Future Generation Computer Systems 183 (2026) 108539

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Heterogeneous GNN for collective-task offloading in cloud-edge via deep Q-learning☆

Nicolas Farabegoli *, Davide Domini, Gianluca Aguzzi, Mirko Viroli

University of Bologna, Department of Computer Science and Engineering, via dell'Università 50, Cesena, Italy

ARTICLE INFO

Keywords:
Task offloading
Edge-cloud computing
Multi-agent reinforcement learning
Heterogeneous graph neural networks
Aggregate computing
Collective adaptive systems

ABSTRACT

Task offloading in edge-cloud computing systems requires determining optimal allocation of application components across heterogeneous infrastructure while balancing multiple objectives, like energy consumption, latency, or cost. This problem becomes particularly complex in large-scale deployments (e.g., smart cities, industrial IoT) where existing approaches fail to address collective phenomena, namely emergent system-wide behaviors like network congestion that arise from multi-device interactions, leading to suboptimal offloading decisions in large-scale deployments. To address these challenges this paper introduces a multi-agent learning framework for collective component offloading that decomposes applications into a directed acyclic graph of macro-components, enabling partial offloading where individual components can be selectively executed locally or migrated to edge/cloud servers. Our system model represents the infrastructure as a heterogeneous graph of application devices and infrastructure nodes, supporting decentralized offloading decisions while maintaining component interdependencies. In particular, we propose Informed Deep Hetero Graph Q-Learning (IDHGQL), which combines: (1) Heterogeneous Graph Neural Networks (HeteroGNNs) for policy representation that naturally handle diverse device types and relationships; (2) Aggregate computing to enrich device observations with collective system state information; and (3) a multi-agent Deep Q-Learning algorithm based on centralized training with decentralized execution that balances individual constraints with emergent collective phenomena. Experimental evaluation demonstrates IDHGQL's effectiveness in multi-objective optimization scenarios, successfully learning policies that balance battery consumption, latency, and infrastructure costs. In density-aware scenarios, agents learn spatially-adaptive strategies that dynamically adjust offloading decisions based on local congestion: favoring local execution in high-density areas to avoid network bottlenecks while leveraging edge/cloud resources in sparse regions. Ablation studies confirm that collective information integration is essential for learning such context-aware policies, with IDHGQL consistently outperforming static baselines across all evaluated metrics.

1. Introduction

Context. The rapid proliferation of IoT devices and resource-intensive applications has driven a significant evolution in computing architectures. IoT deployments are projected to reach 30 billion devices by 2030, generating 4.4 zettabytes of data,[1] pushing the boundaries of traditional cloud computing which struggles to meet stringent latency and throughput requirements [1]. The cloud–edge computing paradigm [2] has emerged as a solution, distributing computational workloads across cloud servers and edge devices to optimize performance and resource utilization. A critical challenge in these systems is determining optimal offloading strategies—deciding which application components execute locally versus being offloaded to edge or cloud infrastructure. Optimizing these decisions while considering individual device constraints and system-wide performance has been extensively studied [3–6], employing traditional optimization methods [7], genetic algorithms [8–10], and machine learning techniques [2,11].

Research gap. Despite significant research efforts in optimizing offloading decisions, a crucial aspect has been overlooked: the global-level *collective behavior* emerging from the interplay of perceptions and information sharing among multiple devices. The edge can be characterized

☆ This article is part of a Special issue entitled: 'Continuum' published in Future Generation Computer Systems.

* Corresponding author.
E-mail addresses: nicolas.farabegoli@unibo.it (N. Farabegoli), davide.domini@unibo.it (D. Domini), gianluca.aguzzi@unibo.it (G. Aguzzi), mirko.viroli@unibo.it (M. Viroli).

[1] https://transformainsights.com/news/global-iot-connections-294

https://doi.org/10.1016/j.future.2026.108539
Received 30 August 2025; Received in revised form 22 January 2026; Accepted 16 April 2026
Available online 19 April 2026

as a Collective Adaptive Systems (CAS) [12], where collective information is essential for informed offloading decisions, allowing devices to consider both individual constraints and the overall system state.

Effective offloading requires decomposing applications into components that can be selectively migrated across the infrastructure. Existing decomposition strategies—including component models [13,14], architecture description languages [15], and service-oriented frameworks [16]—do not consider collective behavior and often rely on centralized control. While approaches like pulverization [17] focus on device-level decomposition, they lack flexibility in partial application offloading.

Machine learning, particularly reinforcement learning, offers a promising solution for optimizing offloading decisions by learning complex relationships directly from data and adapting to dynamic conditions. Deep learning has been successfully applied to task offloading in satellite communication [18], Internet of Vehicles [19], and crowd management [20]. However, existing ML-based approaches typically assume binary offloading decisions (local vs. remote), homogeneous infrastructure, or centralized control—failing to address the partial offloading of workflow-based applications and the collective dynamics characteristic of large-scale edge-cloud systems.

Problem statement. Given a cloud–edge computing system comprising heterogeneous devices (end devices, edge servers, and cloud servers) connected through a multi-hop network, and applications decomposed into directed acyclic graphs of interdependent macro-components, the *collective component offloading problem* consists of determining, for each device at each time step, a placement decision for each application component—either local execution or offloading to an available edge/cloud server—such that an objective function balancing energy consumption, latency, and infrastructure cost is minimized. This problem is complicated by: (i) device heterogeneity in computational capacity, energy constraints, and connectivity; (ii) component interdependencies that impose execution order and data transfer constraints; (iii) emergent collective phenomena (e.g., network congestion, server overload) arising from concurrent offloading decisions of multiple devices; and (iv) the requirement for decentralized decision-making due to system scale and dynamics.

Research question. Building on the problem formulation above, this paper aims to optimize collective component offloading decisions by balancing individual device constraints with emergent collective phenomena. Specifically, we answer:

RQ: *How can we leverage deep reinforcement learning for collective component offloading that accounts for both individual device constraints (e.g., battery level, CPU utilization) and collective phenomena (e.g., network congestion, server load)?*

Contribution. To address this challenge, we propose Informed Deep Hetero Graph Q-Learning (IDHGQL), a distributed learning approach that combines HeteroGNNs with aggregate computing within a Deep Q-Learning framework for collective component offloading, building upon a macro-component system model from previous work [21]. While prior GNN-based offloading methods have combined graph representations with reinforcement learning, they typically assume binary offloading decisions, centralized control, and homogeneous graph models. In contrast, our work introduces the following key contributions:

1. *Heterogeneous graph representation:* unlike existing GNN-RL approaches that model infrastructure as homogeneous graphs, we employ HeteroGNNs that explicitly encode *typed nodes* (IoT devices, edge servers, cloud) and *typed edges* (wireless links, wired, application dependencies), capturing semantic differences that homogeneous models cannot represent.

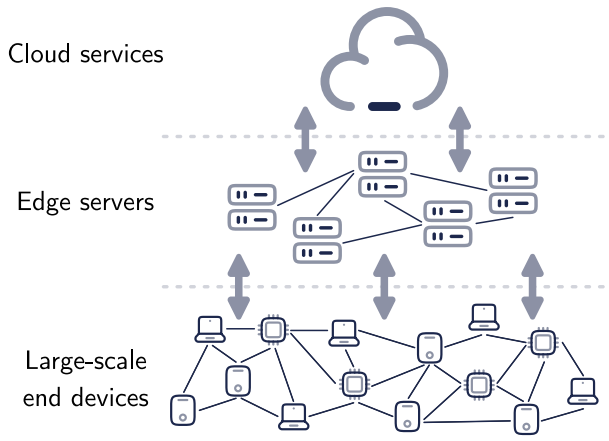


Fig. 1. Overview of edge-cloud computing system.

2. *Decentralized collective context via aggregate computing:* Rather than relying on purely local observations or centralized global state, we integrate aggregate computing [22] to synthesize collective context (e.g., neighborhood density, congestion gradients) in a fully decentralized, self-stabilizing manner—enabling agents to perceive emergent phenomena without centralized coordination.
3. *Partial workflow offloading under decentralized execution:* Unlike GNN-RL approaches that address binary offloading under centralized control, our framework supports *partial offloading of DAG-structured workflows*, thanks to our macro-component model, where individual macro-components can be selectively placed, combined with centralized training and decentralized execution (CTDE) for scalable deployment.

Experimental results demonstrate the effectiveness of IDHGQL in multi-objective optimization, with ablation studies confirming that collective information integration is crucial for learning density-aware policies.

Paper structure. The paper is organized as follows: Section 2 provides an overview of cloud–edge computing systems, as well as the challenges and importance of offloading strategies in the context of collective adaptive systems. Section 3 presents a system model and formulates the problem of collective component offloading in cloud–edge systems. Section 4 describes our proposed approach, leveraging HeteroGNNs and deep reinforcement learning for collective offloading. Section 5 presents the experimental setup and results, demonstrating the effectiveness of our approach. Finally, Section 7 concludes the paper and outlines future research directions.

2. Background and related works

2.1. Task offloading in edge-cloud systems

The edge-cloud computing paradigm has emerged as a promising solution to address the growing demand for low-latency, high-throughput applications in the era of the Internet of Things (IoT) and resource-intensive applications. This paradigm consists of a three-tier architecture (see Fig. 1): *cloud servers* offering vast computational resources but with high latency; *edge servers* providing intermediate processing capabilities closer to data sources for reduced latency [23]; and *end devices* (IoT devices, smartphones, sensors) with limited resources that generate data and require computational support [24]. The heterogeneous nature of these tiers—varying in computational capacity, latency characteristics, and resource constraints—creates the need for intelligent task allocation strategies to optimize system performance. This

operation is referred as *task offloading* [3,11,25–29] and involves deciding which tasks to move from the end device to the edge/cloud and in which order to execute these tasks. Following the literature [29], task offloading in cloud–edge systems can be characterized along several dimensions: *task type* (independent vs. workflow-based tasks), *offloading scheme* (full vs. partial task migration), *control approach* (centralized vs. distributed decision-making), *optimization objectives* (energy, cost, latency, reliability), and *communication infrastructure* (single-hop vs. multi-hop connectivity).

Within partial offloading, existing approaches differ mainly in terms of optimization objectives and solution techniques. A large body of work focuses on minimizing response time or task completion delay [30–32], relying on centralized optimization methods [30,33], heuristic algorithms [34,35], or learning-based solutions [32,36]. Other studies explicitly address cost [37–39] or energy consumption [40–42], while more recent works consider multi-objective formulations that jointly optimize latency, energy, and operational costs through mixed-integer programming or reinforcement learning [43,44]. Despite these advances, most existing solutions focus on independent tasks or assume simplified application models, and often rely on centralized or loosely coordinated decision-making. As a result, partial offloading of workflow-based applications in distributed control settings, especially over multi-hop cloud–edge infrastructures, remains comparatively underexplored.

To address this gap, our work focuses on distributed partial offloading of workflow-based tasks with multi-objective optimization across multi-hop cloud–edge infrastructures, addressing one of the least studied scenarios in the literature [29].

2.2. Edge-cloud as a collective adaptive system

Large-scale edge-cloud deployments are more than just distributed infrastructure; their behavior is best understood through the lens of CASs. CASs are complex systems of numerous autonomous, heterogeneous agents whose interactions lead to emergent, system-wide behaviors not predictable from individual actions [12]. Key characteristics of CAS include *distributed control*, necessitated by their scale and the impracticality of central coordination; *self-organization*, which enables robust adaptation to local changes without explicit global instructions; inherent *resource constraints* (e.g., energy, processing power) that influence agent decisions; and *openness*, where agents may join, leave, or fail at any time [45].

There is a growing interest in the development of applications that leverage the capabilities of cloud–edge systems to provide innovative services [46]. Particularly in very large and complex systems, such as those found in smart cities, the need to consider both individual device constraints and emergent collective behaviors is paramount [47]. The following applications motivate why a CAS-based perspective is critical for effective offloading.

2.2.1. Motivating examples

Crowd steering. In large-scale events or emergency situations, crowd steering applications guide individuals to safe zones or emergency exits, minimizing congestion and potential hazards [48]. These applications rely on real-time data from individual *smart devices* (e.g., location, movement) and collective information (e.g., crowding density). Collective offloading is critical here: in a normal operational condition, devices may offload navigation and route planning task to nearby edge servers preserving their battery life. However, in emergency situations, devices in a high-density area may experience limited bandwidth and network congestion generating communication delays—conditions that must be avoided to ensure timely and effective crowd steering. In such cases, devices can execute tasks locally to avoid network congestion and communication delays, paying the cost of higher energy consumption. This distributed approach ensures responsiveness and personalized guidance even under stress conditions.

Swarms surveillance. Consider a scenario where multiple interconnected drones or robots are employed to monitor large areas for security, disaster relief, or environmental monitoring [49]. Each device collects data (e.g., images or sensor readings) that needs to be processed and analyzed collectively to gain a comprehensive understanding of the monitored area. In these tasks, devices may face limitations in processing power, battery life, and communication bandwidth. Collective offloading enables efficient distribution of these computationally intensive tasks. For instance, image recognition tasks can be offloaded to edge servers closer to the swarm, while data fusion and analysis can be performed on more powerful cloud servers. This hierarchical approach optimizes resource utilization and enhances the overall surveillance effectiveness.

2.2.2. Macroprogramming and aggregate computing

Programming the desired collective behavior in such systems is a long-standing challenge [50]. Traditional bottom-up approaches often fail to reliably produce complex behaviors [51,52]. A more effective, top-down approach is *macroprogramming* [53], which focuses on specifying global behavior. Among macroprogramming models, we leverage *aggregate computing* [22]. It defines system-wide computations using *computational fields*—distributed data structures representing the global state—manipulated via a functional language [54]. This model is application-agnostic and has proven versatile in domains like smart cities and swarm robotics [55–57], making it an ideal foundation for modeling our offloading problem. However, applying such models effectively requires addressing the fundamental challenges inherent in collective offloading. While macroprogramming models like aggregate computing provide a powerful, top-down approach for defining the desired collective behavior, they do not prescribe how individual agents should act to best achieve these goals under dynamic, real-world constraints. An agent must still autonomously decide how to contribute to the collective computation—for instance, by choosing *where* to execute a specific task—while balancing its own resource limitations (like battery life) against fluctuating system-wide conditions (like network congestion). This challenge of continuous, adaptive decision-making in a complex and uncertain environment is precisely the type of problem that Reinforcement Learning (RL) is designed to address.

2.3. Reinforcement learning for decision making

At its core, the task offloading problem is one of sequential decision-making under uncertainty. Devices must continuously decide where to run computational tasks to optimize for certain objectives (e.g., latency, energy consumption) in a dynamic environment where network conditions and server loads can change unpredictably. RL provides a powerful mathematical framework for addressing precisely this kind of problem [11]. In this paradigm, an agent interacts with an environment (namely, something external from itself) to achieve a goal. The interaction between the agent and the environment is typically modeled as a sequence of discrete time steps. At each step the agent observes the state, selects an action, and receives a numerical reward or penalty. The agent learns a **policy** $\pi(a|s)$—a mapping from states to actions—that maximizes cumulative reward over time. Learning is often model-free: the agent improves by trial and error without a prior model of the environment's dynamics.

The standard RL problem can be formalized as a Markov Decision Process (MDP), which consists of the following components:

- A set of **states** S, which represent all possible configurations of the environment that are relevant to the decision. In the context of task offloading, a state could include the device's battery level, the current CPU load on edge servers, and network bandwidth.
- A set of **actions** A, which represents the set of choices available to the agent. For an application device, the action space could be the set of all possible offloading decisions for each of its components (e.g., execute locally, offload to edge server 1, offload to cloud).

- A **reward function** $R(s, a, s')$, which defines the immediate numerical reward an agent receives for taking action a in state s and transitioning to state s'. The reward function is designed to align with the agent's high-level goals. For example, a positive reward could be given for low latency, while a negative reward (penalty) could be associated with high energy consumption or monetary cost.
- A **transition probability function** $P(s'|s, a)$, which defines the probability of transitioning to state s' after taking action a in state s.

The goal of the agent is to find an optimal policy, π^*, that maximizes the expected cumulative discounted reward, known as the **return**. The return from a state s at time t is defined as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where $\gamma \in [0, 1]$ is the discount factor, which prioritizes immediate rewards over future ones.

To find the optimal policy, RL algorithms often learn an **action-value function**, $Q^\pi(s, a)$, which represents the expected return for taking action a in state s and subsequently following policy π.

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right]$$

The optimal policy, π^*, will have a corresponding optimal action-value function, $Q^*(s, a)$, from which the best action in any state can be found by simply choosing the action with the highest Q-value. Algorithms like Q-learning are designed to iteratively estimate this Q^* function. When the state and action spaces become very large, deep neural networks can be used to approximate the Q-function, leading to the Deep Q-Learning (DQL) approach that we leverage in this work [11].

In the context of task offloading, recent works have advanced reinforcement learning-based decision making by integrating graph representations to capture structural dependencies among tasks and infrastructure resources. For instance, Li et al. [58] apply deep Q-learning for partial offloading to multiple access points, balancing latency and energy in a distributed manner. Sun et al. [59] and Wang et al. [60] leverage GNNs to embed network structures into RL agents for improved offloading in MEC systems, but primarily address binary offloading under centralized control. Similarly, energy-aware task offloading in vehicular MEC has been investigated using distributed and federated reinforcement learning, such as federated Soft Actor-Critic frameworks, which focus on per-task local versus MEC execution under highly dynamic wireless conditions [61]. However, these approaches differ from our setting, as they do not consider workflow-level offloading or collective decision making across heterogeneous cloud–edge infrastructures. Task dependency graphs (DAGs) are considered in GNN-DRL frameworks such as Cao and Deng [62] and Wu et al. [63], which employ graph attention networks and PPO to optimize dependent subtasks on heterogeneous servers, yet rely on centralized training assumptions. Multi-agent RL approaches, such as PORA-MATD3 [64], introduce distributed offloading in dynamic vehicular scenarios, while Pamuklu et al. [65] extend hetero-GNN-RL to smart agriculture by modeling UAVs and IoT nodes under failures and topology changes. Moreover, recent works [66,67] propose DRL and game-based intelligence for IRS-aided and secure 5G/6G communications, respectively, further motivating the use of learning-based strategies in complex network optimization.

Despite these advances, existing RL-based solutions often focus on homogeneous graph representations, binary offloading decisions, or centralized learning paradigms, and rarely capture collective infrastructure dynamics. To overcome these limitations, the approach proposed in this paper integrates heterogeneous graph modeling with aggregate computing to provide collective context and adopts a multi-agent deep Q-learning framework to jointly optimize latency, energy, and cost in dynamic cloud–edge environments.

2.4. The challenge of collective offloading

The convergence of CAS and edge-cloud computing introduces significant challenges for task offloading. Existing offloading strategies, which are predominantly designed for independent tasks under centralized control [3,11,25–28], are ill-suited for the complex dynamics of CAS. Three key challenges arise in this context:

- The scale, resource constraints, and real-time needs of CAS demand **decentralized and dynamic task allocation**. Agents must make adaptive offloading decisions based on local and neighboring information, as static, predetermined schemes quickly become suboptimal. This requires lightweight, distributed algorithms that can handle partial offloading, where only a subset of tasks is moved.
- Offloading decisions cannot depend solely on an agent's individual state. They must account for **collective dynamics**, such as network congestion or aggregated server load, which are emergent properties of the system. An effective strategy must therefore capture and leverage this collective context to make informed decisions.
- Offloading in CAS is inherently a **multi-objective problem**, requiring a balance between conflicting goals like minimizing energy consumption, latency, and monetary cost. Addressing these trade-offs in a decentralized manner, while accounting for collective dynamics, presents a significant challenge.

This gap motivates our work, which addresses partial, workflow-based offloading with decentralized, multi-objective decision-making over a multi-hop infrastructure—one of the least-studied scenarios [29].

3. Problem statement

Following the challenges identified in Section 2.4, and the basic approach of system model introduced in [21], we describe the problem of collective component offloading in cloud–edge computing systems. First, we present the macro-behavior model that leverages device neighborhoods and self-organizing principles to enable distributed component deployment, supporting partial offloading of workflow-based tasks. Based on this model, we formulate the collective component offloading problem in cloud–edge systems as a multi-agent reinforcement learning task. Here, agents represent system devices that make offloading decisions by considering both local observations and collective information while optimizing multiple objectives.

3.1. System model

We introduce a system model that decomposes applications into macro-components organized as directed acyclic graphs, enabling partial offloading where components can be selectively migrated across the infrastructure. The key innovation lies in supporting both *local* and *collective* components while maintaining self-stabilizing properties [68] regardless of deployment decisions: (1) a *macroprogramming model* that structures applications as interconnected components with defined ports and bindings; (2) an *infrastructural model* distinguishing between application devices that execute the macroprogram and infrastructural devices that provide offloading resources; and (3) a *deployment model* that maps component instances to execution devices via owner–surrogate relationships.

Table 1 summarizes the notation used throughout this section.

Table 1
Symbols used to describe the system model.

Symbol	Description
MP	Application logic specification defined as a *directed acyclic graph* of components.
C	Minimal unit of computation representing a self-contained piece of possibly *collective* application logic.
$\overline{\texttt{C}}$	Set of components in the macro-program MP.
B	Specifies the connection between components via their input and output ports.
$\overline{\texttt{B}}$	Set of bindings in the macro-program MP.
p	Characterizing element of each component C where values are produced or received.
$\overline{\texttt{p}}$	Set of ports in the macro-program MP.
δ	Device available in the system.
$\mathbf{D}_P$	Set of interconnected *physical devices* defining all the available devices in the infrastructure.
$\mathcal{N}_P$	Set defining the physical neighborhood relationship between *physical devices*.
$\mathbf{D}_I$	Set of *infrastructural devices* available in the system.
$\mathbf{D}$	Set of interconnected *logical devices* defining the logical view of the system.
$\texttt{C}_i^j$	Instance of component $\texttt{C}_i$ in device δ_j.

3.1.1. Macroprogramming model

Macro-program as a DAG of components. The application logic is specified as a *directed acyclic graph (DAG)* of components, captured by a single *macroprogram* MP. The macroprogram is of the form: $(\overline{\texttt{C}}, \overline{\texttt{B}})$, where $\overline{\texttt{C}}$ is the set of *components* and $\overline{\texttt{B}}$ is the set of *bindings*. A *component* C is the fundamental unit of collective computation representing a self-contained piece of application logic. Can be seen as a function taking a (possibly empty) set of inputs and producing a single value as output.

Values produced or received by a component are exchanged through *ports* p, which are defining elements of each component. A *binding* B is of the form: component$(\overline{\texttt{p}}, \texttt{C}, \texttt{p})$, specifies that the component C has a list of input ports $\overline{\texttt{p}}$, and an output port p. When the same port is used in different bindings, multiple components are connected through that port. For instance, the binding component$(\langle \texttt{p}_0 \rangle, \texttt{C}_1, \texttt{p}_1)$ and component$(\langle \texttt{p}_1 \rangle, \texttt{C}_2, \texttt{p}_2)$ specifies that the output of $\texttt{C}_1$ is connected to the input of $\texttt{C}_2$.

A macroprogram MP is said *well-formed* if: (i) different components have different output ports; (ii) each input port of a *downstream* component is connected to the output port of some *upstream* component; (iii) the transitive closure of input-to-output connections is acyclic. Input and output ports not connected to any components are defined *global* ports of the macroprogram MP.

Collective and local components. We distinguish between *collective* and *local* components (Fig. 2). A component is said *collective* (thick border square in Fig. 2(a)) if the execution of one of its instance requires the interaction with other neighbor instances of the same component. Conversely, it is said *local* (thin border square in Fig. 2(b)) if the execution is just a transformation of local inputs to the local output.

For instance a component "*taking the temperature and return a boolean holding true if it is above a certain threshold*" is a pure local component, while a component "*taking the temperature and returning a boolean holding true only if the temperature is above a certain threshold in an area of 200 m in the last hours*" is a collective component since it requires the interaction with neighbor devices to detect a distributed situation over time.

Components interaction. Components instance interaction occurs at two levels: *inter-component interaction* and *intra-component interaction*. The former occurs when a component instance—instantiated in a device—gets inputs from an upstream components instances associated to the same device. The latter occurs when given a component instance C associated to a device, it exchanges information with the C instances associated to its neighbor devices.

In other words, the *inter-component* interaction occurs between the components instances in the same device, while the *intra-component* interaction occurs between the components instances of the same component in neighbor devices. Fig. 2 depicts the *intra-component* interaction for the collective component $\texttt{C}_2$ (depicted in Fig. 2(a)), and the *inter-component* interaction for the local component $\texttt{C}_1$ (depicted in Fig. 2(b)).

Inputs, output and state. When a component instance is executed (i.e., a *round* is performed), it yields a new state θ containing—among other things—the computation output v. The collective nature of the component can be implemented in several ways, as follows we leverage the aggregate computing approach [22] to describe the collective execution of the component. The collective behavior is implemented by making the component instance of a certain component C instantiated on a certain device executes against:

1. its state θ computed in the previous round;
2. the inputs v received from upstream components located in the same device;
3. (if any) the sensors' state of the device; and
4. the states $\overline{\theta}$ of C instances located in the neighbor devices.

Note that the fourth element (i.e., the shared state with neighbor devices) is not available for local components.

3.1.2. Infrastructural and device model

Physical system. The *physical system* consists of a network of *physical devices* $\delta \in \mathbf{D}_P$ interconnected with each other. Assuming they can exchange messages in a peer-to-peer fashion, without a central controller. The communications occur in a reflexive and symmetric way, based on *physical neighborhood* relationships $\mathcal{N}_P$, where $(\delta, \delta') \in \mathcal{N}_P$ represents that the device δ and δ' can exchange messages. The network on the left, depicted in Fig. 3 shows an example of a physical system composed of devices interconnected in a peer-to-peer fashion, determining our *physical system model*.

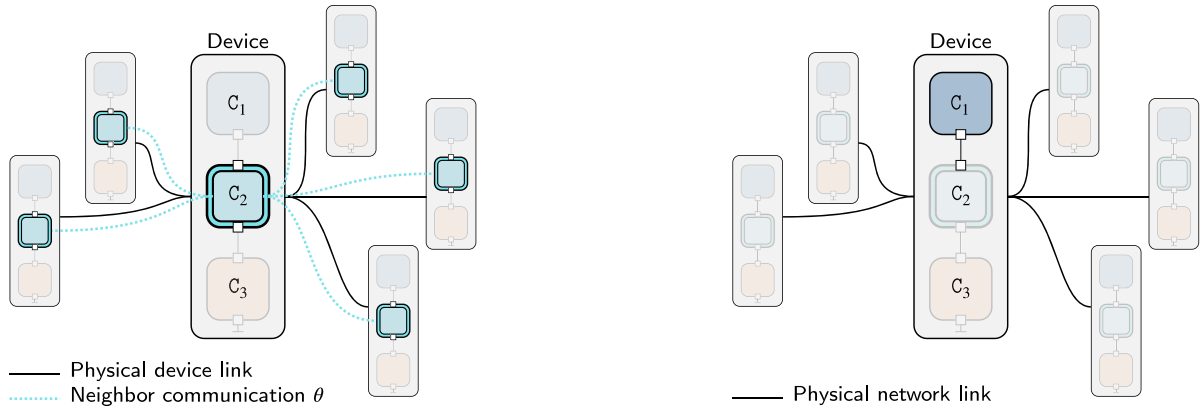
Application devices. Given a macroprogram MP, it is assumed to be collectively executed by a subset of the physical devices $\mathbf{D}$, which we call *application devices*. The network in the middle, depicted in Fig. 3, shows an example of a physical system where some devices are *application devices*. The *application devices* define a neighborhood relationship $\mathcal{N}$ which is a subset of the *physical neighborhood* relationship $\mathcal{N}_P$.

Infrastructural devices. We define additional devices that we call *infrastructural devices* $\mathbf{D}_I \subseteq \mathbf{D}_P$ of physical devices, that can host the computation on behalf of some (zero or many) application devices. The network on the right, depicted in Fig. 3, shows an example of a physical system where some devices are *infrastructural devices*. We assume $\mathbf{D}$ and $\mathbf{D}_I$ to be fixed and known in advance, so that $\mathbf{D} \cup \mathbf{D}_I = \mathbf{D}_P$, and the physical network topology $\mathcal{N}_P$ (and consequently $\mathcal{N}$) may change over time, because of node mobility.

3.1.3. Application and deployment model

Device–component relationship. Given a macroprogram MP, and a set $\mathbf{D}$ of application devices on which the macro-program will be executed, the overall system will run a component instance $\texttt{C}_j^i$ for each pair of component $\texttt{C}_i$ and device δ_j (belonging to $\mathbf{D}$). We call *owner* of a component instance $\texttt{C}_j^i$ the device δ_j, namely the device on which the component instance is intended to be executed. However, there may be situations preventing the execution of the component instance on the *owner* device, requiring to "offload" the execution of the component instance to an infrastructural device. This can be caused by the lack of proper capabilities of the device (e.g., memory, processing power), or simply because it is more convenient to execute the component elsewhere.

When a component is *offloaded* to an infrastructural device, that device is called *surrogate* since it acts as the owner of the component instance $\texttt{C}_i^j$. Different components of the same device δ may be offloaded to different infrastructural devices.



(a) Representation of a collective component requiring the interaction with other instances in neighbor devices.

(b) Representation of a local component requiring no interactions with other instances of the same component.

Fig. 2. Graphical representation of *collective* and *local* components in the proposed system model.

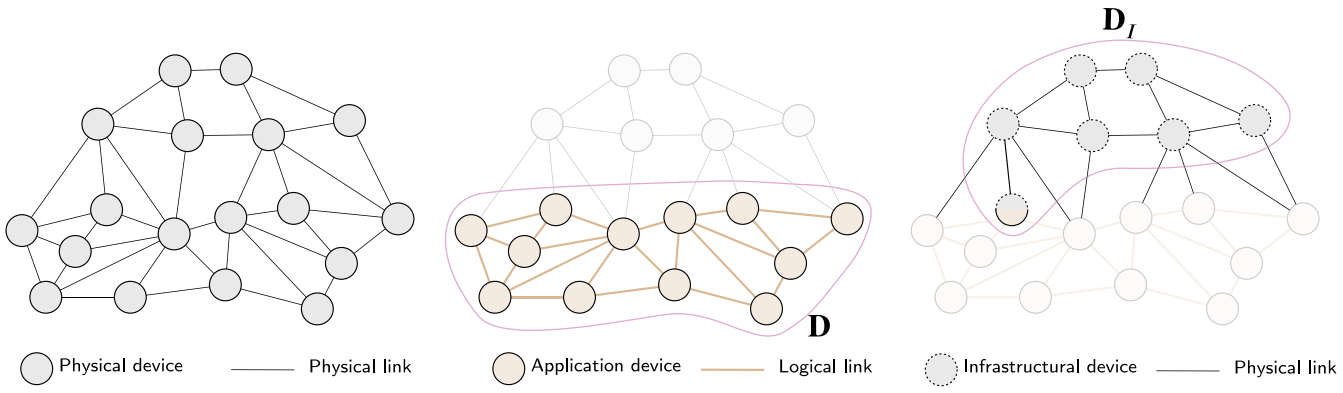


Fig. 3. Example of the macroprogramming model composed of devices interconnected in a peer-to-peer fashion (left), where some devices are *application devices* (middle), and some devices are *infrastructural devices* (right).

Offloading resilience. Given the round-based execution model of each component instance, the eventual behavior of the system is always guaranteed to be self-stabilizing, as discussed in [21]. This property holds regardless of the deployment decisions made for each component instance, meaning that no matter where each component instance is executed (either on the owner device or on a surrogate infrastructural device), the overall system will eventually converge to a correct behavior.

From an offloading perspective, this means we can transparently offload any component instance to any infrastructural device, without worrying about any constraints when we model an offloading strategy. This property greatly simplifies the offloading problem, as we do not need to consider any constraint when deciding where to offload a component instance. Nevertheless, taking into account the components dependencies and the infrastructure conditions can improve the non-functional properties of the system (e.g., latency, energy consumption, monetary cost), while functional correctness is always guaranteed by construction.

3.2. Crowd steering real world example

We consider the *Crowd Steering* application presented in Section 2.2.1, and how it can be modeled using the system model introduced in Section 3.1. Such modeling will be useful to better understand the offloading problem presented in the next sections.

For the sake of clarity, we consider a simple infrastructure composed of a set of *application devices* $\mathbf{D}$, defined by the smart devices carried by the users participating in the event, and a set of *infrastructural devices* $\mathbf{D}_I$, defined by some edge servers deployed close to the event. To keep the example simple, we consider the $\mathbf{D}$ set and the $\mathbf{D}_I$ set to be disjoint, meaning that no infrastructural device is also an application device.

Topologically, the $\mathcal{N}_P$ relationship is defined by the wireless communication capabilities of each device, e.g., Bluetooth connectivity to interconnect user devices, and Wi-Fi or 5G connectivity to interconnect user devices with edge servers. In this scenario, the $\mathcal{N}$ relationship is defined by the Bluetooth connectivity only, as a dynamic peer-to-peer network among user devices; while all the user devices are connected to the closest edge servers via Wi-Fi or 5G.

The macroprogram can be defined as the interaction between four components:

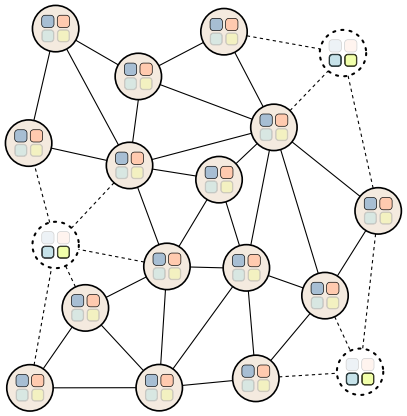
- Position sensor $\mathtt{C}_\sigma$ defining the component responsible for acquiring the device location; this component is *local* since it does not require interaction with other devices;
- Density estimator $\mathtt{C}_\rho$ defining the component responsible for computing the local density of the crowd; this component is *collective* since it requires the interaction with neighbor devices to define a distributed collective data structure;
- Route planner $\mathtt{C}_\kappa$ defining the component responsible to compute the route towards the safe zone; also this component is *collective* since it requires a global view of the system to compute the best route; and
- Navigation system $\mathtt{C}_\alpha$ defining the component responsible for interpreting the route and providing the user with the proper

guidance; this component is *local* since no neighbor interaction is required.

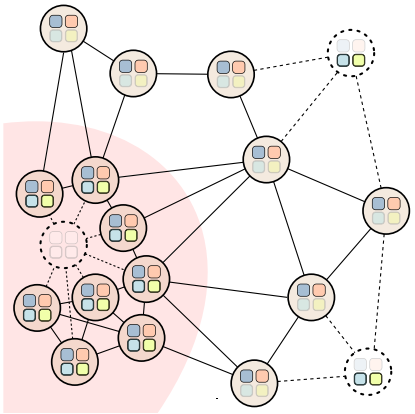
In this application, the Position sensor acquires, for each application device, the current position via a local sensor (e.g., GPS or UWB). The output of this component, expressed with the port $\mathtt{p}_\sigma$, it is then used by the Density estimator to compute the density estimation of the crowd. This component is collective since the density estimation is computed by a global perspective, requiring the continuous interaction with neighbor devices. Then, based on the information provided by the Density estimator and the Position sensor, namely the ports $\mathtt{p}_\rho$ and $\mathtt{p}_\sigma$, the Route planner computes the best route towards the safe zone producing as output the route that should be followed by the user. Finally, this output, made available through the port $\mathtt{p}_\kappa$, is taken by the Navigation system to provide the user with the proper guidance towards the safe zone.

Thanks to the partitioning of the application logic into components, the macroprogram MP can be deployed on the available devices, exploiting the infrastructural devices to offload the computation when needed. In particular, in a "steady" condition of the system, namely when no significant crowd or emergency conditions are present, the computation of the Density estimator ($\mathtt{C}_\rho$) and the Route planner ($\mathtt{C}_\kappa$) can be offloaded to the infrastructural devices $\mathbf{D}_I$ (i.e., edge servers), reducing the load on the application devices and ensuring a longer battery life. Conversely, whenever a critical situation arises, and a significant crowd density is detected, to promptly react to the emergency, all the computation is performed on the application devices $\mathbf{D}$ (i.e., user devices), ensuring a faster reaction time and a more responsive system.

The Fig. 4 shows a (simplified) deployment of this system in the two conditions: *steady* (Fig. 4(a)) and *emergency* (Fig. 4(b)). The solid circles represent the application devices $\mathbf{D}$, while the dashed circles represent the infrastructural devices $\mathbf{D}_I$. The dashed lines towards the infrastructural devices represent the Wi-Fi/5G connections, while the solid lines among application devices represent the Bluetooth connections.



(a) Deployment in steady condition with collective components offloaded to nearest infrastructural devices.



(b) Deployment in emergency condition. The soft red area represents the emergency zone.

Fig. 4. Crowd steering application deployment in two conditions. Application devices: solid circles; infrastructural devices: dashed circles.

3.3. Problem formulation

Given the system model detailed in this section, our goal is to devise a decentralized offloading strategy that optimizes system performance across several objectives (e.g., energy consumption, latency, reliability) while respecting individual device constraints and the collective dynamics of the system. This strategy defines a mapping between the components of the macroprogram MP and the devices in the system. Specifically, this mapping is a function from the set of components $\mathtt{C}$ to the set of devices $\mathbf{D}_O = \mathbf{D}_I \cup \{\delta_j\}$ where δ_j is the current device in which the policy is executed. Each *application device* may locally observe several following features like battery level of each *application device* ($\mathbf{D}$), management cost of each *infrastructural device* ($\mathbf{D}_I$), and the output of its hosted components instances. Notably, not all the information described above may be available to the *application device*, like the management costs. They are eventually gathered from the neighboring devices and the connected *infrastructural devices*. Therefore, from a static point-of-view, the overall system state is represented as a heterogeneous graph $G_t = (V_t, E_t)$ at time t as follows:

- V_t: the set of nodes, representing application devices $\mathbf{D}$ and infrastructure devices $\mathbf{D}_I$. Each node $v \in V_t$, representing a device $\delta \in \mathbf{D}_P$, has a type $\phi(v) \in \{\mathbf{D}, \mathbf{D}_I\}$.
- E_t: the set of edges, representing the communication links between devices. Each edge $e \in E_t$ has a type $\psi(e) \in \{\mathbf{D}_I - \mathbf{D}_I, \mathbf{D}_I - \mathbf{D}, \mathbf{D} - \mathbf{D}_I, \mathbf{D} - \mathbf{D}\}$.

Each application device $\delta \in \mathbf{D}$ has an associated observation $o_{\delta,t}$ which incorporates both individual and collective information; such associated observation comprises:

- $i_{\delta,t}$: the local information of device δ at time t, including features like battery level and CPU utilization.
- $\mathtt{C}_{p,\delta}$: the percentage of the components $\mathtt{C}_p$ executed locally on device δ.
- $N_{\delta,t}$: information from neighboring application devices and connected infrastructure devices computed used the information from the heterogeneous graph G_t and following the edge type device-infrastructural and device–device. This may include the battery level of neighboring devices, the cost of the infrastructural devices, and the output of the hosted component instances.
- $c_{\delta,t}$: the collective system state at time t perceived locally by δ, derived using aggregate computing, which might capture global aspects like network congestion or average server load. Formally, it is composed of the output of the macroprogram MP, namely the output of the components $\overline{\mathtt{C}}$.

Therefore, the observation for device δ at time t is defined as $o_{\delta,t} = (i_{\delta,t}, \mathtt{C}_{p,\delta}, N_{\delta,t}, c_t)$.

To elaborate the dynamics of the system, we formulate this problem within a SwarMDP framework modeling the system as a collective agent composed of individual devices. This SwarMDP is defined by the tuple $\langle \mathbb{A}, \mathcal{E} \rangle$, where $\mathbb{A}$ represents a prototypical swarming agent and $\mathcal{E}$ represents the environment. The agent $\mathbb{A}$ is characterized by:

- $\mathcal{S}$: the global state space, representing the system-wide state (i.e., the heterogeneous graph G_t).

- $\mathcal{O}$: the set of environment observations available to the agent (i.e., $o_{\delta,t}$ for each device δ).
- $\mathcal{A}$: the set of actions the agent can perform.
- $\mathcal{R} : \mathcal{O} \to \mathbb{R}$: the reward function, mapping observations to a numerical reward.
- $\pi : \mathcal{O} \to \mathcal{A}$: the agent's policy, mapping observations to actions.

The environment $\mathcal{E}$ is characterized by:

- P: the number of agents (devices) in the system.
- $\mathcal{T} : \mathcal{S}^P \times \mathcal{A}^P \to \mathcal{S}^P \times \mathbb{R}^P$: the global transition function, mapping the current joint state and joint action to a tuple containing the next joint state and a vector of per-agent rewards.
- $\Phi : \mathcal{S}^P \to \mathcal{O}^P$: the observation function, mapping the global state to individual agent observations.

The SwarMDP dynamics are governed by:

$$o_t = \Phi(s_t), \quad a_t = \pi(o_t), \quad (s_{t+1}, r_t) = \mathcal{T}(s_t, a_t)$$

At time t, agents observe $o_t = (o_{1,t}, \dots, o_{P,t})$, select actions a_t according to policy π, and the environment transitions to state s_{t+1} with rewards r_t. The optimal policy π^* maximizes the expected cumulative reward:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\ \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

where γ is the discount factor.

In our case, $P = |\mathbf{D}|$ agents comprise the system. The action space $\mathcal{A}$ encompasses all possible offloading decisions for each component instance. Formally, it maps each component instance $\mathtt{C}_j$ to a device δ_k:

$$\mathcal{A} = \{\mathtt{C}_j \mapsto \delta_k \mid \mathtt{C}_j \in \overline{\mathtt{C}}, \delta_k \in \mathbf{D}_O\}$$

The cardinality of $\mathcal{A}$ is $|\overline{\mathtt{C}}| \cdot |\mathbf{D}_O|$. The reward function $\mathcal{R}$ is application-dependent. Here, we employ a multi-objective approach, try to balance several objectives, such as energy consumption and system-wide cost. Among the possible objectives, we consider a standard linear combination of the objectives. Formally, given a set of reward functions $\mathcal{R}_i : \mathcal{O} \to \mathbb{R}$, we define the multi-objective reward function as:

$$\mathcal{R}(o) = \sum_{i=1}^{n} \lambda_i \mathcal{R}_i(o) \quad \text{where} \quad \sum_{i=1}^{n} \lambda_i = 1$$

where λ_i are the weights of the objectives.

4. Proposed approach

To address the collective component offloading problem formulated in Section 3, we introduce the **I**nformed **D**eep **H**etero **G**raph **Q**-**L**earning (IDHGQL) framework. This approach finds a decentralized offloading policy that optimizes multiple objectives by leveraging the structural and collective properties of the edge-cloud environment. The framework, depicted in Fig. 5, integrates: a HeteroGNN to represent agent policies and process heterogeneous graph structures; aggregate computing to enrich observations with collective state information ($c_{\delta,t}$); and a multi-agent DQL algorithm with centralized graph replay buffer for stable training. The following subsections detail each of these components.

4.1. HeteroGNN for Q-function approximation

We employ a HeteroGNN as our deep Q-network to approximate the action-value function Q in the SwarMDP formulation. This choice is motivated by three key advantages:

- **Structural Awareness:** the system state is naturally represented as a heterogeneous graph G_t. A HeteroGNN can directly operate on this graph, capturing the complex topological dependencies between application devices, edge servers, and cloud servers.
- **Handling Heterogeneity:** the HeteroGNN architecture uses type-specific transformations for different node types (e.g., $\mathbf{D}$, $\mathbf{D}_I$) and edge types (e.g., $\mathbf{D}-\mathbf{D}$, $\mathbf{D}-\mathbf{D}_I$). This allows the model to learn distinct representations and interaction patterns for each kind of entity in the system, which is essential for making nuanced offloading decisions.
- **Decentralized Execution:** although trained with access to the global graph, a trained HeteroGNN can be executed in a decentralized manner. Each device computes its own embedding (and subsequent Q-values) by passing messages only with its direct neighbors. This aligns perfectly with the operational constraints of a distributed edge-cloud system.

The HeteroGNN takes the feature-augmented graph G_t as input and produces Q-values for all possible offloading actions for each application device. The action with the highest Q-value is then selected by the policy. More details on the HeteroGNN architecture and message-passing formulation follow.

4.1.1. HeteroGNN architecture and message-passing formulation

We now formally describe how the heterogeneous graph structure is processed to compute Q-values for offloading decisions (See Fig. 6).

Handling heterogeneous feature spaces. Application devices and infrastructural devices have different feature spaces (e.g., $\mathbf{x}_v^{\mathbf{D}} \in \mathbb{R}^{d_{\mathbf{D}}}$ including battery level, local CPU usage, and collective density; $\mathbf{x}_u^{\mathbf{D}_I} \in \mathbb{R}^{d_{\mathbf{D}_I}}$ including cost, available capacity, and latency parameters). To handle this heterogeneity, we adopt a practical approach: we define a homogeneous Graph Attention Network (GAT) [69] and automatically convert it into a heterogeneous architecture. This conversion *replicates* the message-passing operators for each edge type, creating separate weight matrices that learn type-specific transformations. Crucially, while input dimensions vary across node types and are inferred lazily at runtime, all replicated operators project their outputs into a *shared hidden space* of dimension h. This common embedding space enables the aggregation of messages from different edge types and ensures that node representations remain compatible across the heterogeneous graph structure.

Edge-type-specific message passing. Following the heterogeneous message-passing paradigm [70], the replicated GAT layers compute messages differently for each edge type $\tau \in \Psi$, where $\Psi = \{\psi(e) \mid e \in E_t\}$ denotes the set of all edge types in the graph, while operating within the shared h-dimensional embedding space. For a node v with neighbors $\mathcal{N}_v^\tau$ connected via edge type τ, the message from neighbor u to v is computed as:

$$\mathbf{m}_{u\to v}^{\tau} = \alpha_{uv}^{\tau} \cdot \left(\mathbf{W}^{\tau} \cdot \mathbf{h}_u^{(\ell)}\right) \tag{1}$$

where $\mathbf{W}^{\tau} \in \mathbb{R}^{h\times h}$ is an edge-type-specific weight matrix, and α_{uv}^{τ} is a learned attention coefficient that weighs the importance of each neighbor. The attention mechanism considers both node embeddings and edge features $\mathbf{e}_{uv}$ (e.g., communication latency), allowing the model to learn how structural properties influence message importance during aggregation.

Heterogeneous neighborhood aggregation. For each node v, messages from all edge types are aggregated to form the updated node embedding:

$$\mathbf{h}_v^{(\ell+1)} = \sigma\left(\sum_{\tau\in\Psi}\sum_{u\in\mathcal{N}_v^{\tau}} \mathbf{m}_{u\to v}^{\tau}\right) \tag{2}$$

where σ is a non-linear activation function. This aggregation scheme replicates the message-passing operations for each edge type and combines incoming messages. Consequently, the embedding of an application device node incorporates: (i) information from neighboring application devices (via $\mathbf{D}$-$\mathbf{D}$ edges), capturing local collective phenomena; and (ii) information from connected infrastructural devices (via $\mathbf{D}$-$\mathbf{D}_I$ edges), capturing offloading target characteristics such as cost and capacity.

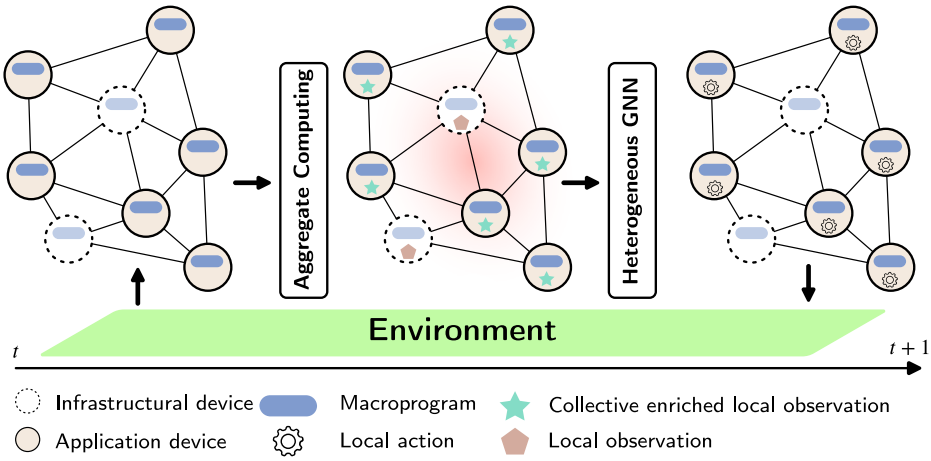


Fig. 5. An overview of the proposed IDHGQL framework. The system state is modeled as a heterogeneous graph, which is enriched with a collective perception computed via aggregate computing. This informed graph representation serves as the input to a HeteroGNN-based reinforcement learning agent, which learns a decentralized policy to determine the optimal offloading strategy for each component instance.

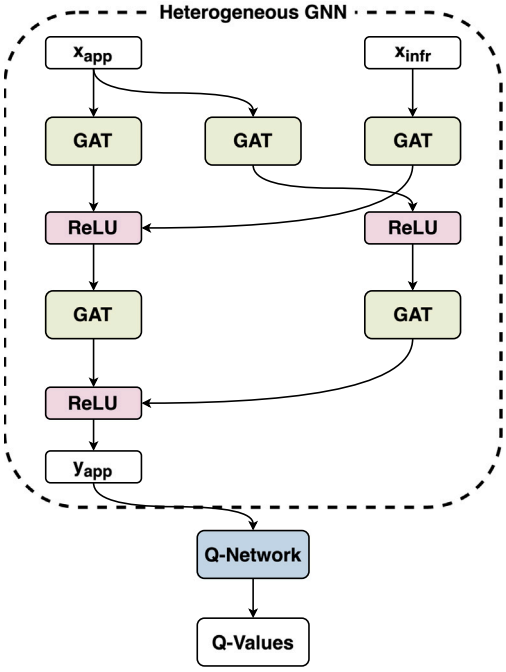


Fig. 6. Conceptual HeteroGNN architecture for Q-function approximation in our offloading problem.

Q-value computation. After L message-passing layers, the final node embeddings $\mathbf{h}_v^{(L)}$ encode both local features and neighborhood context. For application device nodes $v \in \mathbf{D}$, we compute Q-values for all possible offloading actions through a multi-layer perceptron (MLP) with M layers (allowing $M \geq 2$):

$$\mathbf{z}_v^{(0)} = \mathbf{h}_v^{(L)},$$
$$\mathbf{z}_v^{(l)} = \sigma^{(l)}\left(\mathbf{W}^{(l)}\mathbf{z}_v^{(l-1)} + \mathbf{b}^{(l)}\right), \quad l = 1, \ldots, M-1,$$
$$\mathbf{Q}_v = \mathbf{W}^{(M)}\mathbf{z}_v^{(M-1)} + \mathbf{b}^{(M)} \in \mathbb{R}^{|\mathcal{A}|},$$

where $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weight matrix and bias of layer l, and $\sigma^{(l)}$ is the (optional) non-linear activation for layer l (e.g., ReLU). The optimal action for device v is selected as $a_v^* = \arg\max_a \mathbf{Q}_v[a]$.

Critically, Q-values are computed only for application device nodes, as these are the agents making offloading decisions. Infrastructural device nodes serve as information sources during message passing, providing features about offloading targets that influence the Q-value computation for application devices.

4.2. Collective information integration via aggregate computing

A key challenge in CASs is enabling individual agents to act on collective intelligence. Relying solely on the raw graph structure for this can be inefficient, as information must propagate through many message-passing layers. To address this, we "inform" the learning process by directly embedding a high-level summary of the collective state into each agent's observation.

We achieve this using aggregate computing. Before each decision step, an aggregate program runs across the devices. This program computes a distributed data structure—a computational field—that represents a specific collective property. For instance, in the crowd steering scenario, an aggregate program computes the collective device density, which serves as a proxy for potential network congestion. From an operational perspective, the aggregate computation executes in *asynchronous rounds* (following our system model in Section 3.1): each device periodically (i) gathers the most recent messages from its neighbors; (ii) runs the aggregate program locally using this information and its own state; and (iii) broadcasts the resulting value to its neighbors. For a comprehensive formal treatment of aggregate computing and its operational semantics, we refer the interested reader to [54]. This round frequency is a tunable parameter—it can match the learning decision rate, or it can run at a higher frequency to provide fresher collective estimates before each offloading decision. Regarding communication delays, aggregate computing is inherently *self-stabilizing* [54]: even when messages arrive late or are temporarily lost, the computation continuously re-executes and converges toward the correct global result, so transient inconsistencies are naturally absorbed without requiring explicit synchronization barriers. Finally, the per-round computational cost is $O(|N_\delta|)$—linear in the neighborhood size—since each device only processes data from its direct neighbors, making this approach lightweight and practical for resource-constrained edge devices.

The output of this aggregate program, $c_{\delta,t}$, is then appended to the feature vector of the corresponding device δ in the graph G_t. This enriched observation, $o_{\delta,t} = (i_{\delta,t}, \mathbb{C}_{p,\delta}, N_{\delta,t}, c_{\delta,t})$, provides the HeteroGNN with immediate access to both low-level local state and high-level collective context.

Compared to relying solely on deep message passing to capture collective phenomena, aggregate computing offers distinct advantages. Capturing global properties like device density through GNNs alone

would require $O(D)$ message-passing layers (where D is the network diameter), whereas aggregate programs achieve this through self-stabilizing asynchronous rounds with guaranteed convergence even under message delays. Moreover, aggregate programs provide interpretable, domain-specific features (e.g., congestion gradients, resource hotspots) that complement the HeteroGNN's learned representations, allowing the model to focus on learning optimal decision policies—such as executing locally in high-density areas to avoid latency, even when battery levels would otherwise favor offloading—rather than discovering collective properties from scratch. This integration accelerates learning and leads to more robust, context-aware offloading strategies.

4.3. The Informed Deep Hetero Graph Q-Learning (IDHGQL) algorithm

The IDHGQL algorithm integrates the HeteroGNN policy network and the aggregate computing state enrichment into a coherent learning procedure. It adapts the core principles of DQL, including experience replay and the use of a target network, to our multi-agent, graph-based problem.

The algorithm follows a centralized training and decentralized execution pattern. During training, the algorithm has access to the global graph state and can sample transitions from the centralized replay buffer to learn optimal policies. However, during execution, each device makes offloading decisions independently using only its local observation and neighborhood information, enabling fully decentralized operation without requiring a central coordinator.

The key components of the algorithm, detailed in Algorithm 1, are as follows:

1. **Initialization:** the algorithm begins by initializing the Q-network θ and the target network θ^- with identical weights. A graph replay buffer $\mathcal{D}$ is also initialized. This buffer is designed to store entire graph transitions (G_o, a_t, G_r, G'_o), where G_o is the observed graph state, a_t is the joint action, G_r is the graph of rewards, and G'_o is the next graph state.
2. **Action Selection:** at each timestep, the current graph state G is first enriched with the collective state c_t computed by the aggregate function Agg (Line 3). Then, for each device, an action is selected using an ϵ-greedy strategy (Lines 4–11). With probability ϵ, a random action is chosen for exploration; otherwise, the action with the maximum Q-value, as predicted by the current HeteroGNN network θ, is selected.
3. **Experience Storage:** the joint action a_t is executed in the environment, yielding a reward for each device and the next state graph G'_o. This complete transition tuple is stored in the graph replay buffer $\mathcal{D}$ (Line 14).
4. **Model Training:** a mini-batch of transitions is sampled from $\mathcal{D}$ (Line 15). For each transition in the batch, the target Q-value, y_δ, is calculated using the reward and the maximum Q-value for the next state, as estimated by the stable target network θ^- (Line 17). The loss is then computed as the mean squared error between this target value and the Q-value predicted by the main network θ (Line 18). The main network's weights are updated via gradient descent to minimize this loss (Line 20).
5. **Target Network Update:** periodically, the weights of the target network θ^- are updated with the weights from the main network θ to maintain training stability (Line 21).

Algorithm 1 Informed Deep Hetero Graph Q-Learning (IDHGQL)

Require: Environment E, graph replay buffer $\mathcal{D}$, target network θ^-, current network θ, exploration rate ϵ, aggregate function Agg
Ensure: Trained IDHGQL model θ

1: Initialize $\mathcal{D}$ with random transitions, θ with random weights, $\theta^- \leftarrow \theta$
2: **while** not done **do**
3: Observe current graph state G_o and compute collective enrichment $c_t = Agg(G_o)$
4: **for** each device $\delta \in \mathbf{D}$ **do**
5: Obtain device state $o_{\delta,t}$
6: **if** random() $< \epsilon$ **then**
7: Select random action $a_{\delta,t}$
8: **else**
9: Compute Q-values: $G_q = Q(G_o; \theta)$
10: Select action: $a_{\delta,t} = \arg\max_a G_q[\delta](a)$
11: **end if**
12: **end for**
13: Execute joint action set $a_t = \{a_{\delta,t} \mid \delta \in \mathbf{D}\}$ in E
14: Observe per-device rewards (reward graph) G_r and next graph state G'_o
15: Store transition (G_o, a_t, G_r, G'_o) in $\mathcal{D}$
16: Sample mini-batch $\{(G^b, a^b_t, G^b_r, G'^b)\}$ from $\mathcal{D}$
17: **for** each device δ in batch **do**
18: $y_\delta \leftarrow G^b_r[\delta] + \gamma \max_{a'} Q(G'^b; \theta^-)_\delta[a']$
19: $\hat{y}_\delta \leftarrow Q(G^b; \theta)_\delta[a^b_t[\delta]]$
20: **end for**
21: Update θ by minimizing $\frac{1}{|\text{batch}|\,|\mathbf{D}|} \sum_\delta (y_\delta - \hat{y}_\delta)^2$
22: Every C steps: $\theta^- \leftarrow \theta$
23: **end while**

5. Experimental evaluation

To validate our proposed IDHGQL framework, we conduct a series of experiments designed to answer our research question (RQ, Section 1). Specifically, our evaluation aims to: (i) assess the framework's ability to learn effective multi-objective offloading policies that balance individual device constraints (battery) and system-wide objectives (cost, server load), and (ii) demonstrate the critical role of integrating collective information via aggregate computing to handle emergent phenomena like network congestion. The experiments are conducted in a simulated environment that mirrors the motivating crowd steering application.

5.1. Experimental setup

We use the Alchemist simulator [71] to model a system of 54 physical devices in a 20 m × 20 m area, configured to form two high-density zones. Alchemist was chosen because it supports the simulation of large-scale, complex networks of devices and provides seamless integration with macroprograms written in ScaFi, while also ensuring coherence with the system model underlying this work, which builds upon the macrocomponent pulverization approach introduced in [21] and originally evaluated using the same simulator. The system comprises 47 *application devices* and 7 *infrastructural devices* (5 edge servers, 2 cloud servers). Application devices can communicate with others within a 20-meter range and are connected to all edge and cloud servers. Each simulation runs for a set number of timesteps (50 or 100), where each step represents one minute. Initially, all application devices have 100% battery, and all macroprogram components are executed locally.

The learning component is implemented using PyTorch [72]. The partitioned macroprogram and aggregate computing functionalities are built on a modified version of the ScaFi framework [73], enabling the distributed execution of application components. For full reproducibility, the complete source code is available on GitHub.[2]

[2] https://github.com/nicolasfara/experiments-2024-taas-edge-cloud-intelligence/

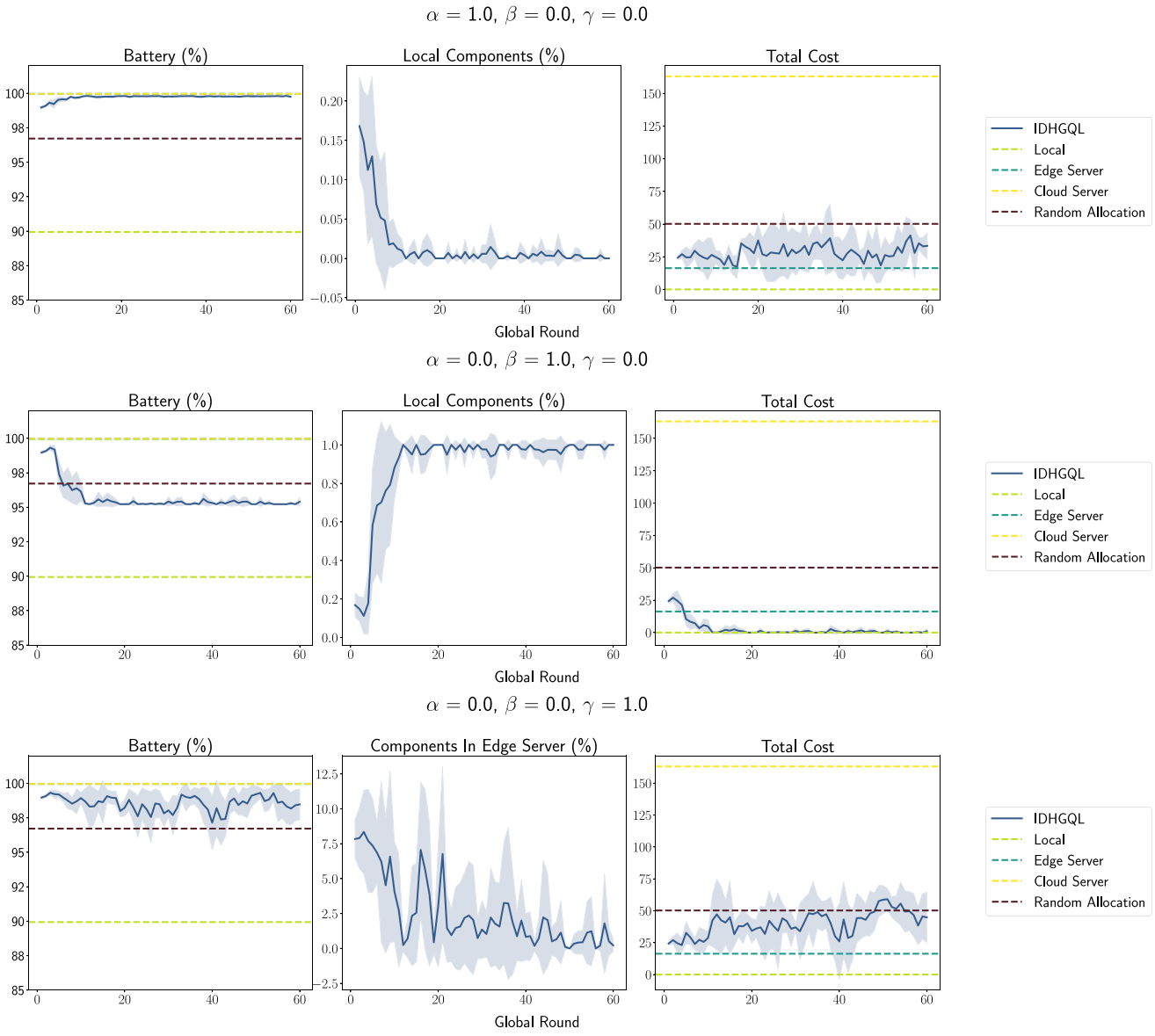


Fig. 7. Learning results in the sanity check scenario for edge cases. Each row represents the optimization of a single component of the reward at a time. The parameters α, β, and γ control the relative importance of the reward signals, corresponding respectively to battery usage, costs, and server load.

5.2. Evaluation scenarios

We design two primary experimental scenarios to systematically evaluate the IDHGQL framework.

5.2.1. Scenario 1: Multi-objective optimization capability

Goal. This experiment evaluates the learning process's stability and its ability to find policies that balance the conflicting objectives of battery consumption, infrastructure cost, and edge server load.

Baselines. We compare our learning-based approach against three static strategies: `Local Execution`, where all components run on the application devices; `Edge Offloading`, where all components are offloaded to a random edge server; `Cloud Offloading`, where all components are offloaded to a random cloud server; and `Random Allocation`, where at each time step each component is randomly allocated.

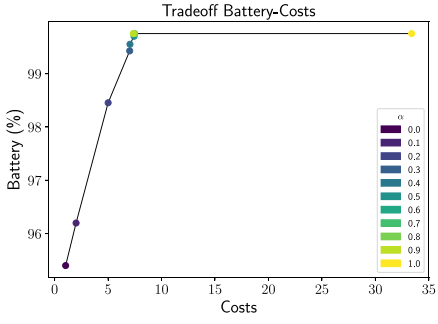


Fig. 8. Trade-off between battery consumption and total costs. Higher battery levels require full offloading, resulting in higher costs, and vice versa.

Learning configuration. The reward function is a weighted linear combination of rewards for battery preservation ($\mathcal{R}_B$), cost minimization

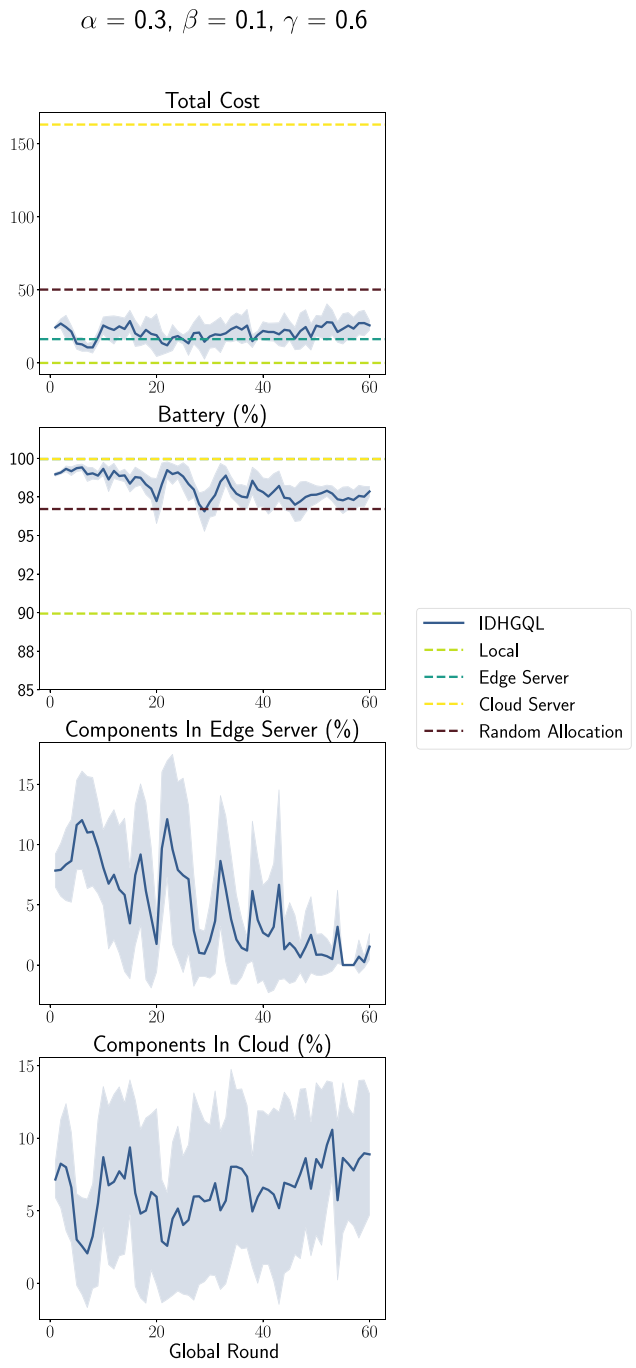


Fig. 9. Learning results in the sanity check scenario, optimizing all three components of the reward simultaneously, but with different levels of importance..

($\mathcal{R}_C$), and preventing edge server overload ($\mathcal{R}_{ES}$):

$$\mathcal{R} = \alpha \cdot \mathcal{R}_B + \beta \cdot \mathcal{R}_C + \gamma \cdot \mathcal{R}_{ES} \quad \text{(s.t. } \alpha + \beta + \gamma = 1\text{)}$$

By varying the weights (α, β, γ), we test the agent's ability to prioritize different objectives. In this scenario, the agent's observation does not include collective information ($c_{\delta,t}$) and relies only on local state and the graph structure.

5.2.2. Scenario 2: Density-aware offloading with collective information

Goal. This experiment assesses the benefit of enriching the agent's state with collective information. We create a scenario with varying device densities, where regions of high device density induce network congestion and increased latency, representing a significant collective phenomenon.

Baseline. We use a heuristic-based strategy that offloads components based on a local density estimate. A device executes components locally if its number of neighbors within a radius s exceeds a threshold th; otherwise, it offloads. This baseline highlights the challenge of setting such parameters optimally without a global perspective.

Learning configuration. Here, we fully leverage the IDHGQL framework. The agent's state is enriched with collective information ($c_{\delta,t}$), specifically the local device density computed via an aggregate program. The agent's observation also includes local battery level, offloading costs, component allocation, and estimated network latency to servers. The reward function balances battery, cost, and latency. To prove the necessity of collective information, we conduct an ablation study where IDHGQL is trained under identical conditions but with the collective density information removed from the agent's observation.

5.3. Metrics

To evaluate the performance of our proposed approach, we measure the following metrics:

- **Battery Consumption:** We measure the average percentage of battery remaining on application devices over time. The discharge model is based on the work of Grochowski et al. [74], considering a 4000 mAh battery, an EPI of 1 nJ/instruction, and a workload of 26.5M instructions per component execution.
- **Local Executed Components:** This metric represents the percentage of macroprogram components executed locally on application devices versus those offloaded to infrastructure.
- **Infrastructural Costs:** We measure the total monetary cost incurred from offloading. Edge servers have a fixed cost of $1.00/h per component instance, while cloud servers cost $10.00/h per instance.

5.4. Results and discussion

5.4.1. Analysis of multi-objective optimization

The baseline results in Figs. 7 and 9 establish the fundamental trade-offs: local execution consumes the most battery but incurs no cost, while cloud/edge offloading preserves battery at a high monetary cost.

Fig. 7 demonstrates that IDHGQL can successfully navigate these trade-offs. When optimizing solely for battery ($\alpha = 1$, top row), the agent learns to offload nearly all components, maintaining high battery levels at the expense of high costs. Conversely, when optimizing for cost ($\beta = 1$, middle row), it learns to execute everything locally, minimizing cost but depleting the battery. When targeting edge server load ($\gamma = 1$, bottom row), the agent effectively avoids offloading to the edge, showing its ability to respond to specific infrastructure constraints. Moreover, compared to the optimized parameter settings (Figs. 7 and 9), our approach consistently outperforms the random allocation baseline, demonstrating that the learning process provides a tangible advantage over uninformed policies.

The Pareto front in Fig. 8 visualizes the direct, competitive relationship between battery and cost optimization, confirming that the learned policies exist along this optimal trade-off curve. Furthermore, Fig. 9 shows a mixed-objective case ($\alpha = 0.3, \beta = 0.1, \gamma = 0.6$) where the agent learns a sophisticated policy. It heavily penalizes edge offloading, resulting in low edge usage, high battery, and correspondingly high costs. These results validate that our DQL-based approach can effectively learn policies for complex, multi-objective optimization problems.

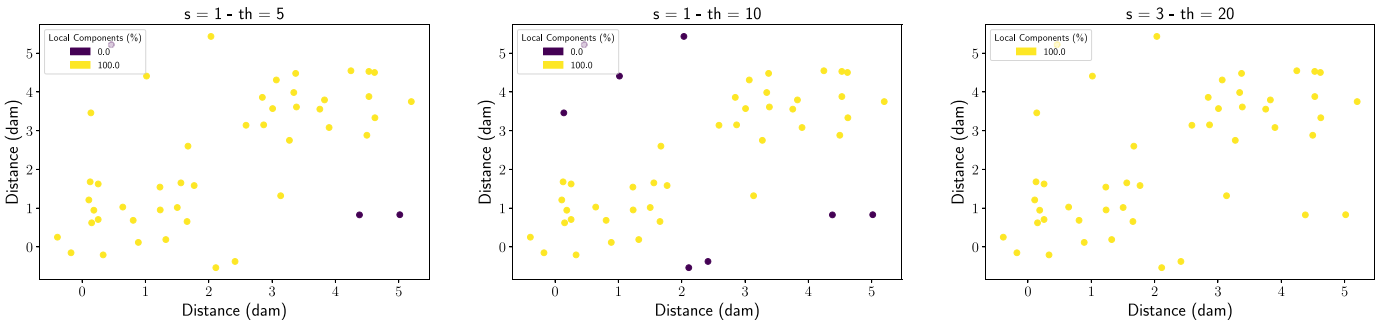


Fig. 10. Density-based offloading strategy with different fringe values s and threshold th. Yellow nodes execute components locally, violet nodes offload to edge/cloud servers.

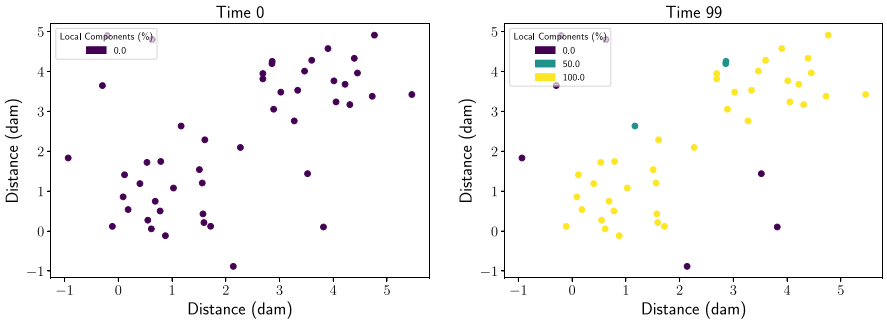


Fig. 11. Density-based offloading strategy using IDHGQL with aggregate computing (at the end of the learning process). Color indicates percentage of components executed locally. Nodes learn to execute locally in dense areas and offload in sparse areas.

5.4.2. Impact of collective information on offloading policy

This scenario directly addresses the core of our research question. Fig. 10 shows the limitations of the heuristic baseline; its performance is highly sensitive to the choice of parameters (s, th), leading to either overestimation or underestimation of density and, consequently, suboptimal offloading decisions.

In contrast, Fig. 11 shows the effectiveness of the full IDHGQL framework. By the end of the learning process (second row), the agents have learned a nuanced, spatially-aware policy: devices in high-density areas (high congestion) learn to execute components locally to avoid latency, while those in sparse areas learn to offload. Devices at the edge of crowded zones learn to partially offload, demonstrating a sophisticated balancing act.

The importance of the collective information is starkly highlighted by the ablation study in Fig. 12. Without access to the aggregate density information ($c_{\delta,t}$), the agents are unable to learn a meaningful policy. They fail to differentiate their actions based on their location and converge to a simplistic partial offloading strategy, regardless of the collective context. This directly proves that integrating collective perception is crucial for making informed decisions in response to emergent system phenomena.

Finally, we test the generalization of the learned policy in a dynamic environment where devices move. Fig. 13 shows that when three devices move from a high-density to a low-density area (at time 100), the pre-trained policy correctly adapts their behavior in real-time, causing them to switch from local execution to full offloading. This demonstrates that the policy learned by IDHGQL is not merely static but is robust and adaptive to dynamic changes in the system topology.

6. Limitations

While our proposed IDHGQL framework demonstrates significant promise in addressing the challenges of task offloading in edge-cloud systems, some limitations must be acknowledged:

Reality gap. Our evaluation is conducted in a simulated environment, which may not capture all the nuances of real-world deployments. Although we strive for realism, adopting already validated consumption models [17,74] and network conditions, real-world data and deployments are necessary to fully validate the approach.

Dynamic environments. While our experiments include dynamic scenarios like node mobility, local features changes, and varying network conditions (e.g., variable neighbor counts), we did not test highly dynamic environments such as rapid topology changes, devices frequently joining/leaving the network, or fluctuating workloads. Even though a comprehensive evaluation of such scenarios is beyond this paper's scope, we are confident that our approach can handle them due to: (i) the inherent adaptability of reinforcement learning, which allows agents to continuously learn and adjust their policies based on new experiences, and (ii) the self-stabilizing nature of aggregate computing, which ensures that collective computations converge to correct values even under message delays or temporary inconsistencies [54].

Multi-objective reward scalarization. Our approach employs a linear weighted combination of objectives ($\mathcal{R}(o) = \sum_{i=1}^{n} \lambda_i \mathcal{R}_i(o)$), a common scalarization technique that, while interpretable, cannot capture non-convex Pareto regions and may be sensitive to weight configurations [75]. The primary focus of this work is on the IDHGQL architecture—combining HeteroGNNs with aggregate computing for collective-aware decision-making—rather than on multi-objective optimization methods per se. The linear scalarization serves as a straightforward mechanism to demonstrate the framework's ability to balance competing objectives. More sophisticated approaches exist, such as multi-objective RL methods that directly learn Pareto-optimal policies [76], or adaptive weight schemes that adjust priorities based on environmental conditions. Integrating such techniques within IDHGQL represents a promising direction for future research.

Preference polarization. The current framework exhibits preference polarization in multi-objective scenarios, where agents tend to select

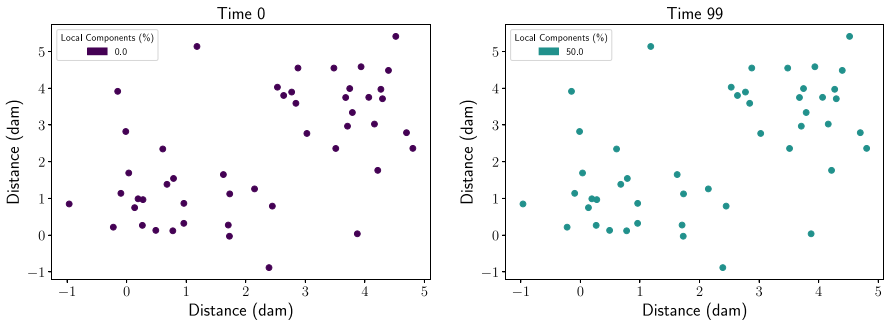


Fig. 12. Density-based offloading strategy using IDHGQL without collective information (ablative study). Time flows left to right. Colors indicate percentage of components executed locally. Without collective information, nodes fail to learn adequate policies and resort to uniform partial offloading.

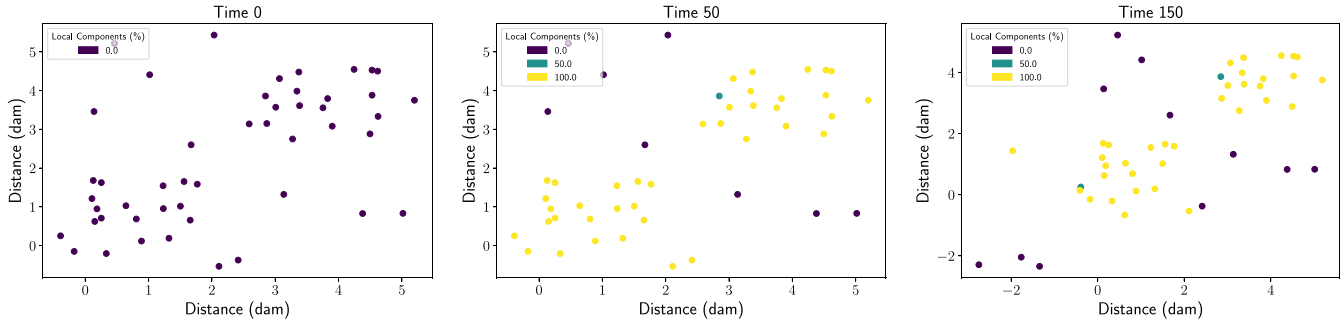


Fig. 13. Density-based offloading strategy using IDHGQL with collective information and node movements. Time flows from left to right. Each point represents an application node, and the color indicates the percentage of components being executed locally. It can be observed that when, at time 100, the three nodes in the bottom-left corner move from a dense area to a less congested one, the system is able to correctly readjust.

either cloud or edge resources exclusively rather than developing more nuanced, mixed strategies. This behavior may result from the homogeneous treatment of application nodes in our HeteroGNN architecture, which does not differentiate between devices with varying characteristics or workload patterns. Future work should explore heterogeneous node representations to enable more sophisticated policy differentiation.

Coordination mechanisms. The framework currently lacks explicit coordination mechanisms for joint component offloading, which could improve efficiency in scenarios requiring synchronized resource allocation. Investigating coordinated offloading strategies for dependent components represents an important direction for future research.

7. Conclusion

In this paper, we addressed the challenge of collective component offloading in edge-cloud computing systems by proposing the IDHGQL framework. Our approach combines HeteroGNNs with aggregate computing to enable decentralized offloading decisions that account for both individual device constraints and emergent collective phenomena. Our key contributions include: (i) a novel system model that supports partial offloading of workflow-based tasks through macro-component decomposition, (ii) the integration of collective state information via aggregate computing to inform offloading decisions, (iii) and the IDHGQL algorithm that leverages HeteroGNNs for Q-function approximation in multi-agent reinforcement learning settings.

Experimental evaluation demonstrates that IDHGQL successfully learns multi-objective offloading policies, effectively balancing battery consumption, infrastructure costs, and server load across different weight configurations. The framework shows particular strength in density-aware scenarios, where agents learn spatially-differentiated policies: devices in high-density areas execute components locally to avoid network congestion, while those in sparse areas offload to preserve battery life. The ablation study confirms that collective information integration is crucial, as agents without aggregate computing fail to learn meaningful density-aware policies.

Future work should evaluate scalability in larger, more complex edge-cloud deployments, and extend the approach to handle time-varying network conditions and more complex collective behaviors.

CRediT authorship contribution statement

Nicolas Farabegoli: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Data curation, Conceptualization. **Davide Domini:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Data curation, Conceptualization. **Gianluca Aguzzi:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Data curation, Conceptualization. **Mirko Viroli:** Supervision, Methodology, Funding acquisition, Conceptualization.

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 publicly available after the publication.

References

[1] M. Satyanarayanan, The emergence of edge computing, Computer 50 (1) (2017) 30–39, http://dx.doi.org/10.1109/MC.2017.9.

[2] D. Rosendo, A. Costan, P. Valduriez, G. Antoniu, Distributed intelligence on the edge-to-cloud continuum: A systematic literature review, J. Parallel Distrib. Comput. 166 (2022) 71–94, http://dx.doi.org/10.1016/J.JPDC.2022.04.004.

[3] P. Mach, Z. Becvar, Mobile edge computing: A survey on architecture and computation offloading, IEEE Commun. Surv. Tutorials 19 (3) (2017) 1628–1656, http://dx.doi.org/10.1109/COMST.2017.2682318.

[4] H. Lin, S. Zeadally, Z. Chen, H. Labiod, L. Wang, A survey on computation offloading modeling for edge computing, J. Netw. Comput. Appl. 169 (2020) 102781, http://dx.doi.org/10.1016/J.JNCA.2020.102781.

[5] A. Islam, A. Debnath, M. Ghose, S. Chakraborty, A survey on task offloading in multi-access edge computing, J. Syst. Archit. 118 (2021) 102225, http://dx.doi.org/10.1016/J.SYSARC.2021.102225.

[6] C. Jiang, X. Cheng, H. Gao, X. Zhou, J. Wan, Toward computation offloading in edge computing: A survey, IEEE Access 7 (2019) 131543–131558, http://dx.doi.org/10.1109/ACCESS.2019.2938660.

[7] F. Saeik, M. Avgeris, D. Spatharakis, N. Santi, D. Dechouniotis, J. Violos, A. Leivadeas, N. Athanasopoulos, N. Mitton, S. Papavassiliou, Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions, Comput. Netw. 195 (2021) 108177, http://dx.doi.org/10.1016/J.COMNET.2021.108177.

[8] A.A. Al-Habob, O.A. Dobre, A.G. Armada, S. Muhaidat, Task scheduling for mobile edge computing using genetic algorithm and conflict graphs, IEEE Trans. Veh. Technol. 69 (8) (2020) 8805–8819, http://dx.doi.org/10.1109/TVT.2020.2995146.

[9] Z. Liao, J. Peng, B. Xiong, J. Huang, Adaptive offloading in mobile-edge computing for ultra-dense cellular networks based on genetic algorithm, J. Cloud Comput. 10 (1) (2021) 15, http://dx.doi.org/10.1186/S13677-021-00232-Y.

[10] H. Song, B. Gu, K. Son, W. Choi, Joint optimization of edge computing server deployment and user offloading associations in wireless edge network via a genetic algorithm, IEEE Trans. Netw. Sci. Eng. 9 (4) (2022) 2535–2548, http://dx.doi.org/10.1109/TNSE.2022.3165372.

[11] A. Ferdowsi, U. Challita, W. Saad, Deep learning for reliable mobile edge analytics in intelligent transportation systems: An overview, IEEE Veh. Technol. Mag. 14 (1) (2019) 62–70, http://dx.doi.org/10.1109/MVT.2018.2883777.

[12] A. Ferscha, Collective adaptive systems, in: K. Mase, M. Langheinrich, D. Gatica-Perez, H. Gellersen, T. Choudhury, K. Yatani (Eds.), Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers, UbiComp/ISWC Adjunct 2015, Osaka, Japan, September 7-11, 2015, ACM, 2015, pp. 893–895, http://dx.doi.org/10.1145/2800835.2809508.

[13] I. Crnkovic, S. Sentilles, A. Vulgarakis, M.R.V. Chaudron, A classification framework for software component models, IEEE Trans. Softw. Eng. 37 (5) (2011) 593–615, http://dx.doi.org/10.1109/TSE.2010.83.

[14] R.E. Ballouli, S. Bensalem, M. Bozga, J. Sifakis, Four exercises in programming dynamic reconfigurable systems: Methodology and solution in DR-BIP, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and Validation. Distributed Systems - 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part III, in: Lecture Notes in Computer Science, vol. 11246, Springer, 2018, pp. 304–320, http://dx.doi.org/10.1007/978-3-030-03424-5_20.

[15] N. Medvidovic, R.N. Taylor, A classification and comparison framework for software architecture description languages, IEEE Trans. Softw. Eng. 26 (1) (2000) 70–93, http://dx.doi.org/10.1109/32.825767.

[16] A.L. Lemos, F. Daniel, B. Benatallah, Web service composition: A survey of techniques and tools, ACM Comput. Surv. 48 (3) (2016) 33:1–33:41, http://dx.doi.org/10.1145/2831270.

[17] N. Farabegoli, D. Pianini, R. Casadei, M. Viroli, Scalability through pulverisation: Declarative deployment reconfiguration at runtime, Future Gener. Comput. Syst. 161 (2024) 545–558, http://dx.doi.org/10.1016/J.FUTURE.2024.07.042.

[18] F. Chai, Q. Zhang, H. Yao, X. Xin, R. Gao, M. Guizani, Joint multi-task offloading and resource allocation for mobile edge computing systems in satellite IoT, IEEE Trans. Veh. Technol. 72 (6) (2023) 7783–7795, http://dx.doi.org/10.1109/TVT.2023.3238771.

[19] Y. Cui, H. Li, D. Zhang, A. Zhu, Y. Li, H. Qiang, Multiagent reinforcement learning-based cooperative multitype task offloading strategy for internet of vehicles in B5G/6G network, IEEE Internet Things J. 10 (14) (2023) 12248–12260, http://dx.doi.org/10.1109/JIOT.2023.3245721.

[20] P. Wang, Y. Pan, C. Lin, H. Qi, J. Ren, N. Wang, Z. Yu, D. Zhou, Q. Zhang, Graph optimized data offloading for crowd-AI hybrid urban tracking in intelligent transportation systems, IEEE Trans. Intell. Transp. Syst. 24 (1) (2023) 1075–1087, http://dx.doi.org/10.1109/TITS.2022.3141885.

[21] N. Farabegoli, M. Viroli, R. Casadei, Flexible self-organisation for the cloud-edge continuum: a macro-programming approach, in: IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2024, Aarhus, Denmark, IEEE, 2024.

[22] J. Beal, D. Pianini, M. Viroli, Aggregate programming for the internet of things, Computer 48 (9) (2015) 22–30, http://dx.doi.org/10.1109/MC.2015.261.

[23] K. Cao, Y. Liu, G. Meng, Q. Sun, An overview on edge computing research, IEEE Access 8 (2020) 85714–85728, http://dx.doi.org/10.1109/ACCESS.2020.2991734.

[24] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, W. Shi, Edge computing for autonomous driving: Opportunities and challenges, Proc. IEEE 107 (8) (2019) 1697–1716, http://dx.doi.org/10.1109/JPROC.2019.2915983.

[25] A. Kamilaris, A. Pitsillides, Mobile phone computing and the internet of things: A survey, IEEE Internet Things J. 3 (6) (2016) 885–898, http://dx.doi.org/10.1109/JIOT.2016.2600569.

[26] A. Heidari, M.A.J. Jamali, N.J. Navimipour, S. Akbarpour, Internet of things offloading: Ongoing issues, opportunities, and future challenges, Int. J. Commun. Syst. 33 (14) (2020) http://dx.doi.org/10.1002/DAC.4474.

[27] M. Aazam, S. Zeadally, K.A. Harras, Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities, Future Gener. Comput. Syst. 87 (2018) 278–289, http://dx.doi.org/10.1016/J.FUTURE.2018.04.057.

[28] J. Wang, J. Pan, F. Esposito, P. Calyam, Z. Yang, P. Mohapatra, Edge cloud offloading algorithms: Issues, methods, and perspectives, ACM Comput. Surv. 52 (1) (2019) 2:1–2:23, http://dx.doi.org/10.1145/3284387.

[29] B. Wang, C. Wang, W. Huang, Y. Song, X. Qin, A survey and taxonomy on task offloading for edge-cloud computing, IEEE Access 8 (2020) 186080–186101, http://dx.doi.org/10.1109/ACCESS.2020.3029649.

[30] K. Guo, M. Yang, Y. Zhang, J. Cao, Joint computation offloading and bandwidth assignment in cloud-assisted edge computing, IEEE Trans. Cloud Comput. 10 (1) (2022) 451–460, http://dx.doi.org/10.1109/TCC.2019.2950395.

[31] J. Meng, H. Tan, C. Xu, W. Cao, L. Liu, B. Li, Dedas: Online task dispatching and scheduling with bandwidth constraint in edge computing, in: 2019 IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, April 29 - May 2, 2019, IEEE, 2019, pp. 2287–2295, http://dx.doi.org/10.1109/INFOCOM.2019.8737577.

[32] I. Sarkar, S. Kumar, M. Mukherjee, An optimized task placement in computational offloading for fog-cloud computing networks, in: 2019 IEEE International Conference on Advanced Networks and Telecommunications Systems, ANTS 2019, Goa, India, December 16-19, 2019, IEEE, 2019, pp. 1–5, http://dx.doi.org/10.1109/ANTS47819.2019.9118134.

[33] M. Alkhalaileh, R.N. Calheiros, Q.V. Nguyen, B. Javadi, Data-intensive application scheduling on mobile edge cloud computing, J. Netw. Comput. Appl. 167 (2020) 102735, http://dx.doi.org/10.1016/J.JNCA.2020.102735.

[34] Y. Miao, G. Wu, M. Li, A. Ghoneim, M. Al-Rakhami, M.S. Hossain, Intelligent task prediction and computation offloading based on mobile-edge cloud computing, Future Gener. Comput. Syst. 102 (2020) 925–931, http://dx.doi.org/10.1016/J.FUTURE.2019.09.035.

[35] C. Kai, H. Zhou, Y. Yi, W. Huang, Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability, IEEE Trans. Cogn. Commun. Netw. 7 (2) (2021) 624–634, http://dx.doi.org/10.1109/TCCN.2020.3018159.

[36] T. Ouyang, R. Li, X. Chen, Z. Zhou, X. Tang, Adaptive user-managed service placement for mobile edge computing: An online learning approach, in: 2019 IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, April 29 - May 2, 2019, IEEE, 2019, pp. 1468–1476, http://dx.doi.org/10.1109/INFOCOM.2019.8737560.

[37] C. Zhang, H. Du, Q. Ye, C. Liu, H. Yuan, DMRA: a decentralized resource allocation scheme for multi-sp mobile edge computing, in: 39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019, Dallas, TX, USA, July 7-10, 2019, IEEE, 2019, pp. 390–398, http://dx.doi.org/10.1109/ICDCS.2019.00046.

[38] O. Chabbouh, S.B. Rejeb, Z. Choukair, N. Agoulmine, A strategy for joint service offloading and scheduling in heterogeneous cloud radio access networks, EURASIP J. Wirel. Commun. Netw. 2017 (2017) 196, http://dx.doi.org/10.1186/S13638-017-0978-0.

[39] Z. Wang, S. Zheng, Q. Ge, K. Li, Online offloading scheduling and resource allocation algorithms for vehicular edge computing system, IEEE Access 8 (2020) 52428–52442, http://dx.doi.org/10.1109/ACCESS.2020.2981045.

[40] J. Zhao, Q. Li, Y. Gong, K. Zhang, Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks, IEEE Trans. Veh. Technol. 68 (8) (2019) 7944–7956, http://dx.doi.org/10.1109/TVT.2019.2917890.

[41] X. Cheng, F. Lyu, W. Quan, C. Zhou, H. He, W. Shi, X. Shen, Space/aerial-assisted computing offloading for IoT applications: A learning-based approach, IEEE J. Sel. Areas Commun. 37 (5) (2019) 1117–1129, http://dx.doi.org/10.1109/JSAC.2019.2906789.

[42] T.T. Nguyen, V.N. Ha, L.B. Le, R. Schober, Joint data compression and computation offloading in hierarchical fog-cloud systems, IEEE Trans. Wirel. Commun. 19 (1) (2020) 293–309, http://dx.doi.org/10.1109/TWC.2019.2944165.

[43] J. Long, Y. Luo, X. Zhu, E. Luo, M. Huang, Computation offloading through mobile vehicles in IoT-edge-cloud network, EURASIP J. Wirel. Commun. Netw. 2020 (1) (2020) 244, http://dx.doi.org/10.1186/S13638-020-01848-5.

[44] M. Khayyat, I.A. Elgendy, A. Muthanna, A.S. Alshahrani, S. Alharbi, A. Koucheryavy, Advanced deep learning-based computational offloading for multilevel vehicular edge-cloud computing networks, IEEE Access 8 (2020) 137052–137062, http://dx.doi.org/10.1109/ACCESS.2020.3011705.

[45] G. Aguzzi, C. Savaglio, Engineering distributed collective intelligence in cyber-physical swarms, in: 20th International Conference on Distributed Computing in Smart Systems and the Internet of Things, DCOSS-IoT 2024, Abu Dhabi, United Arab Emirates, April 29 - May 1, 2024, IEEE, 2024, pp. 570–575, http://dx.doi.org/10.1109/DCOSS-IOT61029.2024.00089.

[46] G.D. Abowd, Beyond weiser: From ubiquitous to collective computing, Computer 49 (1) (2016) 17–23, http://dx.doi.org/10.1109/MC.2016.22.

[47] D. Domini, N. Farabegoli, G. Aguzzi, M. Viroli, Towards intelligent pulverized systems: a modern approach for edge-cloud services, in: M. Alderighi, M. Baldoni, C. Baroglio, R. Micalizio, S. Tedeschi (Eds.), Proceedings of the 25th Workshop "from Objects To Agents", Bard (Aosta), Italy, July 8-10, 2024, in: CEUR Workshop Proceedings, vol. 3735, CEUR-WS.org, 2024, pp. 233–251, URL https://ceur-ws.org/Vol-3735/paper_19.pdf.

[48] A. Sassi, C. Borean, R. Giannantonio, M. Mamei, D. Mana, F. Zambonelli, Crowd steering in public spaces: Approaches and strategies, in: Y. Wu, G. Min, N. Georgalas, J. Hu, L. Atzori, X. Jin, S.A. Jarvis, L.C. Liu, R.A. Calvo (Eds.), 15th IEEE International Conference on Computer and Information Technology, CIT 2015; 14th IEEE International Conference on Ubiquitous Computing and Communications, IUCC 2015; 13th IEEE International Conference on Dependable, Autonomic and Secure Computing, DASC 2015; 13th IEEE International Conference on Pervasive Intelligence and Computing, PICom 2015, Liverpool, United Kingdom, October 26-28, 2015, IEEE, 2015, pp. 2098–2105, http://dx.doi.org/10.1109/CIT/IUCC/DASC/PICOM.2015.312.

[49] S. Chung, A.A. Paranjape, P.M. Dames, S. Shen, V. Kumar, A survey on aerial swarm robotics, IEEE Trans. Robot. 34 (4) (2018) 837–855, http://dx.doi.org/10.1109/TRO.2018.2857475.

[50] M. Viroli, J. Beal, F. Damiani, G. Audrito, R. Casadei, D. Pianini, From distributed coordination to field calculus and aggregate computing, J. Log. Algebraic Methods Program. 109 (2019) http://dx.doi.org/10.1016/J.JLAMP.2019.100486.

[51] G. Mone, Rise of the swarm, Commun. ACM 56 (3) (2013) 16–17, http://dx.doi.org/10.1145/2428556.2428562.

[52] G. Francesca, M. Birattari, Automatic design of robot swarms: Achievements and challenges, Front. Robot. AI 3 (2016) 29, http://dx.doi.org/10.3389/FROBT.2016.00029.

[53] R. Casadei, Macroprogramming: Concepts, state of the art, and opportunities of macroscopic behaviour modelling, ACM Comput. Surv. 55 (13s) (2023) 275:1–275:37, http://dx.doi.org/10.1145/3579353.

[54] G. Audrito, M. Viroli, F. Damiani, D. Pianini, J. Beal, A higher-order calculus of computational fields, ACM Trans. Comput. Log. 20 (1) (2019) 5:1–5:55, http://dx.doi.org/10.1145/3285956.

[55] R. Casadei, G. Fortino, D. Pianini, W. Russo, C. Savaglio, M. Viroli, Modelling and simulation of opportunistic IoT services with aggregate computing, Future Gener. Comput. Syst. 91 (2019) 252–262, http://dx.doi.org/10.1016/J.FUTURE.2018.09.005.

[56] G. Aguzzi, R. Casadei, M. Viroli, MacroSwarm: A field-based compositional framework for swarm programming, in: S. Jongmans, A. Lopes (Eds.), Coordination Models and Languages - 25th IFIP WG 6.1 International Conference, COORDINATION 2023, Held As Part of the 18th International Federated Conference on Distributed Computing Techniques, DisCoTec 2023, Lisbon, Portugal, June 19-23, 2023, Proceedings, in: Lecture Notes in Computer Science, vol. 13908, Springer, 2023, pp. 31–51, http://dx.doi.org/10.1007/978-3-031-35361-1_2.

[57] D. Grushchak, J. Kline, D. Pianini, N. Farabegoli, G. Aguzzi, M. Baiardi, C. Stewart, Decentralized multi-drone coordination for wildlife video acquisition, in: 5th IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2024, IEEE, 2024.

[58] J. Li, W. Zhang, Q. Wang, Y. Liu, Deep Q-learning-based partial computation offloading in multi-access edge computing, IEEE Internet Things J. 8 (4) (2021) 2345–2356, http://dx.doi.org/10.1109/JIOT.2020.3039945.

[59] P. Sun, L. Zhao, Y. Zhang, GRLO: Graph reinforcement learning offloading for mobile edge computing, in: Proceedings of the IEEE INFOCOM, 2021, pp. 1–10, http://dx.doi.org/10.1109/INFOCOM42981.2021.9488672.

[60] H. Wang, L. Chen, M. Xu, M-GNRL: Multi-agent graph neural reinforcement learning for task offloading in MEC, IEEE Trans. Mob. Comput. 23 (2) (2024) 567–578, http://dx.doi.org/10.1109/TMC.2023.3284712.

[61] K. Moghaddasi, R. Jurdak, An energy-aware distributed federated soft actor-critic framework for intelligent task offloading in vehicular mobile edge computing networks, Ad Hoc Networks 180 (2026) 104043, http://dx.doi.org/10.1016/J.ADHOC.2025.104043.

[62] W. Cao, C. Deng, GAT-ppo: Graph attention actor-critic for DAG task scheduling in edge computing, in: Proceedings of IEEE ICWS, 2023, pp. 123–130, http://dx.doi.org/10.1109/ICWS2023.00023.

[63] X. Wu, H. Zhang, Y. Li, Task offloading in vehicular networks using heterogeneous graph attention reinforcement learning, IEEE Trans. Intell. Transp. Syst. (2025) http://dx.doi.org/10.1109/TITS.2025.3300123.

[64] X. Wu, B. Zhang, H. Chen, PORA-MATD3: Multi-agent task offloading with mobility-aware TD3 in vehicular edge computing, in: IEEE International Conference on Distributed Computing Systems, ICDCS, 2025, pp. 222–231, http://dx.doi.org/10.1109/ICDCS.2025.00100.

[65] A. Pamuklu, E. Kara, C. Yigit, Heterogeneous GNN-based task offloading for smart agriculture with UAVs and IoT devices, in: Proceedings of IEEE GLOBECOM, 2023, pp. 1–6, http://dx.doi.org/10.1109/GLOBECOM2023.10001234.

[66] M. Chen, A. Liu, N.N. Xiong, Y. Ren, H.H. Song, ANNS: an intelligent advanced non-convex non-smooth scheme for IRS-aided next generation mobile communication networks, IEEE Trans. Mob. Comput. 24 (9) (2025) 8959–8973, http://dx.doi.org/10.1109/TMC.2025.3559099.

[67] M. Chen, A. Liu, N.N. Xiong, H. Song, V.C.M. Leung, SGPL: an intelligent game-based secure collaborative communication scheme for metaverse over 5G and beyond networks, IEEE J. Sel. Areas Commun. 42 (3) (2024) 767–782, http://dx.doi.org/10.1109/JSAC.2023.3345403.

[68] M. Viroli, G. Audrito, J. Beal, F. Damiani, D. Pianini, Engineering resilient collective adaptive systems by self-stabilisation, ACM Trans. Model. Comput. Simul. 28 (2) (2018) 16:1–16:28, http://dx.doi.org/10.1145/3177774.

[69] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph attention networks, 2017, CoRR, arXiv:1710.10903.

[70] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, P.S. Yu, Heterogeneous graph attention network, in: L. Liu, R.W. White, A. Mantrach, F. Silvestri, J.J. McAuley, R. Baeza-Yates, L. Zia (Eds.), The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019, ACM, 2019, pp. 2022–2032, http://dx.doi.org/10.1145/3308558.3313562.

[71] D. Pianini, S. Montagna, M. Viroli, Chemical-oriented simulation of computational systems with ALCHEMIST, J. Simul. 7 (3) (2013) 202–215, http://dx.doi.org/10.1057/JOS.2012.27.

[72] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic differentiation in pytorch, 2017.

[73] R. Casadei, M. Viroli, G. Aguzzi, D. Pianini, ScaFi: A scala DSL and toolkit for aggregate programming, SoftwareX 20 (2022) 101248, http://dx.doi.org/10.1016/J.SOFTX.2022.101248.

[74] E.T. Grochowski, M. Annavaram, Energy per instruction trends in intel® microprocessors, 2006, URL https://api.semanticscholar.org/CorpusID:17814146.

[75] C.F. Hayes, R. Rădulescu, E. Bargiacchi, J. Källström, M. Macfarlane, M. Reymond, T. Verstraeten, L.M. Zintgraf, R. Dazeley, F. Heintz, E. Howley, A.A. Irissappane, P. Mannion, A. Nowé, G. Ramos, M. Restelli, P. Vamplew, D.M. Roijers, A practical guide to multi-objective reinforcement learning and planning, Auton. Agents Multi-Agent Syst. 36 (1) (2022) http://dx.doi.org/10.1007/s10458-022-09552-y.

[76] A. Navon, A. Shamsian, G. Chechik, E. Fetaya, Learning the Pareto front with hypernetworks, in: International Conference on Learning Representations, 2021, URL https://openreview.net/forum?id=NjF772F4ZZR.