Towards Intelligent Pulverized Systems: a Modern Approach for Edge-Cloud Services*

Davide Domini[1], Nicolas Farabegoli[1], Gianluca Aguzzi[1] and Mirko Viroli[1]

[1]Università di Bologna – ALMA MATER STUDIOURM, Via Dell'Università 50, 47521 Cesena, Italy

Abstract

Emerging trends are leveraging the potential of the edge-cloud continuum to foster the creation of smart services capable of adapting to the dynamic nature of modern computing landscapes. This adaptation is achievable through two primary methods: by leveraging the underlying architecture to refine machine learning algorithms, and by implementing machine learning algorithms to optimize the distribution of resources and services intelligently. This paper explores the latter approach, focusing on recent advancements in pulverized architecture, collective intelligence, and many-agent reinforcement learning systems. This novel trend, which we refer to as *intelligent pulverized system* (IPS), aims to create a new generation of services that can adapt to the complex and dynamic nature of the edge-cloud continuum. Our proposed learning framework integrates many-agent reinforcement learning, graph neural networks, and aggregate computing to create intelligent services tailored for this environment. We discuss the application of this framework across different levels of the pulverization model, illustrating its potential to enhance the adaptability and efficiency of services within the edge-cloud continuum.

Keywords

Edge Cloud Continuum, Many-Agent Reinforcement Learning, Pulverization

1. Introduction

Recent technological developments are fostering a computational landscape that is increasingly articulated and complex across various levels. The historical distinction between cloud, fog, and edge computing is becoming progressively blurred, giving rise to what is known as the edge-cloud continuum (ECC) [1, 2]. This shift is driven by the necessity for services developed on these platforms to be highly opportunistic, capable of dynamically moving up and down the continuum based on local needs and computational requirements.

The ECC is a valuable resource for creating intelligent services that leverage recent advancements in machine learning [3]. These services must be able to utilize the continuum's full

WOA 2024: 25th Workshop "From Objects to Agents", July 8-10, 2024, Forte di Bard (AO), Italy

*Corresponding author.

†These authors contributed equally.

davide.domini@unibo.it (D. Domini); nicolas.farabegoli@unibo.it (N. Farabegoli); gianluca.aguzzi@unibo.it (G. Aguzzi); mirko.viroli@unibo.it (M. Viroli)

https://www.unibo.it/sitoweb/davide.domini/en (D. Domini); https://www.unibo.it/sitoweb/nicolas.farabegoli/en (N. Farabegoli); https://www.unibo.it/sitoweb/gianluca.aguzzi/en (G. Aguzzi); https://www.unibo.it/sitoweb/mirko.viroli/en/ (M. Viroli)

0009-0006-8337-8990 (D. Domini); 0000-0002-7321-358X (N. Farabegoli); 0000-0002-1553-4561 (G. Aguzzi); 0000-0003-2702-5702 (M. Viroli)

© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR-WS.org/Vol-3735/paper_19.pdf

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

potential by adapting to the changing conditions and demands of the environment. Furthermore, making the continuum itself more intelligent is essential for developing complex applications, particularly those that are collective in nature. Modern applications, such as those in pervasive [4], collective [5], and ubiquitous [6] computing, often require collective computations rather than mere local computations. Examples include applications for crowd control, traffic management, and energy monitoring [7, 8]. These scenarios highlight the need for a holistic view of the system rather than just local optimizations.

This paper proposes a new vision of *intelligent pulverized system* (ISP) for creating a smarter ECC by leveraging recent developments in many-agent reinforcement learning [9], pulverization [10, 11], and macroprogramming [12]. Pulverization is a modern approach that describes collective computations capable of being "broken down" and distributed across multiple hosts, which is crucial for fully exploiting the continuum [10]. Additionally, we explore how macroprogramming approaches can effectively capture the collective aspect of these systems, providing a comprehensive framework for intelligent service development within the ECC [12]. This structured approach aims to enhance the intelligence of the ECC, enabling the development of sophisticated, adaptive, and collective applications that can efficiently utilize the continuum's capabilities.

The rest of the paper is structured as follows: Section 2 provides an overview of the edge-cloud continuum, its characteristics, and the concept of pulverization; Section 3 discusses the integration of many-agent reinforcement learning, GNNs, and aggregate computing to develop intelligent services; Section 4 details the application of intelligent collective services to various levels of pulverization, including intelligent reconfiguration, adaptive communication, and scheduling; and Section 5 summarizes our findings and outlines potential future directions for research in this area.

2. Background

2.1. Edge-Cloud Continuum

The rapid adoption of cloud technologies was driven by the benefit of having high availability, and scalability of computational resources. The rise of cloud computing has led to the proliferation of new applications involving smart devices; these applications are typically reified in the Internet of Things (IoT) [13] and Cyber-Physical System (CPS) [14] domains. To meet the increasing demand of low-latency and high-throughput of such applications, the *edge computing* [15] paradigm has been introduced, to bring computational resources closer to end users, thus reducing latency and network traffic.

In this context, the highly distributed nature of such applications and the vast amount of generated data have driven the shift from a centralized cloud computing paradigm to a more distributed model, where the cloud is integrated with the edge. Such integration gave rise to a new, hybrid paradigm, called *edge-cloud continuum* [1, 2]. Depending on the specific scenario and scope, the Edge-Cloud Continuum (ECC) can have slightly different meanings [2]. Generally, the ECC refers to a distributed computing environment that extends from the cloud to the edge, where this continuum can be effectively leveraged to optimize metrics and quality of service.

This new paradigm is characterized by a high level of heterogeneity in terms of devices,

networks, and services. Consequently, various types of hardware devices are part of the continuum, ranging from high-end servers to wearable or embedded devices equipped with microcontrollers. Ideally, any devices equipped with a computational unit and a network connection can be part of the continuum.

This paradigm includes diverse hardware and software stacks (e.g., Linux, embedded firmware, Android Wear, Docker), and various network technologies (e.g., Ethernet, WiFi, ZigBee, LoRa, 5G/6G), making the ECC a highly heterogeneous and dynamic environment, challenging to manage in terms of resource allocation and service deployment. While the continuum offers new possibilities to optimize the performance of applications, the complexity of the environment makes it difficult to fully exploit the full potential of the ECC, fostering the need for novel intelligent solutions.

The rapid increase in data volumes from various applications is driving the evolution of distributed digital infrastructures for data analytics and Machine Learning (ML). Traditionally reliant on cloud infrastructures, the need for low-latency and secure processing has shifted some processing to IoT edge devices, finding in the ECC a prominent solution for the distributed intelligence [3].

2.2. Macroprogramming

When dealing with large-scale systems, such as ECC, it is helpful to shift the focus from individual devices to the collective system, as the behavior of the system as a whole is more important than the behavior of individual devices. Consider, for example, a crowd congestion alarm system. In this case, each smartphone could be part of an ECC system aimed at identifying crowded areas and intelligently guiding the crowd to disperse, thereby avoiding emergencies. Programming such behavior with a device-centric view is complicated because the collective paradigm is not incorporated at that level. In this direction, a modern area of research aims to change the programmer's focus from the device to the aggregate. This area of research is called *macroprogramming* [12], which has its roots in Wireless Sensor Networks (WSN) [16] and has evolved to be used in opportunistic edge computing contexts today [17].

The core of macroprogramming lies in identifying a collective abstraction that then becomes a first-class citizen of the programming language. Various languages have been defined in this scenario, but most of them are designed for specific applications (e.g., PyoT for IoT [18] and Buzz [19] for robotics). A modern approach to macroprogramming is *aggregate computing* [20], which is a functional top-down global-to-local paradigm that allows defining collective behaviors through the manipulation of a distributed data structure called *computational fields*. This macro view is then broken down into local executions of individual devices, which, by computing *iteratively* and *continuously*, achieve the specified collective behavior.

Aggregate computing has been used in the context of ECC to create additive behaviors based on devices' location in space [12]. In this paper, however, we will focus on using this collective abstraction to better guide machine learning in the ECC context. Indeed, aggregate computing was shown to be a powerful tool to guide machine learning in the context of large-scale distributed systems [21, 22, 23].

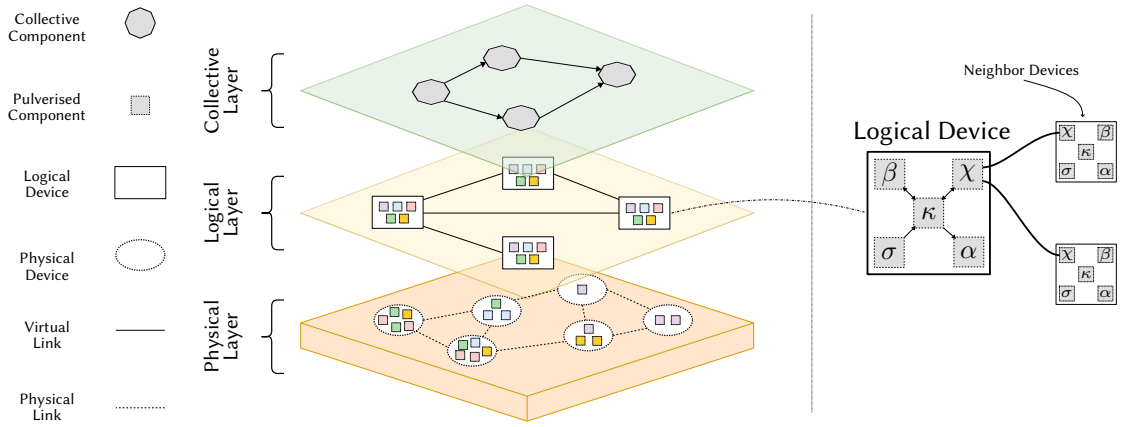

Figure 1: Representation of the different levels of a pulverized system. On the left side, the three abstraction levels: the logical layer, the physical layer, and the infrastructure layer. On the right side, the logical device pulverized into the five subcomponents.

2.3. Pulverization

Pulverization [10, 11] is an approach to simplify the design and deployment of (collective) distributed applications, by devising a peculiar partitioning into *independently* deployable components. The main goal is to provide the developer with a way to specify the application logic in a deployment-agnostic way, and let the platform/middleware take care of the deployment and the communication between the components.

In this context, this approach mainly targets the deployment of collective applications, where through macroprogramming, the application is designed as a composition of collective components (*collective layer* depicted in Figure 1). To achieve this, the pulverization model defines two main abstraction levels: the *logical layer* and the *physical layer*, as depicted in Figure 1. The former is the level at which the developer reasons about the application logic, while the latter is the level at which the specified application is deployed and executed.

System Model at the logical level, the developer specifies the application as an ensemble of *application-specific devices* forming an arbitrary topology. An application-specific device can be "pulverized" if it can be decomposed into *pulverized components* representing: a set of *sensors* (σ), a set of *actuators* (α), a *state* holding the device's knowledge (κ), a *communication* component to interact with other devices (χ), and a *behavior* component defining the device's logic (β). The Figure 1 shows a *logical device* pulverized into the five subcomponents (right side of the figure).

Typically, the logical layer is composed of several (hundreds or thousands) devices forming a dynamic graph. Each *logical device* has a neighborhood with which it can interact. Such neighbourhoods can vary over and the way the neighborhood is defined can be application-specific (like proximity-based of based on physical connections). In pulverized system, the χ component is in charge of managing the communication between the devices, implementing the neighborhood definition and the message exchange.

Execution Model the components interact with each other to perform a MAPE-K like loop: the sensors (σ) collect data from the environment and the neighbour's messages are received by the communication component (χ), the behavior component (β) applies the logic taking as input the state (κ) and the data from the sensors (σ), and the neighbor's messages. The produced output contains the new state (κ), the messages to be sent to the neighbors via the communication component (χ), and the prescriptive actions to be executed by the actuators (α).

Notably, the flexibility of this model allows the implementation of such an interaction loop in many ways. The most common way to implement this interaction is via a round-based execution model, where each component at fixed intervals sends and/or receives messages from the other components. However, more complex implementation can be devised, like a pure reactive model where the components react to the messages received without a fixed schedule; in this case, the β component leads (intelligently) the interaction loop. Another way to implement such a loop is to adopt a "best-effort" approach, where all components requiring input from the other components compute their logic against the most recent available data.

Flexible pulverized Deployments for the ECC The pulverization approach is particularly suitable for collective applications, enabling non-trivial deployments. When macroprogramming like aggregate computing [20] are used, the pulverization model naturally integrates with them, simplifying development, especially in ECC infrastructures. Otherwise, the pulverization can be exploited to design custom systems with the pulverization in mind.

Whenever an application is pulverized, a *deployment mapping* must be provided between the logical and the infrastructure levels. Typically, in this mapping, a logical device is typically deployed on either multiple physical devices or a single one. A logical device is often associated with an application-level physical device that needs to be managed or controlled (e.g., a sensor, a drone, or a person equipped with wearables). However, the capabilities of that physical device might not be enough to host all the components, and the pulverization might require the deployment of the components on different physical devices, namely infrastructure-level devices. The main advantage of pulverization is enabling many deployments without affecting or changing the application logic. Thanks to this feature, a system can be deployed on arbitrary infrastructural devices as long as they provide the required capabilities to host the components (for example, a sensor component can be executed only on devices hosting the appropriate sensors, and the behavior component can be allocated on that host having sufficient computational power). In this sense, the ECC can be exploited to opportunistically allocate the components making possible deployments otherwise impossible.

2.4. Machine Learning for ECC

The emergence of the edge-cloud continuum offers significant potential for the development of new intelligent systems, particularly for applications that rely on machine learning techniques. In the following, we provide the current state of the art in the field of machine learning for the edge-cloud continuum.

ECC for ML Applications ECC has attracted significant attention from the ML community: highly distributed networks composed of devices capable of generating a large amount of data

can provide new resources to enhance the quality of ML models. However, these devices often have limited computing resources and latency constraints that prevent both local learning and offloading learning tasks to cloud servers. Conversely, the ECC provides a great platform with edge servers that add computational resources close to where the data is generated. In recent years, several studies in the literature have proposed leveraging these new resources, for instance: i) Real-time video streaming analysis for applications such as traffic control, surveillance, security, and real-time object detection [24, 25, 26]; ii) Smart vehicular systems for pedestrian recognition and crash detection [8, 27]; and iii) Smart cities for energy management and fault detection [7, 28].

Machine Learning for ECC machine learning can be exploited to optimize the deployment of services in the ECC, since leveraging its potential in real-life scenarios is a non-trivial task. The devices are heterogeneous and have multiple constraints (e.g., energy, latency, computational power). For this reason, it is essential to adapt the deployment *dynamically* and *opportunistically*. In the literature, several approaches based on heuristics and meta-heuristics have been proposed [29, 30, 31]; nonetheless, these solutions are crafted for specific use cases and do not adapt over time. In recent years, solutions based on supervised learning techniques [32, 33, 34] have been proposed to predict the characteristics of microservices in order to find solutions that meet various constraints such as quality of service and energy consumption. This approach requires a vast amount of data that represents the various possible states of the system and the possible actions to be taken, which is challenging given the complexity of these systems. For these reasons, reinforcement learning techniques are emerging as a natural approach. This paradigm allows the optimization of the decision-making process by observing the evolution of the system over time, without the initial need to collect a large amount of data. In [35] the authors introduce a solution based on deep reinforcement learning (specifically using a Dueling DQN model [36]) to optimize the deployment of microservices. Initially, the global state of the system is observed (i.e., CPU usage, memory, and network bandwidth). These observations are then passed to the neural network, which predicts end-to-end latency and peak throughput for each possible action, thereby defining a new resource partitioning. The work in [37], instead, proposes a solution based on distributed deep reinforcement learning (using an actor-critic model) to address the task offloading problem, i.e., deciding which available device will execute a certain task. To achieve this, tasks are divided into three categories, namely: i) delay-sensitive tasks; ii) energy-sensitive tasks; and iii) insensitive tasks. At this point, each device runs its actor network to decide whether a task can be executed locally, on an edge server or in the cloud. Finally, a centralized critic network evaluates the actions taken by the various devices, providing feedback on their effectiveness. In addition to the aforementioned studies, other works leverage reinforcement learning for various optimization tasks in the ECC. For instance, [38] employs offline RL for task scheduling, [39] uses RL to optimize data caching in edge servers, and [40] applies RL to enhance network resource allocation.

3. Intelligent Collective Services for ECC

The pulverization model proposes logical systems in which a series of logical entities form a *dynamic graph* that exposes the following properties:

- *large scale*: an ECC system can be extensively scaled to encompass thousands of devices;
- *locality*: edge devices are spatially located, and their behavior may depend on their location;
- *partial observability*: edge devices do not have a perception of the entire collective but can perceive a certain neighborhood or aggregated information;
- *heterogeneity*: devices belonging to ECC are highly heterogeneous due to their diverse nature.

Applying standard supervised learning techniques to ECC services is challenging because it is difficult to determine the correct behavior a priori, given the highly dynamic nature of these systems. Therefore, considering these properties, we argue that a combination of *many-agent reinforcement learning* [9] (to encode large-scale dynamics), *graph neural networks* [41] (to encode spatial relationships), and *aggregate computing* (to encode collective feedback and observations) can be a suitable approach for developing intelligent services for ECC. In particular, many-agent reinforcement learning plays a crucial role in ECC, as it enables scalable policy learning and influences behavior through delayed collective feedback.

In the following, we introduce both many-agent reinforcement learning and graph neural networks, and then discuss the proposed algorithm for developing intelligent services for ECC.

3.1. Formalization

Many-Agent Reinforcement Learning is an extension of reinforcement learning in which, instead of having a single learning agent, there is an entire family of intelligent agents that learn concurrently. They are referred to as "many-agent" to distinguish them from "multi-agent", as the number of devices can be *extremely* large, potentially involving thousands of devices. Formally, a many-agent system can be modeled through a family of agent $\mathbb{A} = (\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{R}, \pi)$ that lives and interact in an environment $\mathcal{E} = (\mathcal{P}, \mathbb{A}, \mathcal{T}, \xi)$ [42]. Particularly, the agent $\mathbb{A}$ is defined by:

- $\mathcal{S}$, $\mathcal{O}$, and $\mathcal{A}$ represent the sets of local states, observations, and actions, respectively.
- $\mathcal{R} : \mathcal{S} \to \mathbb{R}$ is the reward function, influenced by the environment.
- $\pi : \mathcal{O} \to \mathcal{A}$ is the policy mapping observations to actions, which can be deterministic or stochastic.

Where the environment $\mathcal{E}$ is defined by:

- $\mathcal{P}$ is the fixed population of agents.

- $\mathbb{A}$ is the agent prototype governing each agent in $\mathcal{P}$.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A}^{\mathcal{P}} \times \mathcal{S} \to \mathbb{R}^{\mathcal{P}}$ is the global transition function influenced by agent actions, yielding a collective reward.
- $\xi : \mathcal{S} \to \mathcal{O}^{\mathcal{P}}$ is the global observation model.

The system's evolution over time can be captured as follows:

$$\mathcal{A}_t^{\mathcal{P}} = \pi(\xi(\mathcal{S}_t)), \quad \mathcal{S}_{t+1} = \mathcal{T}(\mathcal{S}_t, \mathcal{A}_t)$$

At any given time t, the system can be represented as a graph $G^t = (V^t, E^t)$, where E^t is constructed from a neighborhood relationship. Each node has an associated local observation $o_t^v \in \mathcal{O}$. This graph representation is crucial for both computing computational fields and serving as input for a GNN, as demonstrated in previous works [43, 44, 45, 21].

Graph Neural Network is a novel neural network model designed to process graph-structured data [41]. Given a graph $G = (V, E)$, where V is the set of nodes and $E \subseteq V \times V$ represents the edges, each node $v \in V$ has an associated feature set f_v. The aim of a GNN is to learn a node embedding h_v for each node $v \in V$ by aggregating information from its neighbors $\mathcal{N}_G(v)$ through a process known as *message passing* [46].

The process involves three main steps in each layer k of the GNN:

$$\mathbf{m}_{uv}^{(k)} = \psi^{(k)} \left(\mathbf{h}_u^{(k-1)}, \mathbf{h}_v^{(k-1)}, e_{uv} \right) \tag{1}$$

$$\mathbf{a}_u^{(k)} = \bigoplus^{(k)} \left(\{ \mathbf{m}_{uv}^{(k)} : v \in \mathcal{N}_G(u) \} \right) \tag{2}$$

$$\mathbf{h}_u^{(k)} = \phi^{(k)} \left(\mathbf{h}_u^{(k-1)}, \mathbf{a}_u^{(k)} \right) \tag{3}$$

where $\mathbf{m}_{uv}^{(k)}$ is the message from node u to node v at layer k, $\bigoplus$ denotes the aggregation function, and $\phi^{(k)}$ is the update function. Initially, h_v^0 is set to the node's feature vector f_v. GNNS enables the effective processing of graph-structured data by iteratively updating node embeddings, capturing the local graph structure around each node. This formulation facilitates the application of GNNs in various tasks within the ECC, where the spatial relationships between devices are critical.

$$GNN(G_f) = \{ \mathbf{h}_u^{(k)} : u \in V, k \in \mathbb{N} \} \tag{4}$$

Where G_f is the graph with features f_v for each node $v \in V$, formally: $G_f = (V, E, \{f_v : v \in V\})$. By leveraging the unique capabilities of GNNs, it is possible to develop more sophisticated and adaptive services that can dynamically respond to the complex and distributed nature of the ECC.

3.2. Contribution

In this section, we discuss an approach for performing many-agent reinforcement learning leveraging GNN in pulverized systems. In this context, as in similar many-agent scenarios, we apply the pattern known as *centralized learning and distributed execution* [47]. This is because, at a large scale and with only partial observability, having a global view of the task helps in better learning collective tasks.

Learning dynamics We base our approach on the model of pulverization, learning under the assumption that there exists a set of components that can act according to local logics – here, we are not concerned with the type of actions. Specifically, as described in Algorithm 1, for each time step t, a graph G^t represents the connectivity between devices. Each node in the graph corresponds to a local device, and edges represent the communication or influence between devices. Each node $i \in G^t$, has an associated observation o_i^t creating a decorated graph G_o^t that represents. This same graph can be passed to a macro program P (e.g., aggregate computing) to encode global system information (e.g., areas of high consumption in the case of energy distribution, areas of the system more allocated for task distribution). The macro program aggregates local information to provide a global perspective, which is crucial for the GNN to make informed decisions. Formally, an evaluation of a macro program P with a graph G_o^t produces a new graph G_m^t, where for each node, associate the previous local state and the evolution of the macro program: $m_i^t = (o_i^t, P(G_o^t))$. This graph is then passed to a GNN that will compute the actions $\mathbf{a}_i^t \in \mathcal{A}^{\mathcal{P}t}$ to be taken. These actions will be executed in the environment (i.e., the various logical components will execute what they need to), transitioning to a new collective graph G_o^{t+1} and obtaining a reinforcement signal $\mathbb{R}^{\mathcal{P}}$. The reinforcement signal $\mathbb{R}^{\mathcal{P}t}$ can be split into local and global components. The local component is derived from individual device metrics (e.g., battery consumption), while the global component is computed from aggregate metrics (e.g., average consumption in a region) that can be also computed by another macro program $P^{\mathbb{R}}$.

Algorithm 1 GNN-Based many-agent reinforcement learning in ECC

1: **Initialize:** Local device observations $\mathbf{o}^0$
2: **for** each time step t **do**
3: **Create decorated graph** G_o^t with observations $\mathbf{o}^t$
4: **Apply** macro program P to G_o^t to produce aggregate observations G_m^t
5: **Input** G_m^t into GNN to compute actions $\mathbf{a}_i^t \in \mathcal{A}_i^t$
6: **Execute** actions $\mathbf{a}^t$ in the environment
7: **Transition** to new graph G_o^{t+1} and obtain reinforcement signal $\mathbb{R}^t$
8: **Calculate** local and collective reinforcement signals
9: **Update** GNN with new graph G_o^{t+1}, observations $\mathbf{o}^{t+1}$, and reinforcement signals $\mathbb{R}^t$
10: **end for**

Implementation discussion This section proposes some hints for the future concrete implementation of the algorithm described above and in Algorithm 1. First, it is worth noting

that the proposed solution is completely agnostic to the choosen many-agent reinforcement learning algorithm. The only constraint is that such algorithm must support the centralized learning and distributed execution model. On the one hand, a classic approach is to implement the DQN algorithm [48] to approximate a function that, for each observation, provides the most valuable action (i.e., value based methods). On the other hand, policy based algorithms exists. For instance, PPO [49] is based on a simple actor and critic method. In this framework, each agent has an actor network which is used to select the current action, while a central entity has a critic network used to reward each action. Thus, this method fits the centralized learning and distributed execution contraint. Moreover, a study shows the effectiveness of PPO in multi-agent cooperative settings [50]. To avoid issues, such as the curse of dimensionality and the exponential growth of interactions between agents, related to the large number of agents present in the reference context, the described approaches can be integrated with modern solutions that make learning more stable. For example, mean-field reinforcement learning [51] integrates both DQN and actor-critic approaches by ensuring that each agent is influenced not by the individual actions of neighbors but by their average. Additionally, also the GNN can be distributed in the system in such a way that it can be executed in a distributed manner without the need for a central cloud [43]. Furthermore, thanks to the new self-organizing approach of the self-organising coordination regions [52] pattern, these learning points are not necessarily known a priori and can also change over time, therefore enabling *continual learning* [53]. This distributed approach is further enhanced by self-organizing patterns, allowing for dynamic adaptation and continual learning in changing environments.

4. Intelligent Pulverized System

In this section, we explore how the concept of intelligent collective services can be applied to various levels of pulverization, building upon the proposed model to enhance system efficiency and adaptability. In particular, in the following, we focus on three main aspects: (i) *reconfiguration*: how reconfiguration policies can be used to optimize the deployment of pulverized components at runtime; (ii) *communication*: how communication protocols can be learned to optimize the exchange of information between devices; and (iii) *scheduling*: how adaptive scheduling policies can be used to optimize the execution of pulverized components.

4.1. Reconfiguration

Discussion In the ECC, reconfiguration is a key component. Specifically, with pulverization, reconfiguration allows for the offloading of one or more components of a logical device (right side of Figure 1) from one host to another to meet certain constraints. These constraints can be expressed through reconfiguration rules, which can be either local (e.g., the host's battery drops below a certain threshold) or global (e.g., defined via aggregate computing) to preserve global coherence and prevent oscillatory conditions. While these approaches can be effective in simple scenarios, they have limitations in more complex situations: (i) they can only represent relatively simple rules; (ii) it is challenging to define all rules a priori, as they can be numerous and complex; and (iii) the rules are fixed, so if the system changes over time, it cannot adapt.

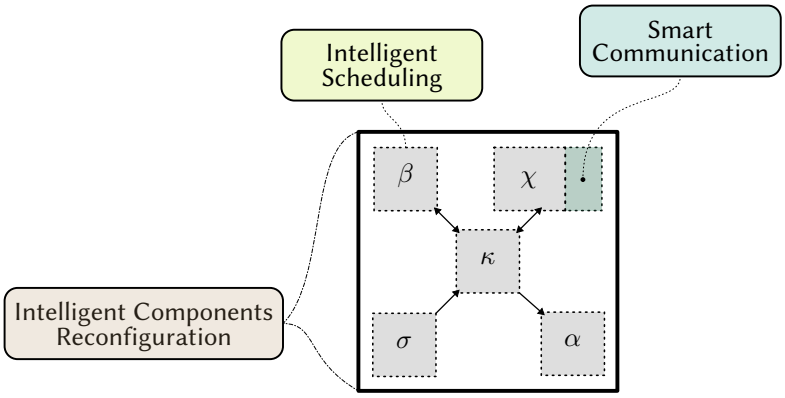

Figure 2: Representation of the main parts in which intelligence can support the pulverization model. *Intelligent Scheduling* manage the intelligent execution of the behavior, *Smart Communication* supports the χ component in efficient communication and neighborhood definition, and *Intelligent Components Reconfiguration* manage the Intelligent and opportunistic relocation of the pulverized components, improving the system's performance.

Vision we believe that an interesting direction is to learn these rules through the proposed many-agent reinforcement learning approach. Specifically, using online reinforcement learning techniques can help manage potential changes or specific cases that were not captured beforehand. In the literature, some works have attempted to explore this field [35, 54, 55]. However, they focus on entire microservices. Our approach offers the following advantages:

1. it is possible to operate more granular, not being forced to relocate the entire logical device, but rather being able to act at the level of individual components;
2. using macroprogramming and graph neural networks, it is possible to integrate neighborhood information, thereby enriching the knowledge of individual devices regarding the global state and objective of the system;
3. it is possible to define complex reconfiguration rules and to learn new rules not known at design time.

4.2. Communication

Discussion in collective systems, communication represents a crucial aspect: thanks to device communication and coordination, a global common goal can be achieved. In this sense, the χ component of the pulverization model is responsible for managing the neighborhood-based communication between devices. For this reason, understanding *what*, *when*, and *with whom* communicate is crucial for effective and efficient coordination. Consequently, as shown in Figure 2, supporting the χ component with intelligent communication can be a key aspect to improve the system performance.

Learning communication protocols is a well-known problem in the literature. Initially, approaches based on heuristics and meta-heuristics were proposed [56, 57]. However, they often fail to guarantee scalability and adaptability over time, which are essential qualities for systems

operating in dynamic and large-scale environments. To overcome these limitations, there has been a significant shift towards utilizing multi-agent reinforcement learning techniques. MARL has shown promise in addressing the complexities associated with learning and optimizing communication protocols among multiple agents [58, 59]. In addition to MARL, Graph Neural Networks have been employed to capture relationships between neighboring agents [60, 61].

Vision we believe the proposed many-agent reinforcement learning approach, combined with GNNs, holds great promise for learning communication protocols in the context of pulverized systems. This approach allows for:

1. optimizing the amount of information exchanged in the network;
2. optimizing the set of neighbors to which a message is sent at a given time step;
3. optimizing the message exchange frequency.

Thus, avoiding large bandwidth consumption and energy waste for individual devices.

4.3. Scheduling

Discussion in the pulverization model, no fixed scheduling policy is generally predefined for calculating the new state (i.e., executing the behavior). This approach allows developers considerable flexibility in choosing how and when to execute the primary behavior. The scheduling policy must be adaptable to the underlying infrastructure layer. For instance, when computations are entirely offloaded to the cloud, there are no significant power constraints. In contrast, when computations are performed on mobile devices like smartphones, power consumption becomes a critical factor.

Beyond simple local rules (e.g., reducing evaluation frequency when battery power is low), scheduling may also be influenced by the collective results of ongoing computations. For example, in a crowd scenario, if the situation becomes less crowded, the overall frequency of evaluations may be reduced. Early work in this area, such as programmable distributed scheduling leveraging aggregate computing [62], involved defining scheduling rules based on the collective outcome of computations. However, these approaches assumed a fixed scheduling policy that did not adapt to changing environmental dynamics. Recent studies [23] have shown that simple variations of Q-learning can improve collective computation by converging more rapidly to collective structures. However, these methods relied on centralized learning and Q-tables, making them unsuitable for scaling to the complexity of diverse scenarios.

Vision our vision of embracing many-agent reinforcement learning and graph networks offers several advantages:

1. creating distributed global scheduling independent of the number of devices;
2. developing distributed policies influenced by global information or larger neighborhoods;
3. enabling more complex scheduling functions using neural networks instead of simple tables.

We believe this is a promising direction, as recent work has successfully applied this approach to specific task allocation problems [63]. Applying it to the pulverization model would enable its use in various collaborative scenarios, opening the possibility of creating services that intelligently optimize certain Quality of Service (QoS) metrics based on the collective layer proposed by the model.

4.4. Applicability

Our approach comprehensively addresses the challenge of deploying and dynamically reconfiguring applications in the ECC injecting intelligence in these processes (cf. Section 4). Typically, application scenarios in the context of the IoT, edge computing, and swarm robotics can take advantages by their deployment in infrastructures like the ECC, improving both functional and non-functional aspects.

For instance, in smart city and wearable technology scenarios, our system offers a dual benefit: extending battery life and minimizing the overall power consumption of deployed systems. Intelligent reconfiguration policies can be automatically devised and inferred to better relocate the component's execution over the infrastructure, extending the device's battery life and reducing the CO_2e of the system, otherwise difficult or even impossible with traditional approaches like optimization algorithm or rule-based systems (cf. Section 4.1). Similarly, determining the optimal execution policy for each component involves a complex interplay of factors, including the target device, environmental conditions, and specific application constraints. All these aspects may (and usually do) change frequently over time making it difficult to predict a priori. In a rescue scenario, our intelligent system can dynamically adapt to the emergency by increasing the computational frequency of the nearby devices, providing more updated status about the emergency conditions, while reducing the computational frequency on the farthest devices not close to the hot spot (cf. Section 4.3). Additionally, via the proposed approach, intelligent communication patterns can be automatically envisioned determining *where* the communication should occur (i.e., the physical place where the communication is implemented), and *who* this communication should involve to effectively transfer the minimal but effective information for the rescuers to promptly intervene in the emergency area (cf. Section 4.2).

5. Conclusion

In this paper, we discussed a vision for creating and deploying intelligent services within the Edge-Cloud Continuum (ECC). Specifically, we proposed an idea called intelligent pulverized systems (IPS), which combines the pulverization model with graph-based networks and many-agent reinforcement learning. This solution possesses the necessary characteristics to adapt to modern ECCs, including scalability, the ability to encode collective information, and the definition of components in an architecture-agnostic manner. We believe that this approach is essential for maximizing the potential of ECCs to create opportunistic and intelligent applications.

In the near future, we plan to leverage state-of-the-art many-agent (like mean-field reinforcement learning [51]) algorithms to effectively validate this approach in real-world scenarios,

such as smart cities and beyond. By doing so, we aim to demonstrate the practical applicability and benefits of ISP in dynamically changing environments.

References

[1] D. Khalyeyev, T. Bures, P. Hnetynka, Towards characterization of edge-cloud continuum, CoRR abs/2309.05416 (2023). URL: https://doi.org/10.48550/arXiv.2309.05416. doi:10.48550/ARXIV.2309.05416. arXiv:2309.05416.

[2] S. Moreschini, F. Pecorelli, X. Li, S. Naz, D. Hästbacka, D. Taibi, Cloud continuum: The definition, IEEE Access 10 (2022) 131876–131886. URL: https://doi.org/10.1109/ACCESS.2022.3229185. doi:10.1109/ACCESS.2022.3229185.

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

[4] M. Satyanarayanan, Pervasive computing: vision and challenges, IEEE Wirel. Commun. 8 (2001) 10–17. URL: https://doi.org/10.1109/98.943998. doi:10.1109/98.943998.

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

[6] M. Friedewald, O. Raabe, Ubiquitous computing: An overview of technology impacts, Telematics Informatics 28 (2011) 55–65. URL: https://doi.org/10.1016/j.tele.2010.09.001. doi:10.1016/J.TELE.2010.09.001.

[7] D. Park, S. Kim, Y. An, J. Jung, Lired: A light-weight real-time fault detection system for edge computing using LSTM recurrent neural networks, Sensors 18 (2018) 2110. URL: https://doi.org/10.3390/s18072110. doi:10.3390/S18072110.

[8] W. Chang, L. Chen, K. Su, Deepcrash: A deep learning-based internet of vehicles system for head-on and single-vehicle accident detection with emergency notification, IEEE Access 7 (2019) 148163–148175. URL: https://doi.org/10.1109/ACCESS.2019.2946468. doi:10.1109/ACCESS.2019.2946468.

[9] Y. Yang, Many-agent reinforcement learning, Ph.D. thesis, University College London (University of London), UK, 2021. URL: https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.830054.

[10] R. Casadei, D. Pianini, A. Placuzzi, M. Viroli, D. Weyns, Pulverization in cyber-physical systems: Engineering the self-organizing logic separated from deployment, Future Internet 12 (2020) 203. URL: https://doi.org/10.3390/fi12110203. doi:10.3390/FI12110203.

[11] R. Casadei, G. Fortino, D. Pianini, A. Placuzzi, C. Savaglio, M. Viroli, A methodology and simulation-based toolchain for estimating deployment performance of smart collective services at the edge, IEEE Internet Things J. 9 (2022) 20136–20148. URL: https://doi.org/10.1109/JIOT.2022.3172470. doi:10.1109/JIOT.2022.3172470.

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

[13] S. Sharma, V. Chang, U. S. Tim, J. Wong, S. K. Gadia, Cloud and iot-based emerging services

systems, Clust. Comput. 22 (2019) 71–91. URL: https://doi.org/10.1007/s10586-018-2821-8. doi:10.1007/S10586-018-2821-8.

[14] A. Taherkordi, F. Eliassen, Towards independent in-cloud evolution of cyber-physical systems, in: 2014 IEEE International Conference on Cyber-Physical Systems, Networks, and Applications, CPSNA 2014, Hong Kong, China, August 25-26, 2014, IEEE Computer Society, 2014, pp. 19–24. URL: https://doi.org/10.1109/CPSNA.2014.12. doi:10.1109/CPSNA.2014.12.

[15] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, A. Ahmed, Edge computing: A survey, Future Gener. Comput. Syst. 97 (2019) 219–235. URL: https://doi.org/10.1016/j.future.2019.02.050. doi:10.1016/J.FUTURE.2019.02.050.

[16] R. Newton, M. Welsh, Region streams: functional macroprogramming for sensor networks, in: A. Labrinidis, S. Madden (Eds.), Proceedings of the 1st Workshop on Data Management for Sensor Networks, in conjunction with VLDB, DMSN 2004, Toronto, Canada, August 30, 2004, volume 72 of *ACM International Conference Proceeding Series*, ACM, 2004, pp. 78–87. URL: https://doi.org/10.1145/1052199.1052213. doi:10.1145/1052199.1052213.

[17] R. Casadei, G. Fortino, D. Pianini, W. Russo, C. Savaglio, M. Viroli, A development approach for collective opportunistic edge-of-things services, Inf. Sci. 498 (2019) 154–169. URL: https://doi.org/10.1016/j.ins.2019.05.058. doi:10.1016/J.INS.2019.05.058.

[18] A. Azzara, D. Alessandrelli, S. Bocchino, M. Petracca, P. Pagano, Pyot, a macroprogramming framework for the internet of things, in: Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems, SIES 2014, Pisa, Italy, June 18-20, 2014, IEEE, 2014, pp. 96–103. URL: https://doi.org/10.1109/SIES.2014.6871193. doi:10.1109/SIES.2014.6871193.

[19] C. Pinciroli, G. Beltrame, Buzz: A programming language for robot swarms, IEEE Softw. 33 (2016) 97–100. URL: https://doi.org/10.1109/MS.2016.95. doi:10.1109/MS.2016.95.

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

[21] G. Aguzzi, M. Viroli, L. Esterle, Field-informed reinforcement learning of collective tasks with graph neural networks, in: IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2023, Toronto, ON, Canada, September 25-29, 2023, IEEE, 2023, pp. 37–46. URL: https://doi.org/10.1109/ACSOS58161.2023.00021. doi:10.1109/ACSOS58161.2023.00021.

[22] G. Aguzzi, R. Casadei, M. Viroli, Towards reinforcement learning-based aggregate computing, in: M. H. ter Beek, M. Sirjani (Eds.), Coordination Models and Languages - 24th IFIP WG 6.1 International Conference, COORDINATION 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings, volume 13271 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 72–91. URL: https://doi.org/10.1007/978-3-031-08143-9_5. doi:10.1007/978-3-031-08143-9_5.

[23] G. Aguzzi, R. Casadei, M. Viroli, Addressing collective computations efficiency: Towards a platform-level reinforcement learning approach, in: R. Casadei, E. D. Nitto, I. Gerostathopoulos, D. Pianini, I. Dusparic, T. Wood, P. R. Nelson, E. Pournaras, N. Bencomo, S. Götz, C. Krupitzer, C. Raibulet (Eds.), IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2022, Virtual, CA, USA, September 19-

23, 2022, IEEE, 2022, pp. 11–20. URL: https://doi.org/10.1109/ACSOS55765.2022.00019. doi:10.1109/ACSOS55765.2022.00019.

[24] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, S. Sinha, Real-time video analytics: The killer app for edge computing, Computer 50 (2017) 58–67. URL: https://doi.org/10.1109/MC.2017.3641638. doi:10.1109/MC.2017.3641638.

[25] G. Kar, S. Jain, M. Gruteser, F. Bai, R. Govindan, Real-time traffic estimation at vehicular edge nodes, in: J. Zhang, M. Chiang, B. M. Maggs (Eds.), Proceedings of the Second ACM/IEEE Symposium on Edge Computing, San Jose / Silicon Valley, SEC 2017, CA, USA, October 12-14, 2017, ACM, 2017, pp. 3:1–3:13. URL: https://doi.org/10.1145/3132211.3134461. doi:10.1145/3132211.3134461.

[26] S. Tuli, N. Basumatary, R. Buyya, Edgelens: Deep learning based object detection in integrated iot, fog and cloud computing environments, CoRR abs/1906.11056 (2019). URL: http://arxiv.org/abs/1906.11056. arXiv:1906.11056.

[27] P. J. Navarro, C. Fernández-Isla, R. Borraz, D. Alonso, A machine learning approach to pedestrian detection for autonomous vehicles using high-definition 3d range data, Sensors 17 (2017) 18. URL: https://doi.org/10.3390/s17010018. doi:10.3390/S17010018.

[28] X. Chang, W. Li, C. Xia, J. Ma, J. Cao, S. U. Khan, A. Y. Zomaya, From insight to impact: Building a sustainable edge computing platform for smart homes, in: 24th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2018, Singapore, December 11-13, 2018, IEEE, 2018, pp. 928–936. URL: https://doi.org/10.1109/PADSW.2018.8644647. doi:10.1109/PADSW.2018.8644647.

[29] I. Kovacevic, E. Harjula, S. Glisic, B. Lorenzo, M. Ylianttila, Cloud and edge computation offloading for latency limited services, IEEE Access 9 (2021) 55764–55776. URL: https://doi.org/10.1109/ACCESS.2021.3071848. doi:10.1109/ACCESS.2021.3071848.

[30] J. Kwak, Y. Kim, J. Lee, S. Chong, DREAM: dynamic resource and task allocation for energy minimization in mobile cloud systems, IEEE J. Sel. Areas Commun. 33 (2015) 2510–2523. URL: https://doi.org/10.1109/JSAC.2015.2478718. doi:10.1109/JSAC.2015.2478718.

[31] S. Wang, M. Zafer, K. K. Leung, Online placement of multi-component applications in edge computing environments, IEEE Access 5 (2017) 2514–2533. URL: https://doi.org/10.1109/ACCESS.2017.2665971. doi:10.1109/ACCESS.2017.2665971.

[32] X. Hou, C. Li, J. Liu, L. Zhang, Y. Hu, M. Guo, Ant-man: towards agile power management in the microservice era, in: C. Cuicchi, I. Qualters, W. T. Kramer (Eds.), Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020, IEEE/ACM, 2020, p. 78. URL: https://doi.org/10.1109/SC41405.2020.00082. doi:10.1109/SC41405.2020.00082.

[33] Y. Gan, Y. Zhang, K. Hu, D. Cheng, Y. He, M. Pancholi, C. Delimitrou, Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices, in: I. Bahar, M. Herlihy, E. Witchel, A. R. Lebeck (Eds.), Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019, ACM, 2019, pp. 19–33. URL: https://doi.org/10.1145/3297858.3304004. doi:10.1145/3297858.3304004.

[34] Q. Chen, H. Yang, M. Guo, R. S. Kannan, J. Mars, L. Tang, Prophet: Precise qos prediction on non-preemptive accelerators to improve utilization in warehouse-scale computers,

in: Y. Chen, O. Temam, J. Carter (Eds.), Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2017, Xi'an, China, April 8-12, 2017, ACM, 2017, pp. 17–32. URL: https://doi.org/10.1145/3037697.3037700. doi:10.1145/3037697.3037700.

[35] K. Fu, W. Zhang, Q. Chen, D. Zeng, M. Guo, Adaptive resource efficient microservice deployment in cloud-edge continuum, IEEE Trans. Parallel Distributed Syst. 33 (2022) 1825–1840. URL: https://doi.org/10.1109/TPDS.2021.3128037. doi:10.1109/TPDS.2021.3128037.

[36] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, N. de Freitas, Dueling network architectures for deep reinforcement learning, in: M. Balcan, K. Q. Weinberger (Eds.), Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016, volume 48 of *JMLR Workshop and Conference Proceedings*, JMLR.org, 2016, pp. 1995–2003. URL: http://proceedings.mlr.press/v48/wangf16.html.

[37] G. Nieto, I. de la Iglesia, U. Lopez-Novoa, C. Perfecto, Deep reinforcement learning techniques for dynamic task offloading in the 5g edge-cloud continuum, J. Cloud Comput. 13 (2024) 94. URL: https://doi.org/10.1186/s13677-024-00658-0. doi:10.1186/S13677-024-00658-0.

[38] S. Sheng, P. Chen, Z. Chen, L. Wu, Y. Yao, Deep reinforcement learning-based task scheduling in iot edge computing, Sensors 21 (2021) 1666. URL: https://doi.org/10.3390/s21051666. doi:10.3390/S21051666.

[39] Y. Wei, Z. Zhang, F. R. Yu, Z. Han, Joint user scheduling and content caching strategy for mobile edge networks using deep reinforcement learning, in: 2018 IEEE International Conference on Communications Workshops, ICC Workshops 2018, Kansas City, MO, USA, May 20-24, 2018, IEEE, 2018, pp. 1–6. URL: https://doi.org/10.1109/ICCW.2018.8403711. doi:10.1109/ICCW.2018.8403711.

[40] D. Zeng, L. Gu, S. Pan, J. Cai, S. Guo, Resource management at the network edge: A deep reinforcement learning approach, IEEE Netw. 33 (2019) 26–33. URL: https://doi.org/10.1109/MNET.2019.1800386. doi:10.1109/MNET.2019.1800386.

[41] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P. S. Yu, A comprehensive survey on graph neural networks, IEEE Trans. Neural Networks Learn. Syst. 32 (2021) 4–24. URL: https://doi.org/10.1109/TNNLS.2020.2978386. doi:10.1109/TNNLS.2020.2978386.

[42] X. Yu, W. Wu, P. Feng, Y. Tian, Swarm inverse reinforcement learning for biological systems, in: Y. Huang, L. A. Kurgan, F. Luo, X. Hu, Y. Chen, E. R. Dougherty, A. Kloczkowski, Y. Li (Eds.), IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2021, Houston, TX, USA, December 9-12, 2021, IEEE, 2021, pp. 274–279. URL: https://doi.org/10.1109/BIBM52615.2021.9669656. doi:10.1109/BIBM52615.2021.9669656.

[43] E. I. Tolstaya, F. Gama, J. Paulos, G. J. Pappas, V. Kumar, A. Ribeiro, Learning decentralized controllers for robot swarms with graph neural networks, in: Proc. of the Conf. on Robot Learning, 2019, pp. 671–682.

[44] E. Tolstaya, F. Gama, J. Paulos, G. Pappas, V. Kumar, A. Ribeiro, Learning decentralized controllers for robot swarms with graph neural networks, in: Proc. of the Conf. on Robot Learning, PMLR, 2020, pp. 671–682.

[45] W. Gosrich, S. Mayya, R. Li, J. Paulos, M. Yim, A. Ribeiro, V. Kumar, Coverage control in multi-robot systems via graph neural networks, in: Proc. of the Int. Conf. on Robotics and

Automation, IEEE, 2022, pp. 8787–8793. doi:10.1109/ICRA46639.2022.9811854.

[46] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, G. E. Dahl, Neural message passing for quantum chemistry, in: Proc. of the 34th International Conference on Machine Learning, 2017, pp. 1263–1272.

[47] P. K. Sharma, R. Fernandez, E. Zaroukian, M. Dorothy, A. Basak, D. E. Asher, Survey of recent multi-agent reinforcement learning algorithms utilizing centralized training, arXiv preprint arXiv:2107.14316 (2021). URL: https://arxiv.org/abs/2107.14316.

[48] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. A. Riedmiller, Playing atari with deep reinforcement learning, CoRR abs/1312.5602 (2013). URL: http://arxiv.org/abs/1312.5602. arXiv:1312.5602.

[49] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, CoRR abs/1707.06347 (2017). URL: http://arxiv.org/abs/1707.06347. arXiv:1707.06347.

[50] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. M. Bayen, Y. Wu, The surprising effectiveness of PPO in cooperative multi-agent games, in: S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh (Eds.), Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022, 2022. URL: http://papers.nips.cc/paper_files/paper/2022/hash/9c1535a02f0ce079433344e14d910597-Abstract-Datasets_and_Benchmarks.html.

[51] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, J. Wang, Mean field multi-agent reinforcement learning, in: J. G. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, volume 80 of *Proceedings of Machine Learning Research*, PMLR, 2018, pp. 5567–5576. URL: http://proceedings.mlr.press/v80/yang18d.html.

[52] R. Casadei, D. Pianini, M. Viroli, A. Natali, Self-organising coordination regions: A pattern for edge computing, in: H. R. Nielson, E. Tuosto (Eds.), Coordination Models and Languages - 21st IFIP WG 6.1 International Conference, COORDINATION 2019, Held as Part of the 14th International Federated Conference on Distributed Computing Techniques, DisCoTec 2019, Kongens Lyngby, Denmark, June 17-21, 2019, Proceedings, volume 11533 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 182–199. URL: https://doi.org/10.1007/978-3-030-22397-7_11. doi:10.1007/978-3-030-22397-7_11.

[53] K. Khetarpal, M. Riemer, I. Rish, D. Precup, Towards continual reinforcement learning: A review and perspectives, J. Artif. Intell. Res. 75 (2022) 1401–1476. URL: https://doi.org/10.1613/jair.1.13673. doi:10.1613/JAIR.1.13673.

[54] Y. Zhang, W. Hua, Z. Zhou, G. E. Suh, C. Delimitrou, Sinan: Ml-based and qos-aware resource management for cloud microservices, in: T. Sherwood, E. D. Berger, C. Kozyrakis (Eds.), ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual Event, USA, April 19-23, 2021, ACM, 2021, pp. 167–181. URL: https://doi.org/10.1145/3445814.3446693. doi:10.1145/3445814.3446693.

[55] Y. Gan, M. Liang, S. Dev, D. Lo, C. Delimitrou, Sage: practical and scalable ml-driven performance debugging in microservices, in: T. Sherwood, E. D. Berger, C. Kozyrakis (Eds.), ASPLOS '21: 26th ACM International Conference on Architectural Support for

Programming Languages and Operating Systems, Virtual Event, USA, April 19-23, 2021, ACM, 2021, pp. 135–151. URL: https://doi.org/10.1145/3445814.3446700. doi:10.1145/3445814.3446700.

[56] C. V. Goldman, J. S. Rosenschein, Emergent coordination through the use of cooperative state-changing rules, in: B. Hayes-Roth, R. E. Korf (Eds.), Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1, AAAI Press / The MIT Press, 1994, pp. 408–413. URL: http://www.aaai.org/Library/AAAI/1994/aaai94-062.php.

[57] C. V. Goldman, S. Zilberstein, Optimizing information exchange in cooperative multi-agent systems, in: The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings, ACM, 2003, pp. 137–144. URL: https://doi.org/10.1145/860575.860598. doi:10.1145/860575.860598.

[58] C. Zhu, M. Dastani, S. Wang, A survey of multi-agent deep reinforcement learning with communication, Auton. Agents Multi Agent Syst. 38 (2024) 4. URL: https://doi.org/10.1007/s10458-023-09633-6. doi:10.1007/S10458-023-09633-6.

[59] J. N. Foerster, Y. M. Assael, N. de Freitas, S. Whiteson, Learning to communicate with deep multi-agent reinforcement learning, in: D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, R. Garnett (Eds.), Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain, 2016, pp. 2137–2145. URL: https://proceedings.neurips.cc/paper/2016/hash/c7635bfd99248a2cdef8249ef7bfbef4-Abstract.html.

[60] A. Agarwal, S. Kumar, K. P. Sycara, M. Lewis, Learning transferable cooperative behavior in multi-agent teams, in: A. E. F. Seghrouchni, G. Sukthankar, B. An, N. Yorke-Smith (Eds.), Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020, International Foundation for Autonomous Agents and Multiagent Systems, 2020, pp. 1741–1743. URL: https://dl.acm.org/doi/10.5555/3398761.3398967. doi:10.5555/3398761.3398967.

[61] J. Jiang, C. Dun, T. Huang, Z. Lu, Graph convolutional reinforcement learning, in: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, OpenReview.net, 2020. URL: https://openreview.net/forum?id=HkxdQkSYDB.

[62] D. Pianini, R. Casadei, M. Viroli, S. Mariani, F. Zambonelli, Time-fluid field-based coordination through programmable distributed schedulers, Log. Methods Comput. Sci. 17 (2021). URL: https://doi.org/10.46298/lmcs-17(4:13)2021. doi:10.46298/LMCS-17(4:13)2021.

[63] C. Jian, Z. Pan, L. Bao, M. Zhang, Online-learning task scheduling with gnn-rl scheduler in collaborative edge computing, Cluster Computing 27 (2023) 589–605. URL: http://dx.doi.org/10.1007/s10586-022-03957-w. doi:10.1007/s10586-022-03957-w.