrated in the work by Alabbas et al. [23]. This approach exemplifies the efficacy of parallel computing in AI-driven scenarios like edge device applications. Similarly, the use of Parsl alongside Docker containers has been instrumental in advancing smart city applications, showcasing the versatility and power of parallel computing [11].

For applications and platforms dealing with federated learning at the network edge, Baughman et al. [24], Patros et al. [25] and Almurshed et al. [1] provide insightful examples. Baughman et al. developed a serverless FL framework optimized for remote endpoints, employing tournament-based pretraining to boost model efficacy. Patros et al. [25] applied serverless FL in agricultural settings through their Rural AI initiative, proving its utility in weed detection. Both instances exemplify the versatility and impact of FL in edge computing environments. Almurshed et al. [1] explored the application of the GNH in rural environments, focusing on training and fine-tuning machine learning models using FL. Their research demonstrates GNH's effectiveness in orchestrating workflows in areas with fluctuating network conditions, showcasing its enhanced performance in managing decentralized FL tasks.

Moreover, the development of a dynamic platform that manages infrastructure and adapts to changes is crucial in contemporary technology landscapes. In this context, Kaushal et al. [26] proposed an approach characterized by its adaptive capabilities in case a network failure occurs. Such a platform significantly enhances the reliability of precision agriculture applications, while also considering the performance of ML task deployment.

More in adaptive resiliency, key studies provide significant advancements in this field. Mohammadi et al. [27] explore the Social Internet of Things, presenting a fault-tolerant framework bolstered by fog computing, which markedly enhances network reliability through advanced algorithms. Amjad et al. [28] tackle dynamic orchestration in IoT devices, emphasizing adaptability and interactivity, particularly pertinent in emergency scenarios and evolving business processes. George et al. [29] review the reliability aspect of edge computing in IoT networks, highlighting the necessity of resilient designs in vital sectors like healthcare. These studies serve as exemplary instances of adaptive resilient edge-cloud systems, showcasing the integration of robustness and flexibility in IoT infrastructures. A summary of recent works for optimization of AM/ML applications is provided in Table 1. The integration of heuristics, meta-heuristics, and machine learning, supported by parallel computing techniques, creates a comprehensive framework for optimizing task and function placement algorithms. This holistic approach utilizes the strengths of each method, resulting in a system that is efficient, adaptable, and effective.

3. System overview

For an IoT application platform to be adaptive, it must integrate four essential elements. Firstly, there should be a sophisticated scheduling planner for executing applications, supported by various system algorithms or optimization tools. Secondly, the platform needs to efficiently handle placement activities and data flow through a distributed processing system, offering workflow feedback that aids in decision-making analysis. Thirdly, the platform requires robust monitoring tools to collect and purify system data, including information on resources like RAM, CPU, and the status of service functions. Lastly, it is important for the platform to have an analysis component that can derive meaningful insights from the collected data. This could involve using the Integer Linear Programming (ILP) method for scheduling optimization, with the results from this analysis phase feeding back into the optimization strategy, enhancing its effectiveness.

In the evolving landscape of IoT, an adaptive system is crucial. By utilizing cutting-edge tools and methodologies, such a system can adeptly handle scheduling, placement, monitoring, and data analysis of IoT tasks in the framework. This results in a platform capable of responding to the dynamic nature of the IoT environment. Parsl [10],

Table 1
Summary of recent works categorized by application use-case, management approach, ML usage, performance objectives, and adaptation characteristics.

Work	Real-life application	ML based task	Self optimization	Self configuration	Cross-Domain evaluation	Risk handling	Meta-Heuristic based	Function chain	Server-less	Resource management
[23]	✓	✓						✓	✓	
[11]	✓	✓	✓	✓				✓	✓	
[24]		✓	✓						✓	
[25]	✓	✓							✓	✓
[1]	✓	✓	✓	✓				✓	✓	
[26]	✓	✓				✓				
[27]			✓					✓		✓
[28]			✓							✓
[29]			✓	✓		✓				✓
EO-GNH	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

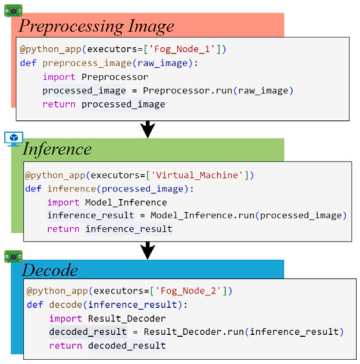


Fig. 2. Service function chaining implemented with Parsl. The *executors* argument in the function decorator specifies the location that runs the service function.

a notable Python library for parallel programming, executes Service Function Chains (SFCs) within the infrastructure and generates detailed logs (as shown in Fig. 2). Its ability to use high-level programming languages for scripting SFCs enables precise depiction of both the SFC graph and its functional logic. The feedback provided by such tools is vital for accumulating critical knowledge about the infrastructure (see Fig. 3). Parsl employs Python functions, known as Parsl apps, to act as service functions in SFCs. Managed by specialized executors, these apps offer asynchronous, non-blocking operations, returning AppFutures that enhance data flow and encapsulate service function logic within the Parsl framework.

Effective scheduling in an IoT system involves analyzing infrastructure data and previous scheduling outcomes. A combination of heuristics, meta-heuristics, optimization solvers, and libraries can be integrated with ILP models to refine scheduling processes. Optimization solvers use deterministic algorithms for solving structured mathematical problems to find optimal solutions. On the other hand, meta-heuristics employ stochastic and iterative approaches for solving problems with vast search spaces or intricate objective function landscapes. In scheduling optimization, solvers are employed to create optimal schedules within certain constraints, whereas meta-heuristics offer flexible solutions for dynamic environments with changing requirements.

The combination of Parsl with an optimization solver, enriched by their respective feedback systems, enables the creation of a self-adjusting platform well-suited to the dynamic nature of IoT systems. This is illustrated in Fig. 4 of our proposed work. This adaptive system is structured around three main components: (1) the deployment of SFC within the infrastructure, (2) monitoring mechanisms that provide essential data for decision-making, and (3) a decision-making process that focuses on analyzing and optimizing the scheduling plan using an ILP model.

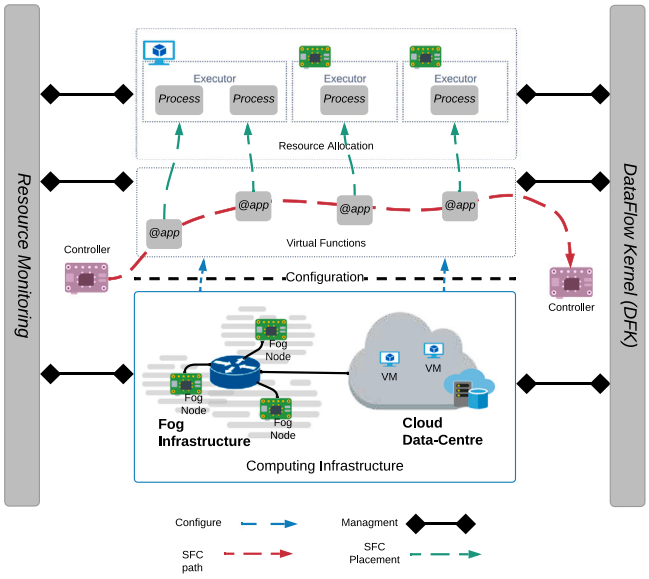


Fig. 3. Pipeline service in a fog-cloud environment with Parsl.

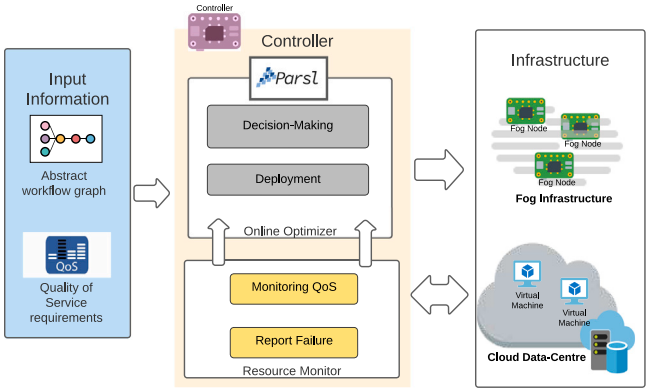


Fig. 4. Controller node supervising the placement process.

The Monitoring component revolves around data collection and storage, utilizing a suite of tools for effective consideration. This includes Stop Watch and Psutil libraries that are instrumental in retrieving information on running processes and system utilization, as cited in [30–32], respectively. Additionally, Parsl logs [10] play a crucial role in capturing detailed records of function execution and outcomes, ensuring comprehensive data storage.

Table 2
Notations used in the EO-GNH problem formulation.

Symbol	Description
F	The set of all functions in the workflow
D	Functions dependency pair set
A	The graph of the workflow, which is $A = (F, D)$
L	The set of all locations that execute functions
i	The sequence of function f_j^i in the workflow
j	The type of the function f_j^i
k	The index of Location l_k
$T_{j,k}$	Processing time for f_j in location l_k
$x_{j,k}$	Binary variable: function f_j^i is executed on location l_k
$y_{j,k}$	Binary variable: $T_{j,k}$ contributes to longest path
o_k	Binary variable: location l_k is used by the application
E	The set of all placed paths of A
$Risk_{j,k}$	Risk of executing f_j in location l_k
$MR_{i,j}$	The maximum replicas for f_j^i (MaxReplicas)
m	Constant to adjust maximum possible replica
r_k	The number of failures per allocations

For Decision making, the focus shifts to analyzing the gathered data and formulating future actions. This phase involves critical data cleaning to maintain data accuracy and integrity. The ILP model is utilized for transforming utility data, with optimization strategies such as greedy heuristics [7] and JmetalPy [12] aiding in efficient scheduling decisions. These combined methodologies ensure a robust and strategic approach to decision-making within the system.

In the Deployment phase, the strategies and actions planned during the decision-making process are put into action. Parsl [10] is utilized to execute SFCs within the infrastructure, facilitating the operational execution of planned tasks. Furthermore, brokers like Apache Kafka [33] and Samba [34] are employed to manage and streamline data flow between various components of the system. These brokers are key to ensuring the smooth execution of actions and maintaining uninterrupted operational flow across the system.

Together, these three components integrate seamlessly to form a dynamic and responsive adaptive system. Each component, equipped with its specialized tools and processes, contributes to the overall efficiency and effectiveness of the system, enabling it to adequately respond to diverse operational demands and challenges.

4. Problem formulation

In order to represent our task problem mathematically, we have defined the optimization problem as a minimization of three distinct factors: *C, R,* and *O*. All the symbols used in the formulation with their brief descriptions are provided below in Table 2.

$$C = min\left(\sum_{j\in F}\sum_{k\in L} T_{j,k}\cdot y_{j,k}\cdot x_{j,k}\right) \tag{1}$$

$$R = min\left(\sum_{j\in F}\sum_{k\in L} Risk_{j,k}\cdot x_{j,k}\right) \tag{2}$$

$$O = min\left(\sum_{k\in L} o_k\right) \tag{3}$$

where the objective function C (formula (1)) represents the minimization of aggregated processing time, and function R (formula (2)) denotes the minimum risk associated with the successful execution of application A. The minimum number of nodes utilized for execution (inclusive of redundancy) is depicted by objective function O (formula (3)).

5. EO-GNH framework

The EO-GNH optimization algorithm seeks solutions from the Pareto Front using a combination of MapReduce and meta-heuristic techniques. This approach is used to locate optimal locations for redundant deployments. Within this framework, workers (known as Mappers) utilize meta-heuristics to explore various decision variables. The outcomes from these Mappers are then communicated to the Reducer, typically a controller edge/fog node, which is responsible for selecting the most appropriate locations for redundant deployment.

Throughout the execution of the SFC graph, Mappers are instrumental in providing solutions that are not dominated by others. When it becomes necessary to decide on the placement of a function within the system, the Reducer uses the Mapper's findings to formulate an effective solution. This process ensures a balanced and efficient approach for deploying functions within the network, leveraging the combined insights of both Mappers and the Reducer.

Table 3
Dataset used to train the decision tree.

Attribute name	Values		
SFC size	5	10	20
Location size	100	500	1000
Population/swarm size	10	50	100

5.1. Components of the framework

The execution of the **parallel model** operates on an asynchrony basis, where the *Reducer* starts making placement decisions without waiting for every *Mapper* to finish their meta-heuristic cycles. Each Mapper writes its current optimal meta-heuristic solutions to a file, to which the Reducer has read access. As soon as a Mapper arrives at a solution, the Reducer can use it, irrespective of its quality, to make placement decisions for individual functions. *Parsl* [35] is employed to facilitate this MapReduce computing process.

In computer science, **The Oracle** is a software tool utilized for specific answers [36]. Here, It provides insights into the compatibility of meta-heuristics with various application and infrastructure setups. It also provides the most suitable meta-heuristics based on objectives like makespan, cost, and risk. The query variables include the number of Mappers, the size of the SFC, the number of locations, the size of the population, and criteria preferences. The Oracle utilizes decision tree models to offer an estimated response based on the historical performance of meta-heuristics. It operates in three phases: (i) inferring decision trees, (ii) employing preference-based sorting, and (iii) assigning meta-heuristics to Mappers. Fig. 5 illustrates The Oracle's interaction with the MapReduce methodology.

A **decision tree** is applied to predict the effectiveness of a meta-heuristic. This tree functions as a unit-wise constant approximation as mentioned in [37]. Its prediction is an estimated output of the objective function of a meta-heuristic like NSGAII (Non-Dominated Sorting Genetic Algorithm II). The model was trained on a dataset obtained from benchmarking a range of meta-heuristics, evaluated against SFC of varying sizes and locations, as provided in Table 3. These features, along with the benchmark results, are used to train the decision tree. This benchmarking process is executed through a simulation, taking inputs (features) as mentioned in Table 3 and combining the outputs with these inputs to form the dataset for the training phase.

Meta-heuristics refer to a category of heuristic algorithms designed to tackle broad spectrum of problem types. In our approach, we utilize a selection of nature-inspired meta-heuristic algorithms. Algorithms such as Generalized Differential Evolution 3 (GDE3) [38], Hypervolume Estimation Algorithm (HYPE) [39], Indicator-Based Evolutionary Algorithm (IBEA) [40], Multi-Objective Cellular Genetic Algorithm (MOCell) [41], and Non-dominated Sorting Genetic Algorithm II (NSGAII) [42] utilize Genetic Algorithm meta-heuristics. In contrast, Multi-Objective Particle Swarm Optimization with Crowding Distance (OMOPSO) [43] and Speed-constrained Multi-objective Particle Swarm Optimization (SMPSO) [44] uses Particle Swarm Optimization (PSO) in their implementation. These algorithms are dynamic during execution,

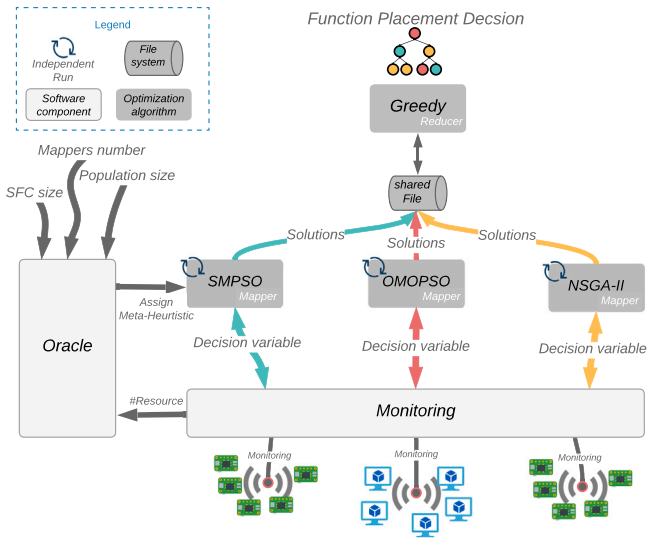


Fig. 5. Asynchronous MapReduce performs EO-GNH, initiated by the Oracle. The Oracle ranks meta-heuristic algorithms according to their attributes.

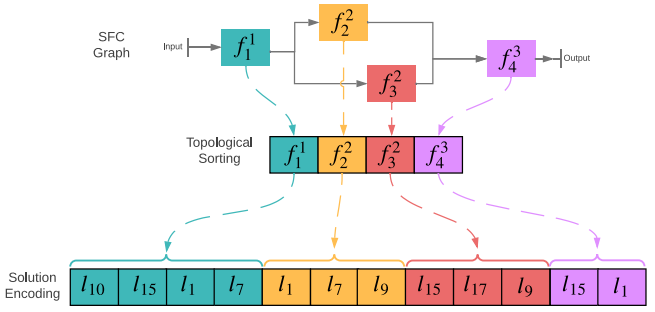


Fig. 6. SFC redundant deployments solution encoding. The solution encoding's elements are location IDs, while an array's indices specify the function.

yielding multiple sets of solutions, each representing an approximation of the best-discovered Pareto Front. Meta-heuristics typically consist of a series of main loops, with each iteration executing a specific step. The nature of these steps varies across different meta-heuristics. For instance, PSO based algorithms simulate a swarm, while Genetic Algorithm (GA) based ones apply genetic operators. In our designed parallel model, the Pareto Front discovered at each step of the meta-heuristic's process is recorded in a shared file.

Solution Encoding refers to the technique used by meta-heuristic algorithms for representing solution data, encompassing the data's format and structural organization. In our framework, as illustrated in Fig. 6, solution encoding takes the form of an integer array. Here, each array value denotes a location ID, while the array indices correlate with functions in the graph. This setup allows for multiple indices to correspond to a single function, effectively outlining a redundant placement strategy for that function. In the context of PSO-based methods, the solution is initially in the form of a real number (floating type), which is inherently is not an integer. To address this, we implement discretization techniques, specifically using the mathematical floor function, to convert these array elements into integers by eliminating their decimal components.

Timsort is an advanced, stable hybrid sorting method derived from both merge sort and insertion sort. It is designed to efficiently handle a diverse range of real-world data types. In our system, Timsort is utilized to organize the decision tree outputs according to user-defined objectives and preferences, such as makespan, cost, and risk. Regarding

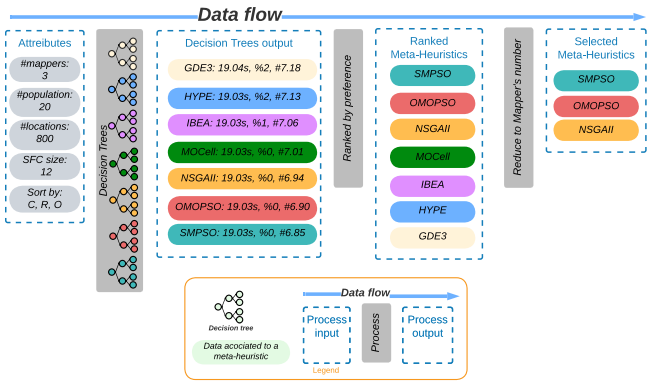


Fig. 7. The Oracle ranks and selects meta-heuristics, each has a color. Inputs for the oracle are the number of SFC mappers, population, locations, and functions.

the **greedy** methodology, it is integrated into EO-GNH, enhancing the original GNH framework. In EO-GNH, the greedy heuristic elements, known as Reducers, process the output files generated by each meta-heuristic. The planning stage for deploying the next function in the sequence is initiated once the current function's execution is complete, ensuring a sequential and efficient deployment process.

5.2. Workflow of the framework

The functional architecture of EO-GNH revolves around the utilization of Oracle and asynchronous MapReduce techniques. Its workflow is segmented into three key stages: (i) engaging with the Oracle, (ii) setting up and triggering the meta-heuristics, and (iii) running the application while dynamically adjusting to evolving conditions. Subsequent sections will provide detailed insights into these critical stages, which are integral to the methodology's overall execution.

5.2.1. Engagement with Oracle

The Oracle's purpose is to determine the most suitable meta-heuristics for operating an application, taking into account the specific characteristics of the existing infrastructure. As shown in Fig. 7, it constructs a query using several key factors. These include the controller's capabilities, the specific parameters of the algorithm, data related to the location, the structure of the SFC graph, and the preferences of the user regarding objectives. The user's preferences also play a crucial role in how the results are prioritized and presented.

Decision tree models are employed to project the anticipated outcomes of different meta-heuristics, focusing on crucial aspects such as makespan, risk, and cost. Following this, the meta-heuristics undergoe an assessment and are ordered according to how well they align with the user's priorities. The process saturates with the assignment of the top-ranked meta-heuristics to the mappers at hand.

5.2.2. Engagement with meta heuristics

Unlike the Mappers in the original GNH, the EO-GNH approach involves making decisions for the entire SFC graph, as depicted in Fig. 6. This decision-making strategy is outlined in Algorithm 1 of our paper. It describes the process of the Mapper initiating the meta-heuristic (referenced at line 3) and then moving into the main loop of the algorithm.

Within this loop, there's a continuous update of decision variables (seen in lines 5–7), leading to simultaneous changes in the current values of the objectives (outputs from the objective functions). The *Step* procedure, as mentioned in line 8, employs the previous solution along with locations and the SFC to perform an individual step.

This approach results in the generation of a new solution, which is then stored in a file shared with the Reducer (lines 9–10). The

Algorithm 1 The *solution* is an array where each index refers to the function (f_j^i), whereas its content is the location *id*

```
1:  class MAPPER
2:      method MAP (L; SFC)
3:          solution ← INITSOLUTION( L; SFC )
4:          while NOTCOMPLETED do
5:              if IsDecisionVarChanges(L; SFC): then
6:                  DecisionVar ← UPDATEVAR(L; SFC)
7:                  solution ← UPDATESOLUTION(solution)
8:              solution ← STEP(L; SFC; solution)
9:              MapperResult ← SOLUTION
10:             solution ← UPDATESHAREDFILE(MapperResult)
```

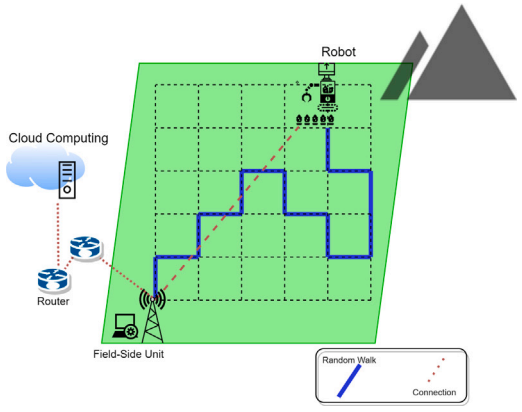


Fig. 8. Robot performs a random walk in agricultural field, affecting the connection to a field side unit.

algorithm concludes once the execution of the SFC is fully completed, as indicated when the NOTCOMPLETED status in line 4 switches to *False*.

The specific operations within the *Step* method differ depending on the type of meta-heuristic being used. For algorithms derived from PSO, the sequence includes (i) updating the velocity, (ii) updating the position, (iii) evaluating the objectives, (iv) updating the global best, and (v) updating the particle best.

In contrast, for GA-based algorithms, the sequence involves (i) selection, (ii) crossover, (iii) mutation, (iv) objectives evaluation, and (v) replacement. In the GA context, the replacement step is crucial as it updates the solution while favoring the superior offspring, thereby promoting evolutionary advancement.

6. Service deployment use-cases

This section outlines three different IoT applications utilized for evaluating the performance of proposed EO-GNH approach. These applications were analyzed with a focus on their workflows, particularly emphasizing the execution times on Raspberry Pi 4B devices, thereby demonstrating their variable requirements and practical functionality in real-world scenarios.

6.1. Federated learning (FL) based rural-AI application

Data-driven technologies in precision farming offer significant prospects for enhancing agricultural productivity and results. A notable example is the application of automated weed control [45]. By accurately identifying and managing weeds, this technology can boost farm yields, cut labor costs, and reduce pesticide usage, leading to more efficient and sustainable agricultural practices. Aiming to promote farm independence, this application utilizes FL, a more secure alternative to traditional ML methods [46].

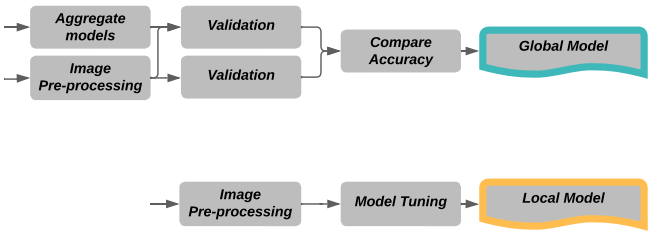


Fig. 9. Workflows for FL-based training and inference.

However, rural areas are often characterized by their limited network infrastructure, they also pose distinct challenges for implementing precision farming technologies [1]. Traditional machine learning algorithms may struggle in these environments, potentially compromising network reliability and service availability. Therefore, a solution that functions effectively within these network constraints is essential for maintaining consistent and successful precision farming operations.

In this setting, robots functioning as edge devices, are designed to improve field coverage and data collection, contributing to a collective model without the need to exchange raw data [47]. The movement of these robots is managed using a truncated random walk method, ensuring field coverage and task precision. Importantly, the distance of these robots from computing resources acts as an indicator of network latency in mobile edge devices, a factor that can influence the quality of data transmission, as illustrated in Fig. 8.

The FL workflow as depicted in Fig. 9, is broken down into distinct functions, each playing a vital role. These functions are outlined as follows:

- **Image Pre-processing:** This initial step involves modifying the color mode, resizing images, formatting data, and scaling pixel values to optimally prepare images for FL models. Subsequently, these processed images, along with their labels, are stored for future reference.
- **Model Tuning:** In this phase, the weights of a neural network are fine-tuned with a new dataset to enhance the existing model's performance. The refined weights are then preserved separately for subsequent applications.
- **Model Aggregation:** This process involves the amalgamation of parameters (specifically weights) from multiple trained models, often through averaging, to form a singular, aggregated model.
- **Validation:** Here, the efficacy of the trained machine learning model is evaluated against new data, utilizing metrics such as loss and accuracy. These metrics are also reported alongside the model.
- **Accuracy Comparison:** This function entails comparing the accuracy and loss function across various models to ascertain the most effective one, resulting in the selection of the best-performing model.

We utilized FL in this scenario, where the data is generated at different timestamps and initially trains a linear model locally. We then aggregate the weights to form a global model, as illustrated in Fig. 9. To simulate the real-world scenarios where data becomes available at different times, we tested our approach using the DeepWeeds dataset [48], which contains 17,509 images of various weed species, to generate local models. We use a ResNet50 model with a residual network architecture of 50 layers for evaluation in this use-case. Moreover, to provide a practical context, the average execution times on Raspberry Pi devices for these five functions have been recorded as follows: *0.33* s for Image Pre-processing, *178.16* s for Model Tuning, *22.33* s for Model Aggregation, *37.16* s for Validation, and *0.10* s for Accuracy Comparison.

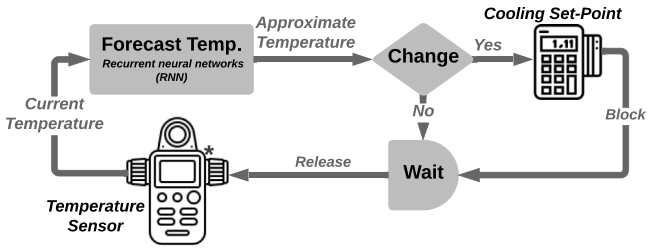


Fig. 10. Forecasting temperature control via sensors, providing data for prediction and set-point updates.

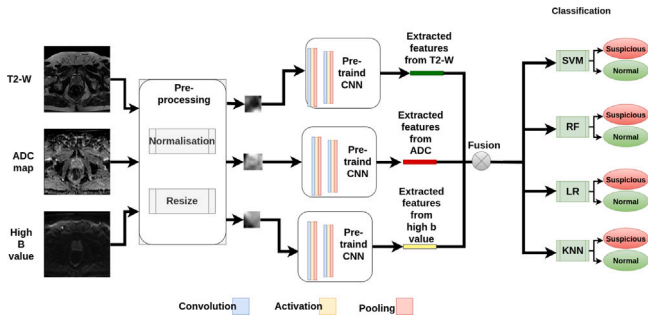


Fig. 11. Intelligent decision support system for prostate cancer diagnosis.

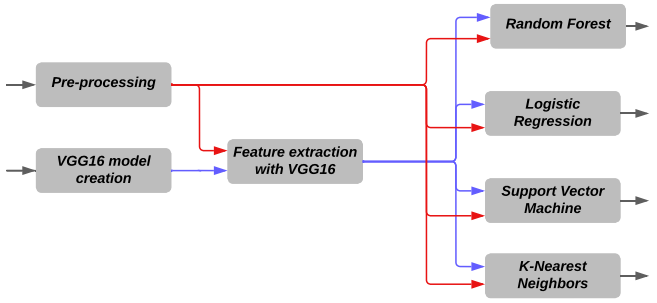


Fig. 12. Workflow for prostate cancer classification using ML.

6.2. Recurrent Neural Network (RNN) based smart energy application

In the complex and regulation intensive world of food production, innovative approaches are essential to meet the high standards required. Climate-controlled storage systems are particularly vital in ensuring the proper preservation of food products.

IoT-based systems for monitoring temperature, as shown in Fig. 10, are instrumental in providing real-time updates on storage conditions, thus ensuring compliance with necessary standards and maintaining optimal environments for food preservation. Another example is a food processing facility that utilizes a smart energy management system to enhance energy efficiency. In our forecasting model, a preprocessing function prepares the raw data by scaling and encoding its values. A series of neural networks process this data, encapsulating each in distinct functions known as neurons — x1, x2, x3, x4, and x5. The neuron processes a variety of operational inputs from past and projected future states such as set points, energy usage, capacity, average temperatures, and seasonal data. Then, we apply a hyperbolic tangent (tanh) function as its activation, which processes the output of these inputs through matrix operations, such that the specific parameters of the neuron effectively normalize the results and manage gradients. Neurons apply specific transformations to extract features necessary for output predictions. The refined data from these layers is then fed into predictive functions for forecasting the system's average temperature and energy consumption.

The structure of the RNN includes an input layer responsible for receiving sequential data, one or more recurrent layers that capture temporal dependencies over time through hidden states, and an output layer that generates predictions or forecasts for energy consumption and temperature levels. Before feeding the data into the RNN, a data preparation step is typically performed to format, normalize, and split the dataset.

The workflow of a neural network designed for energy efficiency is outlined in the following steps:

- **Feature Scaling:** This process involves normalizing input data, such as the current settings of the chamber, power, capacity, and the prevailing season. The data is reformatted to a standard scale that is compatible with the RNN, facilitating more accurate model-based predictions.
- **Middle-Layer Neurons (X_1 to X_5):** These neurons are the core of the RNN's middle layer. They assign weights to the inputs and process them using a hyperbolic tangent (tanh) activation function. The outputs from the feature scaling stage are modified here, integrating the weighted values with the activation function's output.
- **Predicting Energy and Temperature:** These two crucial functions forecast energy usage and temperature, respectively. Each function comprises an output layer, an unscaling layer, and a bounding layer. The output layer sums up the outputs from the X_j layers, adjusted by their weights. This sum is then reverted to its original units (kilowatt-hours for energy and degrees Celsius for temperature) in the unscaling layer.

A private fish facility dataset offering a time-series analysis of temperature and energy, using a six-step sequence per training cycle is utilized by the RNN model. Additionally, to provide an idea of the system's performance, the average execution times on Raspberry Pi devices for these three functions are recorded as 1.29 s for Feature Scaling, 0.44 s for the Middle-Layer Neurons, and 0.07 s and 0.06 s for the Energy and Temperature predicting functions, respectively.

6.3. Machine learning based cancer diagnostic application

Prostate Cancer (PCa) ranks as the second most common cancer in men globally, with about 1.4 million new diagnoses in 2020. The role of AI in precise disease categorization is increasingly recognized as crucial for effective treatment and minimizing risks. Over the past century, cancer research has achieved significant advancements, particularly in diagnostic and therapeutic techniques for PCa. This advancement has resulted in a substantial accumulation of cancer-related data. However, accurately detecting cancer remains a complex challenge.

Presently, the application of machine learning methods (similar to Fig. 11) is showing remarkable success in interpreting intricate patterns and identifying various types of cancer. The ProstateX dataset [49], a component of the Cancer Imaging Archive (TCIA) [50], includes prostate MRI scans of 195 patients. It features expert-annotated, healthy and cancerous regions of interest within 32 × 32 pixels, specifically for cancer research.

Fig. 12 presents the machine learning training process for classifying prostate cancer. The steps involved in this workflow are as follows:

- **Data Pre-processing:** This stage entails retrieving data from a designated source and conducting initial cleaning and structuring of the data.
- **Formulating the VGG16 Model:** We utilize the technique of transfer learning by applying VGGNet, a renowned architecture with 16 layers. The model leverages pre-training on the ImageNet database, which contains an extensive collection of over 10 million natural images across 1000 different categories.

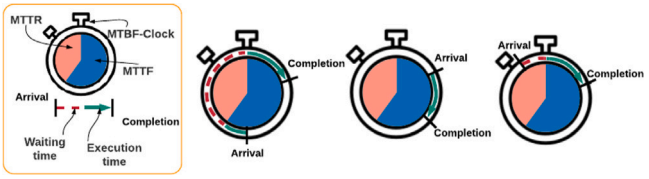


Fig. 13. The execution of a service function with completion times considering the MTTF and MTTR.

- **Feature Extraction from MRI with VGG16:** In this phase, the processed data is fed through the VGG16 model to extract features. Specifically, the first two blocks of the VGG model are employed for feature extraction. This process is conducted separately for each MRI modality before fusing the extracted features.
- **Constructing, Training, and Evaluating Machine Learning Models:** This final task involves setting up, training, and testing various models such as Random Forest, Logistic Regression, Support Vector Machine, and K-Nearest Neighbors from the Scikit-learn library on the Raspberry Pi. The models are trained using the previously extracted features and labels. Their performance is assessed using new, unseen data.

The average execution times on Raspberry Pi devices for these four functions of cancer detection model are recorded as *0.47* s for Data Preprocessing, *1.53* s for Developing a VGG16 Model, *10.23* s for Feature Extraction from MRI with VGG16, and *2.99* s, *28.09* s, *39.98* s, *7.65* s for Constructing, Training, and Evaluating Machine Learning Models, respectively.

7. Experimentation details

This section describes the experimental setup, failure model, and other evaluation factors considered during experimentation in this work. Each subsection below provides description about specified parameters in more details.

7.1. Evaluation criteria

During evaluation, we mainly considered two aspects in this work: (i) *algorithmic execution efficiency* (Section 8) and (ii) *quality of generated solutions* (Section 9). Algorithm efficiency is related to *algorithm speed*, *algorithm memory overhead*, and *Pareto Front volume*. Moreover, quality of the solution is associated with *makespan* – defined as the total duration required to execute a workflow, *risk* pertains to the likelihood of completing tasks within the set deadline, and *cost* interpreted as the utilization of locations.

7.2. Testbed setup

This section outlines the simulation models and methods applied to approximate real-world conditions, which are fundamental for assessing the algorithm's performance. It further presents the wireless and failure models utilized to evaluate the algorithm within a controlled setting.

7.2.1. Failure model

This section details the failure model employed in our evaluation. It specifies the conditions under which each processing node (i.e. fog node) encounters failures or recovery events. We simulate fog node failures to evaluate and validate the performance of a scheduler using federated learning application. Note that software defects caused by faulty service function implementations are not included in this model.

A task failure is considered to occur if a fog node fails to respond to requests within a set time threshold. The delay in response contributes to the overall recovery time required to restore the fog node. The Mean Time To Failure (MTTF) represents the average operational time of the system, while the Mean Time To Recovery (MTTR) indicates the duration until the fog node is ready again to process tasks. The Mean Time Between two Failures (MTBF) is calculated as the sum of MTTF and MTTR.

Each fog node is equipped with a repeating MTBF-clock timer, which schedules the arrival of requests. The timing of the function request's arrival, as per the MTBF-clock, determines the success or failure of the allocated fog node. Successful deployment of a function means it commenced and completed within the node's MTTF period. Conversely, a deployment failure suggests that the function execution coincided with the node's MTTR period in the MTBF-clock. Fig. 13 illustrates service functions submitted across different MTBF-clock periods [1].

Table 4
Parameters and their default values for our Wireless Simulation.

Parameter	Default values
Frequency	2.4×10^9 Hz
Obstacle factor	1.0
Bandwidth	10^6 Hz
Noise power	10^{-9} Watts
Data size	10^6 bits
Fading coefficients	Rayleigh distribution (scale = 1, size = 100)

7.2.2. Wireless model

For the robotic weed detection agricultural application, we utilized a simulation that calculates the distance between robots and fog nodes. This distance is integral to evaluating the wireless connection quality. The connection quality is determined not only by distance but also by factors such as frequency, obstacles, and signal fading. These elements collectively affect the path loss and signal-to-noise ratio (SNR), which are critical in determining the effective capacity of the wireless link.

The simulation incorporates a detailed model for signal propagation and fading (values specified in Table 4). Path loss is calculated using both the free-space path loss (FSPL) formula (Eq. (4)) and additional loss due to obstacles, which is a function of distance. The formula for path loss is given by Eq. (5) [51].

$$FSPL = 20\log_{10}(Distance) + 20\log_{10}(Frequency) - 20\log_{10}(c) \tag{4}$$

In wireless communication, c represents the speed of light in a vacuum, roughly 3×10^8 meters per second, essential for signal propagation calculations.

$$LossdB = FSPL + AdditionalLoss \tag{5}$$

$$AdditionalLoss = (AdjustedExponent) \cdot (\log_{10}(Distance)) \tag{6}$$

The model also considers Rayleigh fading to simulate the impact of multipath propagation. The adjusted fading coefficients for path loss are computed as in Eq. (7) [51].

$$AdjustedFadingCoeffs = (Fadingcoefficients) \cdot (LossLinear) \tag{7}$$

Using these parameters, the simulation applies Shannon's formula, as shown in Eq. (8), to estimate the capacity of the wireless link [51].

$$Capacity = Bandwidth \cdot \log_2(1 + Average(SNR)) \tag{8}$$

Taking into account the bandwidth of the channel and the average SNR under fading conditions. The time to send data over the network is then calculated using Eq. (9).

$$TransmissionTime = \frac{Datasize}{Capacity} \tag{9}$$

This approach provides a more realistic estimation of the transmission time in a wireless network, considering various environmental and technical factors. The simulation moves beyond a simple distance-based quality metric to a more comprehensive model that includes path loss, fading, and SNR, offering a nuanced understanding of wireless network performance in different scenarios.

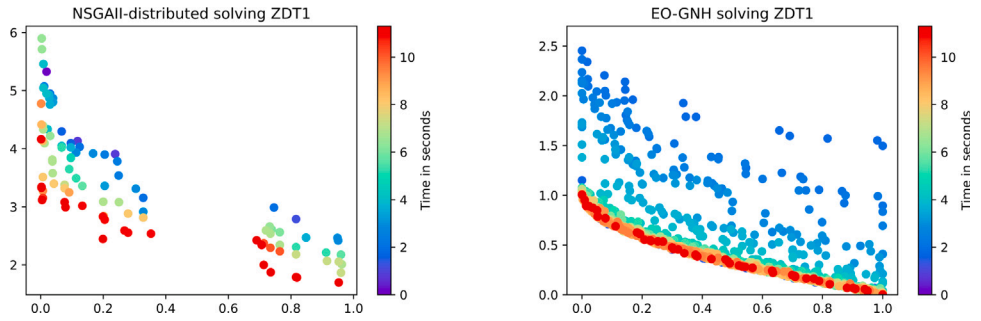


Fig. 14. Algorithm comparison: when solving the *ZDT1* problem, the color is assigned to the time the pareto front was collected.

7.3. Performance comparison

We compare parallel optimization approaches using Zitzler-Deb-Thiele-1 (ZDT1), a well-known synthetic benchmark problem [52]. Both algorithms leverage Python-based tools to enable parallel processing. EO-GNH, in particular, uses Parsl [10], while dNSGAII is built on Apache Dask [21]. For EO-GNH, the reducer is specifically set up to accumulate Pareto fronts from the mappers, and we periodically capture the Pareto Front to monitor how the algorithms progress over time.

The experimental setup involves Google's cloud server infrastructure, using a Jupyter notebook setup in Google Colaboratory. The virtual machine deployed for this purpose includes a single-core Intel(R) Xeon(R) CPU at 2300 MHz with 12 GB RAM and operates without GPU support. Both Dask and Parsl are optimized to fully utilize two cores; Dask directly, while Parsl assigns one executor per core.

In dNSGAII, NSGAII's operators like selection and reproduction are executed asynchronously. Each step in the dNSGAII process receives outputs from the previous operation and forwards them to the next. For example, the outputs of the selection phase are inputted into the reproduction stage. Even though dNSGAII operates asynchronously, its iterations are synchronized.

On the other hand, EO-GNH treats each NSGAII instance as an autonomous algorithm. The workers function asynchronously, and a master node periodically gathers the latest solutions. EO-GNH is scalable; when more resources become available, it increases the number of NSGAII instances to potentially improve performance results.

8. Performance comparison of EO-GNH

This section evaluates EO-GNH against the distributed Non-dominated Sorting Genetic Algorithm II (dNSGAII), analyzing their parallel structures, Pareto solution counts over time, and resource usage for Pareto front generation. Notably, both algorithms operate asynchronously. Within EO-GNH, multiple NSGAII instances function as Mappers and Reducers, enabling a comparative analysis of the non-dominated solution quantities produced by each algorithm.

8.1. Results

Within the first ten seconds of operation, EO-GNH demonstrates its capability to accumulate a greater quantity of solutions for the controller's use. The Pareto Front approximation by EO-GNH also shows considerable improvements in quality, as illustrated in Fig. 14. Remarkably, EO-GNH achieves in just two seconds what dNSGAII accomplishes in ten seconds.

Although dNSGAII operates asynchronously, its iteration process is synchronous. This approach leads to a reduced collection of non-dominanted solutions, a fact highlighted in Fig. 14. Additionally, the implementation of NSGAII operations in dNSGAII often faces delays due to queuing times.

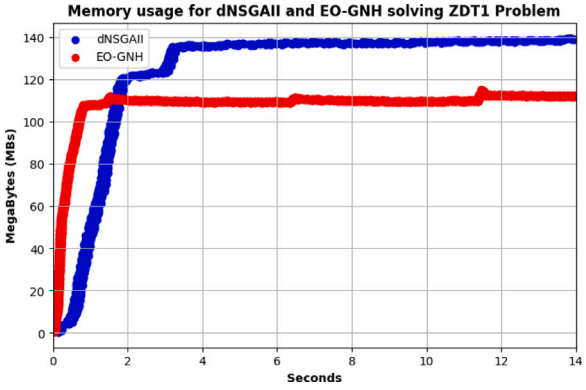


Fig. 15. Memory overhead over time for EO-GNH and dNSGAII.

In comparison, EO-GNH runs multiple NSGAII instances across two separate processes, minimizing the chances of any process idling while waiting for CPU allocation. This efficient allocation of computational resources enhances EO-GNH's overall performance.

The memory overhead associated with these algorithms over time, as shown in Fig. 15, indicates notable differences in their memory consumption patterns. EO-GNH exhibits an initial rapid increase in memory use, spiking from 0 to 60 MB in under a second. In contrast, dNSGAII shows a more gradual memory usage, reaching the same 60 MB mark in approximately 2 s.

After this initial surge, EO-GNH's memory consumption fluctuates slightly between 100 MB and 110 MB, maintaining this range for the duration of its operation. On the other hand, dNSGAII displays a steady increase in memory use from 120 MB to 140 MB after three seconds, sustaining this level for the rest of the operation.

During a single run, both EO-GNH and dNSGAII appear to use a consistent amount of memory. However, it is possible that EO-GNH's memory overhead might rise with the addition of more NSGAII instances. Conversely, dNSGAII's memory usage is expected to remain stable unless more computing cores are added, a scenario Dask is well-equipped to handle.

EO-GNH, with Parsl's flexibility, dynamically utilizes additional resources as they become available. In contrast, dNSGAII is designed to utilize all available resources right from the start, leading to a steady memory usage despite the availability of extra resources.

9. Performance evaluation of use-cases

In assessing the EO-GNH framework across various scenarios, we focus on the following key performance indicators: *(1) Success Rate:* this metric analyzes the probability of task completion within a set deadline. It works in parallel with risk assessment (the likelihood of failing to meet the deadline) to proactively reduce service interruptions. *(2) Makespan:* this measures the total duration required to

execute a workflow using distributed resources. The primary goal of the scheduler is to minimize makespan, thereby achieving the fastest completion time. *(3) Utilized Location (Cost):* this is determined by the quantity of resources employed in a workflow. The objective here is to achieve equilibrium in resource utilization, aiming to reduce network bottlenecks and effectively balance the trade-off between risk and cost associated with redundant deployments.

The deadline is specified based on the estimated execution time of the function and the estimated data transmission time, representing the expected end-to-end completion time. Missing this deadline is considered a failure, which influences the overall project duration, known as the makespan which is the actual end-to-end completion time. A higher success rate in meeting these deadlines leads to a shorter makespan [4].

The evaluation of the EO-GNH algorithm considered various configurations of EO-GNH Mappers, ranging from a single Mapper in EO-GNH-1 to a setup comprising four Mappers in EO-GNH-4.

9.1. FL based rural-AI

In order to evaluate the proposed framework, we tested it on three different applications performing ML or AI tasks on an edge-cloud infrastructure. The first application is a FL based Rural-AI application which is performing weed detection using an agricultural robot. The detailed evaluation of this application is as follows:

9.1.1. Model tuning

Fig. 16 summarizes the performance of various optimization algorithms, detailing their statistical explanation for a minimization problem. GNH algorithm's execution times are spread between 134.02 s and 310.79 s, with its mean and median performance at 159.30 s and 154.05 s, respectively, and a considerable interquartile range (IQR) of 26.87 s, suggesting a wider dispersion of values. GDE3 shows a similar pattern but with a slightly narrower range, a mean of 157.78 s, and a median of 152.75 s. HYPE and IBEA exhibit tighter performance clusters, reflected in their lower IQRs of 24.81 s and 23.99 s respectively. The MOCell and NSGAII algorithms further tightens the range, with IQRs of 22.99 s and 22.32 s, indicating a more concentrated set of execution times around their mean values, which revolves around 154 s.

OMOPSO and SMPSO maintain this trend of narrow IQRs at 21.48 s and 21.36 s respectively, with mean values just above 153 s, signifying consistent performance. The EO-GNH series, iterating from 1 to 4, consistently shows a median performance around 149 s and decreasing IQRs, which suggest a refinement in algorithmic efficiency with each successive version. The 95% confidence intervals for these algorithms are relatively tight, all below 0.77 s, indicating precise estimates of the mean execution times and underscoring the reliability of these algorithms in solving the considered minimization task.

Fig. 17 illustrates the performance disparities among three heuristic placement algorithms in terms of their execution time statistics. Random Placement demonstrates the broadest range of execution times, from approximately 134.05 s to 491.30 s, with an average time of 224.98 s and a median of 208.16 s. This is indicative of a considerable spread in data, as evidenced by an IQR of 86.61 s, the largest among the three algorithms. The Round Robin algorithm shows a slightly less varied range with execution times from around 134.16 s to 476.14 s, a mean slightly higher than Random Placement at 225.50 s, and a median of 212.41 s, coupled with an IQR of 85.48 s, which is marginally less than that of Random Placement.

The Greedy algorithm, while also having a wider range, showcases a notably better performance with a minimum and maximum execution time between 134.04 s and 440.46 s, a lower mean of 214.80 s, and a median of 199.13 s, indicating a more favorable central tendency compared to the other two algorithms. Its IQR of 76.01 s is the smallest, suggesting a tighter concentration of values. The 95% confidence intervals for these algorithms are relatively large, all above 2.30 s, which points to less precision in the mean estimation compared to algorithms with tighter data clustering. Nonetheless, the Greedy algorithm stands out with slightly more precision in its mean estimation, as seen in its 95% confidence interval of 2.33 s, which is comparably higher but in the context of a lower mean, suggests a more efficient performance in the minimization problem at hand.

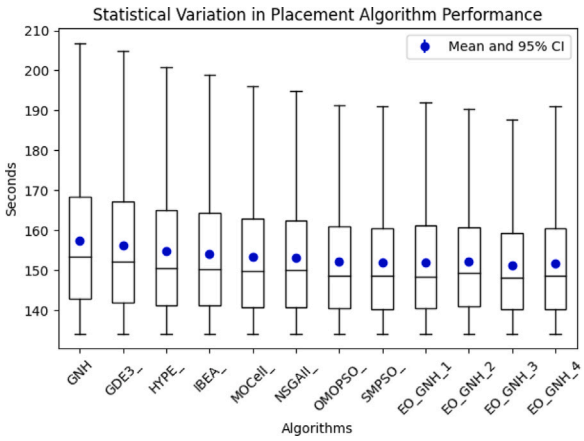


Fig. 16. For "Model Tuning": Box plot showing execution times for algorithms. Median and mean (blue dot with 95% CI bars) indicated. EO-GNH series highlights efficiency gains with added mappers.

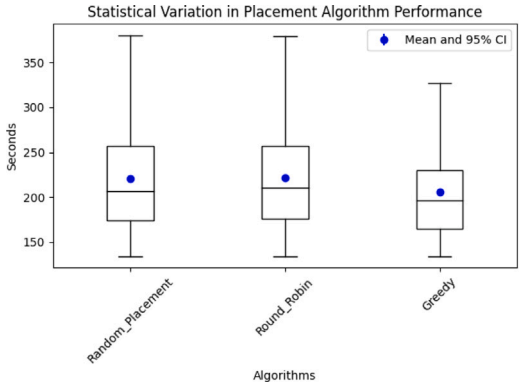


Fig. 17. For "Model Tuning": Box plot comparing execution times of Greedy, Random Placement and Round Robin algorithms. Median and mean (with 95% CI bars) are highlighted. Shows efficiency comparisons.

Table 5
Performance evaluation of placement algorithms in a Rural Agriculture -Tuning Model.

Algorithm	Utilized locations	Makespan	SR (%)
Greedy	1.03	213.11	58.00
GNH	4.05	159.90	96.00
EO-GNH-1	4.60	156.19	99.00
EO-GNH-2	4.70	156.31	99.00
EO-GNH-3	4.75	155.07	100.00
EO-GNH-4	4.78	154.69	100.00

The Table 5 detailing the performance of a rural agriculture setting for model tuning reveals a discernible trend towards improved efficiency with the EO-GNH series. The Greedy algorithm, despite minimal location usage, falls short in success rate (SR) and has the highest makespan. In contrast, the EO-GNH algorithms demonstrate enhanced makespan efficiency and an impressive SR, with EO-GNH-3 and EO-GNH-4 achieving a perfect 100% success rate. This progression suggests that the EO-GNH's advanced iterations successfully optimize for both makespan and reliability.

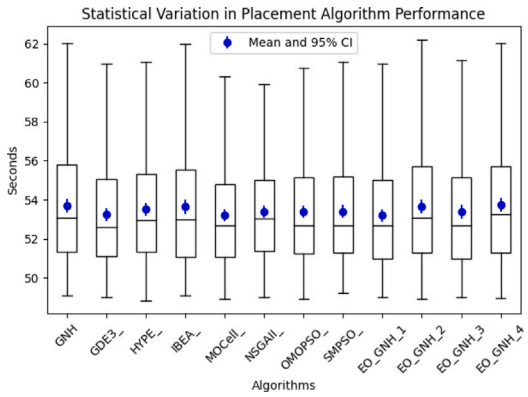


Fig. 18. For "Models Aggregation": Box plot showing execution times for algorithms. Median and mean values (blue dot with 95% CI bars) are also indicated. EO-GNH series highlights efficiency gains with added mappers.

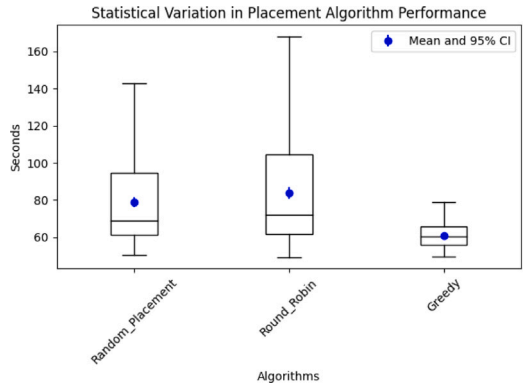


Fig. 19. For "Models Aggregation": Box plot comparing execution times of Random Placement and Round Robin algorithms. Median and mean (with 95% CI bars) are highlighted. Shows efficiency comparisons.

9.1.2. Models aggregation

The Fig. 18 reflects the tightly clustered performance metrics of several optimization algorithms, each demonstrating efficiency in a minimization context. The GNH algorithm shows a moderate range of execution times with a minimum of around 49.09 s and a maximum of 65.64 s, maintaining an average of 53.90 s and a median close to 53.14 s. The GDE3 follows closely with a slightly narrower range and a mean of 53.56 s, indicating consistent execution times. HYPE extends the maximum slightly more but with a mean that stays competitive at 53.87 s.

The IBEA presents a wider range yet manages to keep its mean at 53.77 s with a median marginally less, suggesting balanced performance. MOCell and NSGAII report similar behaviors with tight IQRs and means just above 53.50 s, indicative of their stable nature in solving the problem. OMOPSO expands the range further but still delivers a mean of 53.68 s, while SMPSO maintains this pattern with a mean of 53.72 s.

The EO-GNH series, with its iterations from 1 to 4, displays a consistent median around 53.00 s and slightly increasing IQRs, yet it still indicates a refinement in performance with each successive version. The 95% confidence intervals for all algorithms are modest, varying from around 0.36 s to 0.38 s, which provides confidence in the stability of the mean execution times and underscores the algorithms' effectiveness in consistently achieving near-optimal solutions.

The Fig. 19 provides a detailed analysis of the performance of three distinct placement algorithms, showcasing varied execution time ranges and central tendencies. The Random Placement algorithm exhibits a wide range of execution times, with a minimum of approximately 50.43 s and a maximum of 201.10 s, resulting in a mean time of 81.08 s and a median significantly lower at 69.32 s. This discrepancy between the mean and median, coupled with a substantial IQR of 34.62 s, points to a diverse distribution of execution times.

As shown in Fig. 19, the Round Robin algorithm demonstrates an expanded range from around 49.08 s to 182.78 s, with a mean of 84.71 s and a median of 73.21 s, higher than Random Placement. The IQR of 42.92 s is the largest among the three, indicating a greater spread of execution times and a less consistent performance compared to the other algorithms.

In contrast, the Greedy algorithm presents a notably tighter performance range with a minimum and maximum between 49.33 s and 145.23 s. It achieves a mean of 63.18 and a median of 61.22 s, both markedly lower than those of the other two algorithms. The IQR of 10.43 s is significantly smaller, suggesting a more concentrated and consistent set of execution times. The 95% confidence interval (CI) of 1.35 s for the Greedy algorithm is considerably smaller than those of Random Placement and Round Robin, which have CIs of 3.01 s and 3.26 s, respectively. This indicates a higher precision in the Greedy algorithm's performance, making it a more efficient choice in solving the minimization problem at hand.

Looking at the performance metrics for global model aggregation in Rural Agriculture (Table 6) underscores a notable distinction between the subsequent algorithms. The Greedy approach, while exhibiting the lowest utilization of locations, lags in makespan performance and does not achieve full success, with an 87% rate. Conversely, GNH and all versions of EO-GNH showcase a remarkable 100% success rate, with a progressively decreasing makespan from GNH to EO-GNH-4. The consistent success rate across the GNH and EO-GNH algorithms, coupled with their competitive makespan results, highlight their efficacy and reliability in an agricultural application.

Table 6
Performance evaluation of placement algorithms in Rural Agriculture - Global Model Aggregation.

Algorithm	Utilized locations	Makespan	SR (%)
Greedy	1.11	62.95	87.00
GNH	4.11	54.35	100.00
EO-GNH-1	4.32	54.18	100.00
EO-GNH-2	4.32	53.53	100.00
EO-GNH-3	4.31	53.64	100.00
EO-GNH-4	4.31	53.91	100.00

9.2. ML based cancer diagnosis

The second application considered for evaluation in this work is a cancer detection framework using ML. In this evaluation, the datasets are statistically analyzed with different algorithms, revealing a pattern of high consistency and measurement reliability. In Fig. 20 algorithms such as GNH, GDE3, HYPE, and IBEA share characteristics of close data clustering, as evidenced by their tight IQR and very narrow 95% CI for the mean, with mean values predominantly between 147.8 s and 148.0 s. The algorithms MOCell, NSGAII, OMOPSO, and SMPSO exhibit a high level of data consistency, though their mean values are marginally lower, spanning from approximately 147.4 s to 147.7 s. Moreover, the EO-GNH algorithm with different mapper set-ups, numbering from 1 to 4, shows a gradual decline in the mean execution time every time a mapper is added to the algorithm.

The reliability of replicating the same results in these algorithms is further validated by the calculated standard error of the mean (SEM). Groups GNH, GDE3, and HYPE have SEMs approximately at 0.070 s, 0.065 s, and 0.062 s, respectively, which showcases the accuracy in their mean estimation. The lower SEMs found in groups such as SMPSO and the EO-GNH with different mapper setups, which range from 0.040 s to 0.028 s, denote an even greater precision in mean estimation. Collectively, these metrics imply that the data across all the

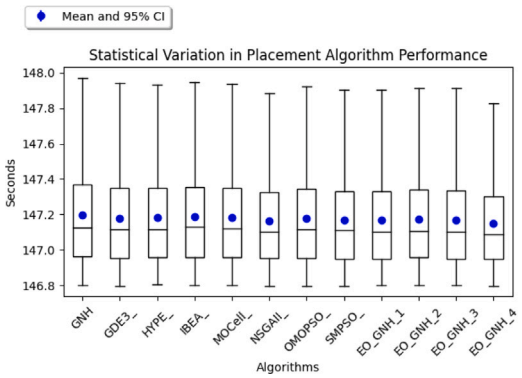


Fig. 20. For "ML-based Cancer Diagnosis": Box plot showing execution times for algorithms. Median and mean (blue dot with 95% CI bars) are also indicated in the figure.

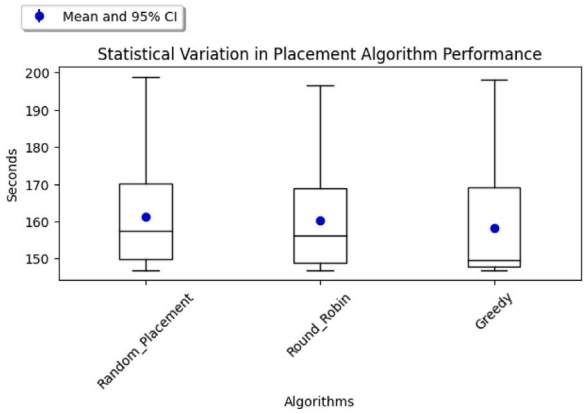


Fig. 21. For "ML-based Cancer Diagnosis": Box plot comparing execution times of Greedy Random Placement and Round Robin algorithms. Median and mean (with 95% CI bars) are also highlighted. Figure also shows the efficiency comparisons between algorithms.

algorithms is tightly centred around the mean with very little variation, highlighting the overall precision of the outcome. (see Fig. 20).

Fig. 21 captures the performance variations of three distinct placement algorithms. The Random Placement algorithm exhibits a considerable range in execution times, stretching from 146.85 s to 208.87 s, with an average of 161.35 s and a median indicating a central tendency at 157.51 s. The spread of data, as denoted by an IQR of 20.24 s, reflects a significant diversity in performance. The Round Robin algorithm, although showing a similar pattern, has a slightly tighter performance range with a minimum and maximum between 146.84 s and 196.47 s. It achieves a mean of 160.26 s and a median of 156.18 s, with an IQR of 19.93 s, suggesting a marginally more consistent performance than Random Placement. The Greedy algorithm, while having a comparable maximum value to Random Placement, has a lower mean of 158.31 s and a notably lower median of 149.59 s, accompanied by the largest IQR of 21.56 s, indicating wider variability in its results. The 95% confidence intervals are relatively narrow for all, with Random Placement at 0.47 s, Round Robin at 0.45 s, and Greedy at 0.50 s, pointing to a high precision in the mean estimates across samples for these algorithms.

The performance data captured in Table 7 for a 100 Raspberry Pi setup presents that the Greedy algorithm, while modest in location usage, shows a considerably higher makespan and a success rate that does not reach half the benchmark. In contrast, the GNH algorithm improves the makespan significantly while nearly reaching high success rate. The EO-GNH algorithms, particularly EO-GNH-3 and EO-GNH-4, achieve a 100% success rate, affirming their robustness with a marginally better makespan compared to their predecessors. The data

Table 7
Performance evaluation of placement algorithms for 100 RPi setup in Cancer Classification.

Algorithm	Utilized locations	Makespan	SR (%)
Greedy	1.97	159.57	42.00
GNH	6.52	148.56	93.00
EO-GNH-1	6.38	147.27	99.00
EO-GNH-2	6.39	147.26	99.00
EO-GNH-3	6.36	147.16	100.00
EO-GNH-4	6.35	147.15	100.00

Table 8
Performance evaluation of placement algorithms for 1000 RPi Setup in Cancer Classification.

Algorithm	Utilized locations	Makespan	SR (%)
Greedy	1.94	159.01	37.00
GNH	7.32	147.69	93.00
EO-GNH-1	7.44	147.17	100.00
EO-GNH-2	7.44	147.14	100.00
EO-GNH-3	7.44	147.17	100.00
EO-GNH-4	7.44	147.10	100.00

represented in Table 8 shows the result for the 1000 Raspberry Pi setup. The Greedy algorithm uses the fewest locations but has the lowest success rate and the highest makespan. In stark contrast, the GNH algorithm significantly improves the success rate to 93%, while the EO-GNH series achieves a 100% success rate across its variants. The makespan for EO-GNH versions shows a very minor but consistent decrease, suggesting incremental optimizations. These result implies that the EO-GNH algorithms are highly effective, maintaining an excellent balance between resource usage and successful classification, which is critical in medical applications that rely on accurate and timely data processing.

9.3. RNN based energy forecasting

The third application considered for evaluating the proposed EO-GNH framework is a RNN-based refrigeration cooling system. The analysis of execution times for various placement algorithms, as illustrated in Fig. 22, shows a notable uniformity in their performance. This is evidenced by patterns indicating tight clustering of data and high precision in measurements. The Greedy algorithm exhibits a broader spectrum of execution times, indicated by its IQR being approximately 1.095 s and a 95% CI pointing to a moderate level of variability, with average execution times around 3.766 s. On the other hand, algorithms such as GNH, GDE3, HYPE, IBEA, MOCell, NSGAII, OMOPSO, and SMPSO demonstrate more narrow IQRs, suggesting a more compact distribution of their execution times, with average values generally around 2.00 s. This pattern reflects a stable performance across these algorithms, underscored by their low Standard Error of the Mean (SEM) values, ranging approximately from 0.015 s to 0.020 s, indicating a high precision in their average execution times.

Upon evaluating the EO-GNH algorithm with varying numbers of mappers, there is an observed gradual reduction in average execution time with the inclusion of more mappers, hinting at enhanced efficiency. This trend is depicted across four different EO-GNH configurations, with mean execution times slightly declining from EO-GNH-1 to EO-GNH-4. All configurations maintain average times under 2.0 s, exhibiting close 95% CIs and small Standard Error of the Means (SEMs). These findings underscore the algorithm's consistent ability to attain near-optimal solutions, emphasizing its effectiveness in addressing the minimization challenge presented.

Fig. 23 reveals that the Random Placement algorithm displays a broad spectrum of execution times, ranging from 1.84 s to 37.75 s, with an average time of around 12.14 s and a median slightly lower at 11.74 s. This range, evidenced by an IQR of 8.56 s, indicates a

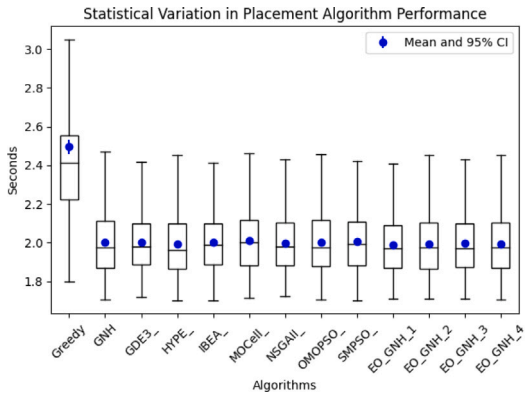


Fig. 22. For "Energy Forecasting": Box plot showing execution times for algorithms. Median and mean (blue dot with 95% CI bars) indicated. EO-GNH series highlights efficiency gains with added mappers.

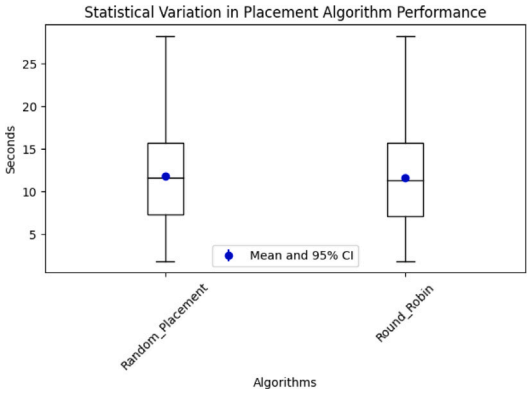


Fig. 23. For "Energy Forecasting": Box plot comparing execution times of Random Placement and Round Robin algorithms. Median and mean (with 95% CI bars) are also highlighted. Figure also shows efficiency comparisons between algorithms.

Table 9
Performance comparison of placement algorithms in 100 RPis setup (Smart Cooling System).

Algorithm	Utilized locations	Makespan	SR (%)
Greedy	3.34	1.37	73.00
GNH	5.10	2.02	98.00
EO-GNH-1	4.53	2.00	100.00
EO-GNH-2	4.48	2.00	100.00
EO-GNH-3	4.44	1.99	100.00
EO-GNH-4	4.42	1.99	100.00

wide variability in its performance. In comparison, the Round Robin algorithm, while also showing a diverse range of execution times, has a slightly better average of around 11.78 s and a median of 11.38 s. Its IQR is marginally narrower at 8.72 s, coupled with a 95% CI of 0.38 s, suggesting a more consistent grouping of sample means than the Random Placement. This indicates towards a more stable yet still varied performance profile.

Analyzing the performance across two different RPi setups for a Energy Forecasting System, the EO-GNH algorithm demonstrated a consistent trend of efficiency and effectiveness. When comparing the algorithms in a 100 RPi environment (Table 9), we observed that EO-GNH variants (1 through 4) achieved a perfect SR of 100% with a closely packed makespan around the 4.5 s mark, indicating a high level of reliability and performance. On the other hand, the Greedy algorithm, while using fewer locations, lagged behind in SR and had a shorter makespan, suggesting less effective resource allocation. Expanding the infrastructure to 1000 RPis (Table 10) showed a similar pattern.

Table 10
Performance comparison of placement algorithms in a 1000 RPis setup (Smart Cooling System).

Algorithm	Utilized locations	Makespan	SR (%)
Greedy	3.55	1.48	71.00
GNH	5.27	2.01	99.00
EO-GNH-1	4.26	2.00	100.00
EO-GNH-2	4.24	2.00	100.00
EO-GNH-3	4.24	1.99	100.00
EO-GNH-4	4.23	1.99	100.00

EO-GNH's performance remained stable in terms of success rate, achieving a full score across its variants, and its makespan slightly improved as well. Interestingly, both GNH and Greedy showed marginal variations in their makespan when scaled up, with Greedy still trailing slightly in success rate. These tables suggest that EO-GNH is robust across different scales, maintaining high success rates without significant increases in makespan, which is indicative of an algorithm well-suited for scalable systems.

The consistent performance of EO-GNH across the three application scenarios describes its potential as a reliable choice for distributed edge-cloud infrastructures. In Table 6, the reduction in the number of utilized locations has a direct effect on time efficiency. In this specific scenario, the scalarization method gives priority to cost considerations over the makespan, which can be seen as a minor overall enhancement for the cost but significantly impacted the makespan. Tables 7 to 9 demonstrate that merely increasing the number of mappers does not always lead to improved performance. This is attributed to the fact that when a solution has already achieved its optimal state, further optimization becomes redundant. Therefore, for future improvements, it is imperative that the framework integrates a mechanism to ascertain the optimal number of available mappers. This strategy is vital to avoid the unnecessary usage of resources, ensuring that the resources are utilized in the most efficient manner possible.

10. Conclusion

The paper describes Enhanced Optimized-Greedy Nominator Heuristic (EO-GNH), a sophisticated framework specifically designed for optimizing ML and AI application placement within edge computing environments. The detailed evaluation across diverse setups – ranging from smart agriculture to healthcare – has demonstrated EO-GNH's proficiency in improving the Quality of Service (QoS) parameters significantly. The adaptability of EO-GNH, coupled with its advanced hierarchical meta-heuristic design, is inherently independent of specific problems and addresses the issue of slow convergence by concurrently investigating multiple approximations of the Pareto Front. This capability extends the scope of EO-GNH way beyond the domain of IoT task allocation/placement. The approach can be adeptly applied for selecting features and fine-tuning parameters within the domain of ML and AI.

The EO-GNH framework represents a significant advancement in the field of edge computing and IoT. Its capacity to adapt to varying scales, maintain high success rates, and improve computational efficiency positions it as a powerful tool for future developments in designing intelligent systems. We plan to explore the potential of reinforcement learning, temporal difference learning, and the Markov decision process in the future work. Implementing these methods as optimization algorithms could significantly enhance the system's adaptability and performance. In future, we also plan to design safe and secure methods for sharing computational resources at the edge/fog layers. This will require the development of a robust mechanism for establishing a trust chain in the infrastructure.

CRediT authorship contribution statement

Osama Almurshed: Writing – original draft, Visualization, Validation, Resources, Project administration, Methodology, Data curation, Conceptualization. **Ashish Kaushal:** Writing – review & editing, Resources. **Souham Meshoul:** Supervision, Methodology. **Asmail Muftah:** Resources, Data curation. **Osama Almoghamis:** Writing – review & editing. **Ioan Petri:** Writing – review & editing, Resources. **Nitin Auluck:** Writing – review & editing, Supervision. **Omer Rana:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] O. Almurshed, P. Patros, V. Huang, M. Mayo, M. Ooi, R. Chard, K. Chard, O. Rana, H. Nagra, M. Baughman, et al., Adaptive edge-cloud environments for rural AI, in: 2022 IEEE International Conference on Services Computing, SCC, IEEE, 2022, pp. 74–83.

[2] F. Computing, The internet of things: Extend the cloud to where the things are, 2015, Cisco White Paper.

[3] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee, O. Rana, The internet of things, fog and cloud continuum: Integration and challenges, Internet Things 3 (2018) 134–155.

[4] O. Almurshed, O. Rana, K. Chard, Greedy Nominator Heuristic: Virtual function placement on fog resources, Concurr. Comput.: Pract. Exper. (2021).

[5] F. Hwang, S.-J. Chen, C.-L. Hwang, Fuzzy Multiple Attribute Decision Making: Methods and Applications, Springer Berlin/Heidelberg, 1992.

[6] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.

[7] T.A. Feo, M.G. Resende, Greedy randomized adaptive search procedures, J. Global Optim. 6 (2) (1995) 109–133.

[8] S. Radouche, C. Leghris, Network selection based on cosine similarity and combination of subjective and objective weighting, in: 2020 International Conference on Intelligent Systems and Computer Vision, ISCV, IEEE, 2020, pp. 1–7.

[9] J.K. Samriya, N. Kumar, An optimal SLA based task scheduling aid of hybrid fuzzy TOPSIS-pso algorithm in cloud environment, Mater. Today Proc. (2020).

[10] Y. Babuji, A. Woodard, Z. Li, D.S. Katz, B. Clifford, R. Kumar, L. Lacinski, R. Chard, J.M. Wozniak, I. Foster, et al., Parsl: Pervasive parallel programming in python, in: Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing, 2019, pp. 25–36.

[11] O. Almurshed, O. Rana, Y. Li, R. Ranjan, D.N. Jha, P. Patel, P.P. Jayaraman, S. Dustdar, A fault tolerant workflow composition and deployment automation IoT framework in a multi cloud edge environment, IEEE Internet Comput. (2021).

[12] A. Benitez-Hidalgo, A.J. Nebro, J. Garcia-Nieto, I. Oregi, J. Del Ser, jMetalPy: A Python framework for multi-objective optimization with metaheuristics, Swarm Evol. Comput. 51 (2019) 100598.

[13] M. Zeleny, Compromise programming, in: Multiple Criteria Decision Making, 1973.

[14] J.L. Herrera, J. Berrocal, S. Forti, A. Brogi, J.M. Murillo, Continuous QoS-aware adaptation of Cloud-IoT application placements, Computing 105 (9) (2023) 2037–2059.

[15] Á.M. Aparicio-Morales, J.L. Herrera, E. Moguel, J. Berrocal, J. Garcia-Alonso, J.M. Murillo, Minimizing deployment cost of hybrid applications, in: 2023 IEEE International Conference on Quantum Computing and Engineering, Vol. 2, QCE, IEEE, 2023, pp. 191–194.

[16] J. Pearl, Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley Pub. Co., Inc., Reading, MA, 1984.

[17] Y.F. Yiu, J. Du, R. Mahapatra, Evolutionary heuristic a* search: Heuristic function optimization via genetic algorithm, in: 2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering, AIKE, IEEE, 2018, pp. 25–32.

[18] Y.F. Yiu, R. Mahapatra, Hierarchical evolutionary heuristic a* search, in: 2020 IEEE International Conference on Humanized Computing and Communication with Artificial Intelligence, HCCAI, IEEE, 2020, pp. 33–40.

[19] M.I. Jordan, T.M. Mitchell, Machine learning: Trends, perspectives, and prospects, Science 349 (6245) (2015) 255–260.

[20] E. Alba, Parallel Metaheuristics: A New Class of Algorithms, John Wiley & Sons, 2005.

[21] Dask Development Team, Dask: Library for Dynamic Task Scheduling, Desk Development Team, 2016.

[22] Apache Software Foundation, Apache OpenWhisk, 2020, URL https://openwhisk.apache.org/downloads.html. (Accessed 08 December 2023).

[23] A. Alabbas, A. Kaushal, O. Almurshed, O. Rana, N. Auluck, C. Perera, Performance analysis of apache openwhisk across the edge-cloud continuum, in: 2023 IEEE 16th International Conference on Cloud Computing, CLOUD, IEEE, 2023, pp. 401–407.

[24] M. Baughman, N. Hudson, R. Chard, A. Bauer, I. Foster, K. Chard, Tournament-based pretraining to accelerate federated learning, in: Proceedings of the SC'23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis, 2023, pp. 109–115.

[25] P. Patros, M. Ooi, V. Huang, M. Mayo, C. Anderson, S. Burroughs, M. Baughman, O. Almurshed, O. Rana, R. Chard, et al., Rural ai: Serverless-powered federated learning for remote applications, IEEE Internet Comput. 27 (2) (2022) 28–34.

[26] A. Kaushal, O. Almurshed, A. Alabbas, N. Auluck, O. Rana, An edge-cloud infrastructure for weed detection in precision agriculture, in: IEEE International Conference on Pervasive Intelligence and Computing, IEEE, 2023, pp. 0269–0276.

[27] V. Mohammadi, A.M. Rahmani, A. Darwesh, A. Sahafi, Fault tolerance in fog-based social internet of things, Knowl.-Based Syst. 265 (2023) 110376.

[28] S. Amjad, A. Akhtar, M. Ali, A. Afzal, B. Shafiq, J. Vaidya, S. Shamail, O. Rana, Orchestration and management of adaptive IoT-centric distributed applications, IEEE Internet Things J. (2023).

[29] T.T. George, A.K. Tyagi, Reliable edge computing architectures for crowdsensing applications, in: 2022 International Conference on Computer Communication and Informatics, ICCCI, IEEE, 2022, pp. 1–6.

[30] G. van Rossum, time - Time access and conversions, 2023, URL https://docs.python.org/3/library/time.html.

[31] G. van Rossum, timeit, URL https://docs.python.org/3/library/timeit.html.

[32] G. Rodia, psutil, URL https://github.com/giampaolo/psutil.

[33] Apache Software Foundation, Apache Kafka, Apache Software Foundation, 2011, URL https://kafka.apache.org/.

[34] A. Tridgell, T.S. Team, Samba: A file and print server for unix, in: USENIX Conference on File and Storage Technologies, 1992, URL https://www.samba.org/.

[35] Y.N. Babuji, K. Chard, I.T. Foster, D.S. Katz, M. Wilde, A. Woodard, J.M. Wozniak, Parsl: Scalable parallel scripting in Python, in: IWSG, 2018.

[36] A.M. Turing, Systems of logic based on ordinals, Proc. Lond. Math. Soc. Ser. 2 45 (1939) 161–228.

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[38] S. Kukkonen, J. Lampinen, GDE3: The third evolution step of generalized differential evolution, in: 2005 IEEE Congress on Evolutionary Computation, Vol. 1, IEEE, 2005, pp. 443–450.

[39] J. Bader, K. Deb, E. Zitzler, Faster hypervolume-based search using Monte Carlo sampling, in: Multiple Criteria Decision Making for Sustainable Energy and Transportation Systems: Proceedings of the 19th International Conference on Multiple Criteria Decision Making, Auckland, New Zealand, 7th–12th January 2008, Springer, 2010, pp. 313–326.

[40] E. Zitzler, S. Künzli, Indicator-based selection in multiobjective search, in: International Conference on Parallel Problem Solving from Nature, Springer, 2004, pp. 832–842.

[41] A.J. Nebro, J.J. Durillo, F. Luna, B. Dorronsoro, E. Alba, Mocell: A cellular genetic algorithm for multiobjective optimization, Int. J. Intell. Syst. 24 (7) (2009) 726–746.

[42] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.

[43] M.R. Sierra, C.A. Coello Coello, Improving PSO-based multi-objective optimization using crowding, mutation and ϵ-dominance, in: International Conference on Evolutionary Multi-Criterion Optimization, Springer, 2005, pp. 505–519.

[44] A.J. Nebro, J.J. Durillo, J. Garcia-Nieto, C.C. Coello, F. Luna, E. Alba, SMPSO: A new PSO-based metaheuristic for multi-objective optimization, in: 2009 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making, MCDM, IEEE, 2009, pp. 66–73.

[45] F. Balducci, D. Impedovo, G. Pirlo, Machine learning applications on agricultural datasets for smart farm enhancement, Machines (2018).

[46] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, H. Yu, Federated learning, Synth. Lect. Artif. Intell. Mach. Learn. 13 (3) (2019) 1–207.

[47] B. Pang, Y. Song, C. Zhang, R. Yang, Effect of random walk methods on searching efficiency in swarm robots for area exploration, Appl. Intell. 51 (7) (2021) 5189–5199.

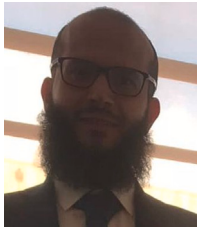
[48] A. Olsen, D.A. Konovalov, B. Philippa, P. Ridd, J.C. Wood, J. Johns, W. Banks, B. Girgenti, O. Kenny, J. Whinney, et al., DeepWeeds: A multiclass weed species image dataset for deep learning, Sci. Rep. 9 (1) (2019) 1–12.

[49] R. Cuocolo, A. Stanzione, A. Castaldo, D.R. De Lucia, M. Imbriaco, Quality control and whole-gland, zonal and lesion annotations for the PROSTATEx challenge public dataset, Eur. J. Radiol. 138 (2021) 109647.
[50] G. Litjens, O. Debats, J. Barentsz, N. Karssemeijer, H. Huisman, ProstateX Challenge data, 2017, http://dx.doi.org/10.7937/K9TCIA.2017.MURS5CL, The Cancer Imaging Archive.
[51] A.F. Molisch, Wireless Communications, vol. 34, John Wiley & Sons, 2012.
[52] E. Zitzler, K. Deb, L. Thiele, Comparison of multiobjective evolutionary algorithms: Empirical results, Evol. Comput. 8 (2) (2000) 173–195.

Osama Almurshed has a Ph.D. from Cardiff University's School of Computer Science & Informatics, United Kingdom, and is a faculty member at Prince Sattam Bin Abdulaziz University, Kingdom of Saudi Arabia. His research interests include designing and optimizing intelligent Internet of Things applications and engineering the dependability of fog-cloud computing environments.

Ashish Kaushal is a Ph.D. scholar at Indian Institute of Technology Ropar, India. He completed his B.Tech + M.Tech in a 5 year Dual Degree program from National Institute of Technology Hamirpur, India. He was a Commonwealth Split-Site Scholar 2021 at Cardiff University, U.K. His research interests include edge-cloud computing, distributed systems, and intelligent IoT systems.

Souham Meshoul is a Professor at the College of Computer and Information Sciences, Princess Nourah Bint Abdulrahman University, Kingdom of Saudi Arabia. She also directs the Master of Science in Data Science Program. Her research spans artificial intelligence and computational methods.



Asmail Muftah is a faculty member in the Department of Computer Science at Azzaytuna University and specializes in machine learning, neural symbolic AI, image processing, and medical imaging analysis. His research primarily focuses on enhancing image analysis through machine learning and the integration of neural networks and symbolic AI, with a significant focus on medical applications.

Osama Almoghamis is a lecturer in the Department of Computer Science at King Saud University, Saudi Arabia. His research focuses on IoT security, with specific interests in applied cryptography, system design, and adaptive protection systems within fog-cloud computing environments.

Ioan Petri is a Senior Lecturer in Smart Infrastructure Engineering at Cardiff University, United Kingdom. His research specializes in Artificial Intelligence and Edge Computing with applications in smart buildings and cities. He focuses on Blockchain and IoT integration, energy optimization systems, and Quantum Computing for sustainable urban development.

Nitin Auluck received his Ph.D. degree from the University of Cincinnati, USA in 2005. He is an Associate Professor with the Department of Computer Science & Engineering, Indian Institute of Technology Ropar, India. He was an Assistant Professor with the Department of Computer Science, Quincy University, from 2004 to 2010. His research interests include fog computing, real-time systems, and parallel and distributed systems.

Omer Rana is currently a Professor of Performance Engineering at School of Computer Science & Informatics, Cardiff University, United Kingdom. His research interests are edge-cloud platforms, intelligent systems and high performance distributed computing.