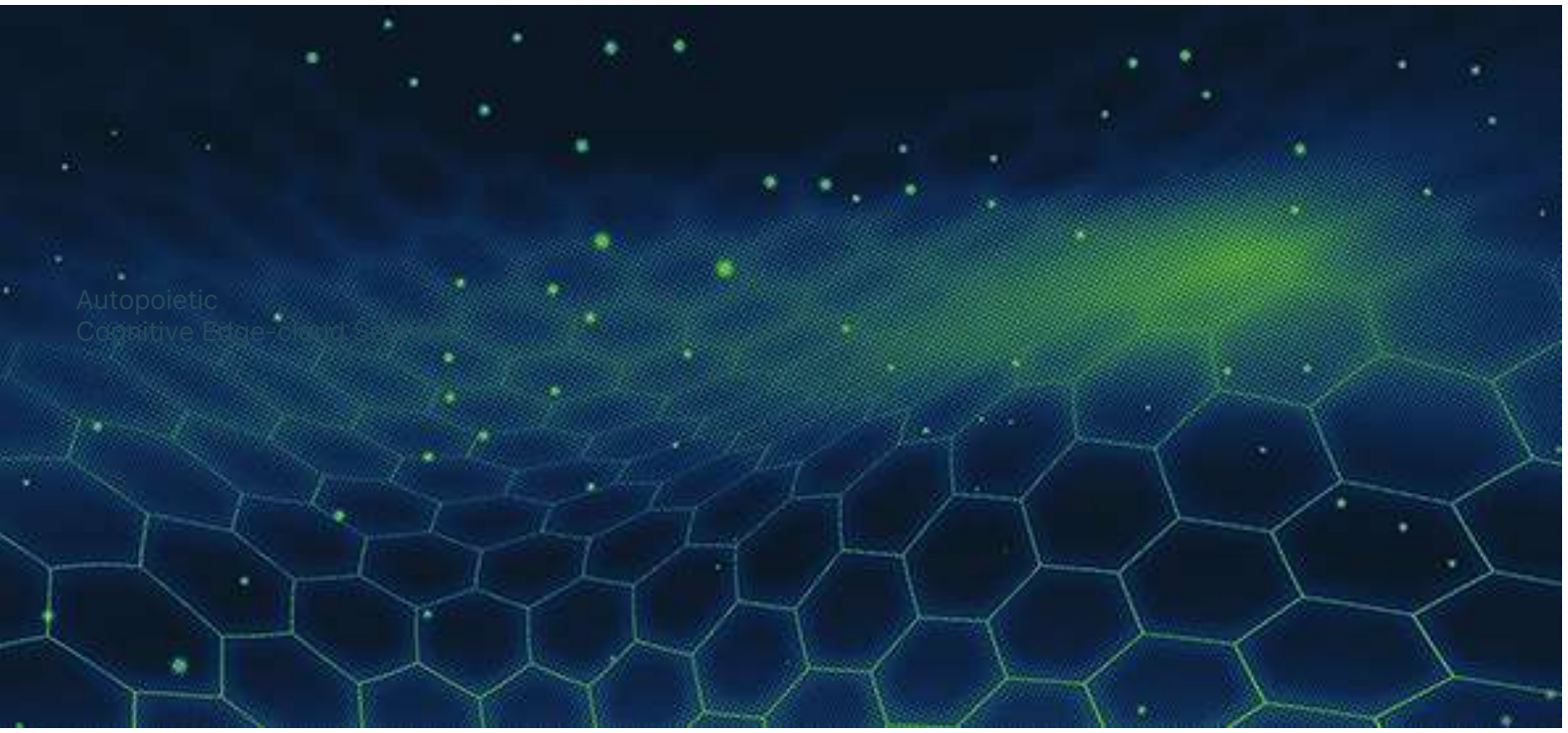

Deliverable 4.2
Action Language and Library

Grant Agreement Number: 101093126

hiro MICRODATACENTERS

inesc id lisboa

ipto

sixsq

Lakeside Labs
SELF-ORGANIZING NETWORKED SYSTEMS

POLITÉCNICA

MARTEL Innovate

TECHNISCHE UNIVERSITÄT DARMSTADT

University of Ljubljana
Faculty of Computer and Information Science

IDSIA SUPSI

DATAPOWER CONSULTING

Autopoietic Cognitive Edge-cloud Services	
Project full title	Autopoietic Cognitive Edge-cloud Services
Call identifier	HORIZON-CL4-2022-DATA-01
Type of action	RIA
Start date	01/01/2023
End date	31/12/2025
Grant agreement no	101093126

Funding of associated partners
The Swiss associated partners of the ACES project were funded by the Swiss State Secretariat for Education, Research and Innovation (SERI).

D4.2 – Action Language and Library	
Author(s)	Felix Cuadrado, Hugo Parada, Melanie Schranz, Loris Canelli, Fernando Ramos, Cláudio Correia, Luis Rodrigues, Thien Duc Nguyen
Editor	Felix Cuadrado
Participating partners	UPM, UL, LAKE, INESC-ID, HIRO, IDSIA, TUD
Version	2.0
Status	Completed
Deliverable date	M12
Dissemination Lvl	PU - Public
Official date	31 December 2023
Actual date	23 December 2023

Executive Summary

Deliverable D4.2 – ACES Action Library - of the Horizon Europe ACES project outlines the technical achievements of the first year from its WP4, focussed on the definition of actions that will be automatically executed by reasoning agents in order to provide the ACES platform of emerging self-management properties.

We have established over this year the ACES view of autopoiesis, presenting a novel interpretation of autopoietic principles, originally a biological concept, to the challenges of managing edge-cloud continuum systems. The ACES approach addresses core systems challenges in a novel manner, pioneering a path for building such platforms that can provide edge-cloud continuum systems of substantially stronger resiliency, and adaptability to a dynamic environment. The document details the main Machine Learning-based approaches prior to this project and discusses some of the intrinsic limitations of these approaches with respect to scalability and adaptability.

A key aspect of the deliverable is the initial version of the ACES action library. This collection of actions provides the first complete definition of the ACES platform from the point of view of reasoning agents; actions define what are the possible control levers that internal management systems can decide to exert to correct or improve the runtime status of the decentralized distributed system. The deliverable looks at this space from two complementary viewpoints: first, potential actions on workload elements, storage components, and networking elements are evaluated. Then, we specifically analyze key non-functional aspects, namely network performance, security, and privacy. Performance objectives have been set to enhance network functions, thereby improving system responsiveness and efficiency. Security and privacy measures include intrusion detection within the network, container security enhancement, and protection against machine learning algorithm attacks.

Another highlight of the work from this first year is the integration of swarm intelligence and machine learning (ML) methods to achieve emerging autopoietic behavior. The deliverable describes our initial approach, effectively combining swarm agent algorithms with ML, while overcoming limitations inherent in these approaches when used independently. Supply and demand swarm agents represent simple units that characterize the core needs and actions, while scaling in their interacting over a large-scale distributed management environment. Several swarm algorithms, such as hormone and ant colony algorithms, have been adapted to suit the environment's needs. Furthermore, the ML component applies Bayesian reasoning techniques to modify swarm hyperparameters, improving performance based on the observed effects of actions.

Overall, Deliverable D4.2 of the ACES project establishes a solid foundation exploring autopoietic systems within the edge-cloud continuum. The proposed set of actions and the hybrid swarm-ML approach enables the ACES platform to leverage collected knowledge, translating it into reasoning components that address key management challenges within the edge-cloud infrastructure.

Disclaimer

The ACES consortium consists of the following partners:

No	PARTNER ORGANISATION NAME	ABBREVIATION	COUNTRY
1	INSTITUTO DE ENGENHARIA DE SISTEMAS E COMPUTADORES, INVESTIGACAO E DESENVOLVIMENTO EM LISBOA	INESC ID	PT
2	HIRO MICRODATACENTERS B.V	HIRO	NL
3	TECHNISCHE UNIVERSITAT DARMSTADT	TUD	DE
4	LAKESIDE LABS GMBH	LAKE	AT
5	UNIVERZA V LJUBLJANI	UL	SI
6	UNIVERSIDAD POLITECNICA DE MADRID	UPM	ES
7	MARTEL GMBH	MAR	CH
8	SCUOLA UNIVERSITARIA PROFESSIONALE DELLA SVIZZERA ITALIANA	IDSIA	CH
9	INDIPENDENT POWER TRANSMISSION OPERATOR SA	IPTO	EL
10	DATAPOWER SRL	DP	IT
11	SIXSQ SA	SIXSQ	CH

Document Revision History

DATE	VERSION	DESCRIPTION	CONTRIBUTIONS
15/09/2023	1.0	Table of contents	UPM
16/10/2023	1.1	Updated structure after Ljubljana workshop discussions	UPM
20/102023	1.2	Updated the agent interactions, inserted candidate swarm algorithms	LAKE
25/10/2023	1.3	Update document structure and integrate framework language and tools from D3.1	UPM, LAKE
31/10/2023	1.5	Updated drafts of several sections	UPM, IDSIA, INESC ID, LAKE
30/11/2023	1.8	Full draft of every section and subsection ready for internal review	UPM, INESC ID, UL
15/12/2023	1.9	Internal review completed. Updated version of the document with all suggestions applied	UPM, LAKE, UL
23/12/2023	2.0	Final review	INESC-ID

Authors

AUTHOR	PARTNER
Felix Cuadrado, Hugo Parada	UPM
Melanie Schranz	LAKE
Loris Canelli	IDSIA
Fernando Ramos, Cláudio Correia, Luis Rodrigues	INESC ID
Thien Duc Nguyen	TUD

Reviewers

NAME	ORGANISATION
Timotej Gale	UL
Melanie Schranz	LAKE

List of terms and abbreviations

ABBREVIATION	DESCRIPTION
ABC	Artificial Bee Colony
AI	Artificial Intelligence
API	Application Programming Interface
CPU	Central Processing Unit
CRD	Custom Resource Definition
CSI	Container Storage Interface
DLRA	Distributed Long Running Application
DNS	Domain Name System
EMDC	Edge Micro Data Centres
FL	Federated Learning
GPS	Global Positioning System
GPU	Graphics Processing Unit
GWO	Grey Wolf Optimizer
HCL	HashiCorp Configuration Language
IDS	Intrusion Detection System
IDW	Inverse Distance Weighting
I/O	Input/Output
IOPS	Input/Output Operations per Second
IP	Internet Protocol
JSON	JavaScript Object Notation
KPI	Key Performance Indicator

LRA	Long Running Applications
ML	Machine Learning
NF	Network Function
NFS	Network File System
NGN	Next-Generation Network
NVMe	Non-volatile Memory Express
PromQL	Prometheus Query Language
PSO	Particle Swarm Optimization
PV	Persistent Volume
PVC	Persistent Volume Claims
QoS	Quality of Service
RAM	Random Access Memory
RBF	Radial Basis Function
RL	Reinforcement Learning
R/W	Read/Write
SLA	Service Level Agreement
SLI	Service Level Indicator
SLO	Service Level Objective
WL	Workload
WOA	Whale Optimization Algorithm
WP	Work Package
XAI	Explainable Artificial Intelligence
XML	Extensible Markup Language

Table of contents

1 Introduction

The ACES project is at the forefront of addressing the unique challenges presented by the dynamic nature of environments in the edge-cloud continuum. These environments are characterized by low latency requirements, heterogeneous hardware, resource constraints, and demand volatility. Therefore, effectively operating this complex runtime platform is highly challenging.

Drawing inspiration from the concept of autopoiesis, the ACES project aims to develop a self-managing architecture with analogous properties, i.e., being capable of autonomously maintaining and renewing itself. This architecture is designed to proactively respond to both external and internal variations, as well as evolving service requirements. In order to achieve these goals, the ACES project will employ reasoning agents capable of deciding on the exertion of a set of potential actions to influence the environment based on their perception and internal knowledge.

This deliverable, primarily focused on the work undertaken in Work Package 4 (WP4) of the ACES project, encapsulates the cumulative efforts of WP4 tasks over the first year of the project. The document expands the definition of multiple components of the ACES architecture, as detailed in Deliverable D2.1 - ACES Architecture Definition, and details how the ACES knowledge and data described in Deliverable D3.1 – ACES Data and Knowledge Model will be used for managing the environment. Collectively, these components form the first comprehensive view of the autopoietic approach employed by ACES to tackle the complexities associated with managing the edge-cloud continuum.

1.1 Approach

This document presents the first iteration of the ACES action library. Starting from the concept of autopoiesis and its potential application to the management of distributed edge and cloud infrastructure, a library of actions has been identified that can be invoked by ACES agents to manage the ACES platform components to improve its operation.

To obtain this first version of the library, the project has performed multiple activities. During the Darmstadt workshop in early May 2023, a blueprint for the ACES system was defined. During this process, a comprehensive set of potential metrics and autopoietic behaviors were identified and collected. The autopoietic behavior, in combination with the definition of a set of core elements and functions of the ACES platform, has led to the identification of a set of potential actions that will be available to achieve the identified runtime and service level objectives.

In parallel to this effort, consortium members have analyzed the state of the start, evaluated and explored potential artificial intelligence (AI), machine learning (ML) and swarm approaches for implementing the identified autopoietic characteristics of ACES. In particular, a unique approach has been identified that combines in a novel way ML methods for parameter optimization with fully decentralized agent swarm algorithms. This deliverable reports the main results of these activities.

1.2 Structure of the document

The present deliverable begins with an introduction to autopoietic systems and their significance to distributed computing, discussing the main works from the state of the art in pursuit of this paradigm. Section 3 follows with an identification of the list of actions that compose the ACES action library, including actions on specific types of runtime resources from the execution platform, and actions that aim to improve the security and network performance of non-functional aspects that are key to the project. Section 4 section expands on the unique hybrid swarm agent with the proposed ML approach researched in the project. The document is closed with a final section presenting the main conclusions and further goals for the remainder of the project.

2 Background

The goal of this section is to provide context for this deliverable and contribute a comprehensive overview of autopoietic systems, related ML approaches and swarm intelligence. We first explore the concept of autopoiesis, detailing its main characteristics according to the literature, and how these ideas have been applied to the management of large-scale distributed systems.

Based on that, we describe the main approaches for achieving these characteristics in the past. We discuss the strong points and limitations of ML-based approaches, to conclude the section with an overview of agent swarm methods, as they present substantial potential to achieve autopoietic goals, and palliate several base limitations found in the literature when using more established ML techniques.

2.1 Autopoietic systems

Autopoiesis, a term of Greek origin meaning "self-creation" or "self-production," was introduced in 1972 by Chilean biologists Humberto Maturana and Francisco Varela to describe the self-sustaining and replicating capabilities of living cells through the management of their internal environment and continual renewal of components [1]. An autopoietic system is a network of processes that produce components, which in turn continuously regenerate and realize the network that produced them, establishing a closed operational loop. This concept, central to living systems, is articulated as follows:

"An autopoietic system (machine) is organized (defined as a unity) as a network of processes of production (transformation and destruction) of components which: (i) through their interactions and transformations continuously regenerate and realize the network of processes (relations) that produced them; and (ii) constitute it (the machine) as a concrete unity in space in which they (the components) exist by specifying the topological domain of its realization as such a network." [2]

This definition is abstract and self-referential, elegantly depicted by the artistic interpretation of autopoiesis in Figure 1, the process by which a system regenerates itself through the self-reproduction of its own elements and network of interactions. This underpins a model for "soft" computing that diverges from conventional computational systems, emphasizing autonomy and self-regulatory processes [3][4].

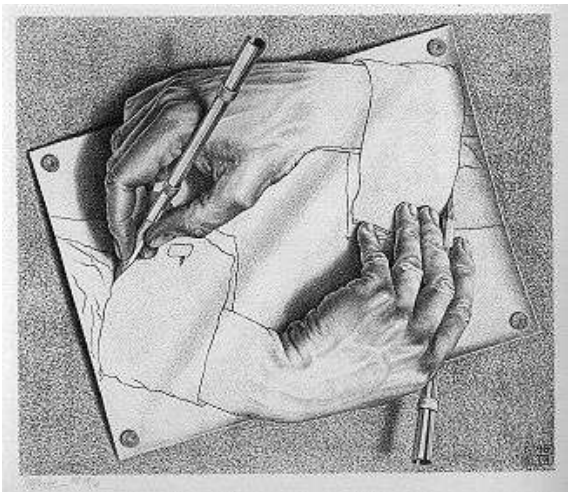

Figure 1 An artistic representation of the process of autopoiesis (From Escher`s Drawing Hands [14])

In distributed systems, autopoiesis manifests as networks where components operate autonomously, yet form a cohesive whole, maintaining structure and function through internal processes. This concept has inspired the development of adaptable and robust computing systems capable of self-management without external intervention [4].

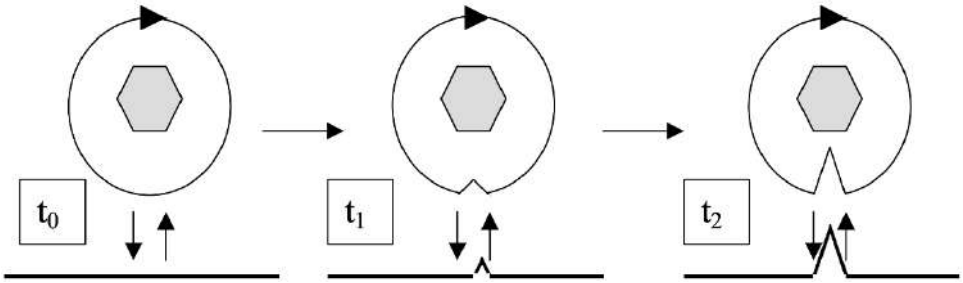

Figure 2 Structural coupling of an autopoieic system

In Figure 2 we can see an illustration of structural Coupling, an aspect of autopoiesis by which the living system and its medium determine—in a mutual way and as a result of a historic process—some of their properties [5]. The autopoietic system is symbolized by a circle, which initially encounters an environment without a structured object. Through ongoing recurrent interactions, the system starts to take form. At the point t_1, an entity emerges consisting of dual interdependent aspects—one resides within the environment and the other signifies a modification in the structural configuration of the autopoietic system [2].

When applied to computing, autopoietic systems focus on internal dynamics and self-guidance, where the environment merely provokes the system, prompting internal modifications that conform to its structure, as visible in Figure 2. This approach is pivotal for creating systems that pre-empt changes, potentially leading to more resilient and intelligent computational models [2][5][13].

Autopoietic systems demonstrate an ability to manage external and internal complexities, balancing multiple objectives through self-organization capabilities such as self-creation, self-replication, self-renewal, self-management, and self-configuration [2][3][6].

2.1.1 Characteristics of an autopoietic system

Autopoietic systems embody several defining characteristics that set their dynamic and autonomous nature. First, these systems exhibit **unity**, where diverse elements operate interactively, independently, and autonomously. They act collectively as a cohesive whole. This unity is not merely a sum of individual components but represents an emergent property arising from the intricate relationships and interactions within the system.

The behavior of autopoietic systems further echoes that of **living entities**, reflecting a form of autonomy akin to living systems. This autonomous behavior is characterized by self-governance, adaptability, and responsiveness to internal and external stimuli. In essence, autopoietic systems emulate the nature of living systems, showcasing a capacity for self-maintenance and continuous adaptation to their environment.

Another fundamental characteristic of autopoietic systems is their inherent **ability to regenerate**. This entails the transformative process of (re-)creating and renewing themselves or specific components and processes within the system. The concept of regeneration emphasizes the dynamic nature of autopoietic systems, allowing them to adapt, evolve, and, in a sense, perpetually recreate and reproduce their essential structures.

Furthermore, autopoietic systems display **self-organization and regulation**. Self-organization refers to the system's capability to organize its components and processes coherently and purposefully without external intervention. Regulation involves the system's ability to govern its composition, preserving its boundaries and ensuring a continuous and stable existence. These regulatory mechanisms contribute to the system's autonomy and resilience, allowing it to navigate and adapt to its environment while maintaining a distinct identity.

In managing the inherent complexity of their environment, autopoietic systems aim to **balance**

external and internal complexity. This balance involves addressing multi-objectives within the system, ensuring that the internal complexity aligns with external challenges and objectives.

2.1.2 Relationship to autonomic and cognitive Computing

Two emerging areas of computing that also draw inspiration from human biology are autonomic computing and cognitive computing. Despite these concepts having much in common with complex systems, they also possess distinct characteristics that differentiate them depending on their objectives. While autopoiesis is inspired by how cells reproduce and multiply to preserve a body, autonomic computing gets inspiration from the human autonomic nervous system and how it autonomously reacts to inputs [7]. On the other hand, cognitive computing aims to replicate human brain functions by employing AI to process diverse data types, adapting based on experiences [8].

As previously mentioned, the concept of autopoiesis is capable of self-creation, as observed in the cells of the biological systems of living beings. This self-referential nature allows such systems to maintain their identity through self-regulation and continuous renewal, independent of their environment. This concept has profound implications for computing, suggesting a paradigm where systems can be designed to self-maintain, autonomously managing their internal structure and operations without external interventions. These systems are reactive and proactive, anticipating changes and adapting accordingly, showcasing a degree of autonomy and resilience that enables them to operate continuously despite internal and external perturbations.

Autonomic computing, inspired by the autonomic nervous system's ability to regulate the human body, refers to self-managing computing systems that can adapt to unpredictable changes while hiding intricacies from users and operators. This concept extends the principle of operational closure from autopoietic systems into the realm of computing. Autonomic computing emphasizes self-* properties: self-configuration, self-healing, self-optimization, and self-protection [9]. In contrast, autopoietic systems focus on self-creation, self-renewal, self-reproduction, and self-organization[10]. Although autonomic computing shares the self-organization aspect of autopoietic systems, it differs in its approach to achieving this. Autonomic systems are not necessarily closed operational systems; instead, they can interact highly with their environment, using feedback to adjust and optimize their performance continuously.

Cognitive computing, in contrast, aims to replicate human cognitive processes in computerized models. This approach leverages AI technologies and approaches, such as machine learning, neural networks, and natural language processing, to create systems that can understand, reason, learn, and interact naturally. Cognitive computing systems are designed to process vast and varied data types, adapting operations based on learned experiences and insights. This is similar to autopoietic systems in terms of adaptation and evolution, but cognitive computing also introduces an element of "understanding" unique to this concept. It transcends the self-sufficiency of autopoietic systems and the reactive nature of autonomic systems by aiming for a form of computational intelligence that is predictive, insightful, and capable of contextual understanding [11][12].

In comparison, autopoietic systems concentrate on self-preservation and the maintenance of identity. At the same time, autonomic computing focuses on self-management and adaptability, and cognitive computing is dedicated to emulating human cognitive processes. Autopoietic and autonomic concepts complement each other, allowing for the construction of more robust, resilient, and autonomous systems. Cognitive computing can enhance both by bringing a layer of intelligence, allowing autopoietic and autonomic systems to make better decisions, predict situations with greater precision, and interpret the environment with enhanced accuracy.

2.2 ML approaches for autopoietic behavior

Machine Learning (ML) approaches have been thoroughly explored to realize self-adaptive characteristics inherent to autopoietic and autonomic computing paradigms within the cloud-edge continuum. In principle, ML techniques stand out for their ability to enable intelligent decision-making, adapt dynamically to changing environments, and optimize resource utilization. The capacity of ML

algorithms to learn from data, identify patterns, and make predictions is particularly valuable in scenarios where rapid, real-time processing and decision-making are crucial, such as in IoT applications and real-time analytics. By leveraging ML, autonomic computing systems within the cloud-edge continuum can achieve improved efficiency, scalability, and responsiveness, thus addressing the critical needs of modern distributed computing environments. In the next paragraphs we explore some work related to this topic.

The authors of [15] presented a comprehensive survey of ML approaches for self-adaptive behavior, including numerous works that improve the quality properties of the overall system. By workload adaptation, a set of predefined objectives and overall system properties can be reached. On top of that, there are well-known trade-offs with increased resource utilization and its impact to cost or energy efficiency. These problems can be modelled effectively using ML techniques. Finally, some studies apply ML to collect unavailable prior knowledge, such as obtaining information about the utility of different configurations [16] or inferring management policies [17].

The primary role of ML algorithms in enabling self-adaptative behavior is to support analysis and planning tasks. Most of the existing work focuses on supervised or interactive learning, which typically utilizes results from runtime analysis and the observed effects of applied adaptations. The most common self-adaptation problem addressed through learning is the updating and changing of adaptation rules and policies, often tackled using regression and reinforcement learning (RL). Other notable challenges include predicting and analyzing resource usage and maintaining up-to-date runtime models, mainly solved using regression and classification. Model-free reinforcement learning emerged as a popular method for updating adaptation rules and policies. Over the years, supervised and interactive learning have been consistently used, whereas unsupervised learning, valuable for detecting new patterns in unlabeled data, has seen limited application. No single type of learning—supervised, interactive, or unsupervised—has been distinctly more impactful over time for the academic community.

[18] presents an analysis of ML approaches explored for self-adaptive systems. RL methods are the most prevalent approach, due to its ability to explore solution spaces with substantial uncertainty and adapt to dynamic changes in the environment. Following that, both fuzzy logic-enhanced approaches, genetic algorithms and Bayesian methods have been integrated into self-adaptive functions to achieve self-adaptive behavior.

These algorithms nonetheless present challenges for their application to autopoietic behavior functions. A major challenge of many techniques is the scalability of the approaches, considering the complexity, heterogeneity, and potentially vast volume of data that must be handled to reach an action decision in this environment. Moreover, the performance of these techniques in improving the target system has several aspects that need to be evaluated. As a closed control loop in practice, it is important when applying these techniques to study and understand the effect of the decisions themselves over time, as they in turn will change the ACES environment, and therefore receive future feedback on the runtime state. Moreover, distributed, large environments present highly dynamic conditions that require the ability to adapt to scenarios potentially different to the ones initially trained on, which is a capability shared by several of the ML approaches we have listed.

2.3 Swarm intelligence techniques for autopoietic edge cloud systems

The requirements of optimizing resource management in edge-cloud computing present numerous challenges derived from the complexity and big data scale of the problem. We list some examples of optimization problems in big data analytics that can exhibit expensive computational complexity [20]:

- Combinatorial Feature Selection: When dealing with many features (variables) in a dataset, selecting the optimal subset of features for a machine learning model can be computationally intensive. The number of possible feature combinations grows exponentially with the number of features, leading to exponential complexity [21][22].

- Clustering in High-Dimensional Spaces: In high-dimensional spaces, clustering algorithms like k-means can become computationally expensive. As the number of dimensions increases, the data points tend to become more distant from each other, making it challenging for clustering algorithms to identify meaningful clusters. This phenomenon is often referred to as the curse of dimensionality [23][24] .
- Optimizing Distributed Systems: Optimizing the allocation of computing resources in distributed machine learning systems for big data analytics can be computationally expensive. These systems often involve multiple nodes and parallel processing of large data sets. Ensuring efficient resource allocation to reduce training time and resource waste is a challenging optimization problem [25][26].

A potential approach to overcome the scalability challenges of these management techniques for ACES is to explore bottom-up approaches. Autopoiesis can be pursued through emergent optimization using swarm intelligence by employing interacting embodied agents that make decisions based on local information using agent-based modelling. Such algorithms are robust, adaptive, and scale due to their distributed characteristic leading to a real emergent behavior of a complex system [20]. Next, we present some related approaches.

In the context of Next-Generation Networks (NGN), Pham et al. [27] do an in-depth review of the implementation of swarm intelligence for and state the advantage of swarm intelligence in guaranteed convergence, robustness, near-optimal solution, and computational traceability. The common approach is to initially create a random set of solutions. This set of candidate solutions is improved iteratively, optimizing the objective function, which quantifies the goodness of a solution. Swarm intelligence has been applied also for spectrum management and resource allocation, wireless caching, and network security.

More recent research explored several optimizations for edge computating. In smart homes, to minimize the energy consumption of a residential consumer-centric load-scheduling, Lin & Hu [29] proposed a constrained Particle Swarm Optimization (PSO) algorithm, where the possible solutions are modeled as agents. Feng et al. [30] also describe a task offloading strategy, which is able to reduce the energy consumption, the time latency and the service price in mobile edge computing. Their strategy is to use a Grey Wolf Optimizer (GWO), Whale Optimization Algorithm (WOA) and hybrids where the agents are the percentages of how much of a mobile device's task is computed locally on the mobile device, because a task can also be partially offloaded to the edge server. This means that whenever there is a change in the tasks or the mobile devices, the algorithm needs to compute the optimal solution with a new set of input. Lee et al. [31] provides a swarm intelligence algorithm, an Artificial Bee Colony (ABC), for the allocation of a given task set to a given edge server set and a cloud. A relatively recent work, Mahenge & Sanga [32] presents a strategy to offload resource-intensive tasks in mobile edge computing energy-efficiently using a hybrid approach (PSO and GWO), where the algorithm gathers the information about the tasks and servers and then calculates the optimal offloading strategy. Bacanin et al. [33] perform energy optimization in 5G-enabled edge nodes using PSO. Attiya et al. [34] tackle the problem of IoT application task scheduling using the Manta Ray Foraging Optimization (MRFO) combined with Salp Swarm Algorithm (SSA). In Singh et al. [35] the authors write all available resources into an availability list. On this list, a swarm algorithm (ant colony optimization, ACO) is executed for searching an optimized (centralized) solution for resource allocation and scheduling. Another approach is presented in de Melo et al. [36]. Here, the focus is on methodologies to parallelize swarm algorithms on parallel hardware to increase execution performance. The aim is to accelerate finding an optimal solution to a problem which is then mostly applied in a centralized manner. No decentralized agent-based approach is revealed in this work. Although the proposed solutions in the literature apply different swarm intelligence algorithms, they are executed centrally. Typical problems that arise from this approach are single point of failure, higher computational effort, and lack of dynamicity to occurring changes in the environment or incoming demands.

To the best of our knowledge, ACES is a first application of an agent-based approach in the edge-cloud continuum where resources and requests are regarded as agents, and scheduling along with relevant objectives (utilization, low latency, energy efficiency, etc.) are considered emergent properties of the agent's local decision making and interaction (autopoiesis).

3 Action library

This section presents the categories and specific actions that can be decided by agents in order to modify the ACES environment. These actions affect runtime elements of the ACES platform. We structure this action library in two complementary categories. First, we describe actions that explicitly aim to satisfy demand-side goals. These frequently deal with resource allocation requirements, and we will separate them depending on the type of resource that is being accessed. Additionally, we present a second set of actions that are meant to improve non-functional concerns that have been identified in the project, in particular security and network performance challenges.

3.1 Actions to achieve goals for the demand

Demand actions define what changes on the environment can be triggered by the agents, swarm, and ML components, i.e., how ACES interacts with elements such as the Kubernetes base engine, or the networking devices. We present them in three subsections, first for workload management, then for storage, and finally for networking resources.

3.1.1 Workload placement actions

The ACES workload is captured as Kubernetes deployments, pods, and services. The way to interact with these elements is through the core Kubernetes mechanisms. The ACES way of working will provide an efficient approach to managing these Kubernetes resources via workload placement actions.

To enhance fault tolerance, optimize workload placement, and manage demand fluctuations, the ACES platform must possess the capability to schedule, reschedule, scale up, and scale down deployed workloads efficiently. Controlling the number of replicas for each pod is a core capability that enables these processes. Actions are used to describe how the current state of the platform needs to be modified to achieve a new state, which is a step closer to the optimal placement of the workloads. Intelligent components, or a combination thereof that consider workload descriptors, as well as the historical and present platform state (metrics, traces, current placements, etc.), should be employed to generate a set of actions. However, the task still needs to be completed as the platform must now discern the execution order for these actions. This order may involve sequential execution, parallel execution, or a combination, depending on factors such as temporary replica surges, pod disruption budgets, etc. In short, the actions must be transformed into an execution plan.

In this section, we identify the types of actions that a smart component can generate. Firstly, we categorize actions into two levels: workload-level actions and replica-level actions. The distinction lies in the fact that a workload action is invariably expanded into one or multiple replica actions. For example, consider the scenario where a workload-level action aims to relocate all replicas from one EMDC to another. In this instance, a replica-level action must be generated for each replica in the edge layer, orchestrating the move between locations. Consequently, a singular workload-level action expands into multiple replica-level actions. Likewise, a workload-level action aimed at scaling up the workload should be first expanded into a workload-level action regarding the distribution of the new replicas. This, in turn, unfolds into multiple replica-level create and place actions.

We have identified the following simple placement and re-placement actions related to replicas of the workload:

- *Create and place a replica of the workload to a node*: this action instructs the creation of a new replica for the workload and specifies its placement on the designated node.
- *Delete replica of workload*: deletion of a specified replica of the workload.

- *Move a replica of the workload from one node to another*: movement of a specified replica from node, at which this replica is currently placed on, to another node.
- *Swap the replicas*: movement of a specified replica from node, at which this replica is currently placed on, to another node.

The list below includes more complex placement and re-placement actions:

- Distribute or redistribute replicas present in the list of workloads across nodes in the list using distribution strategy.
- Horizontally scale up the workload by expanding replicas, adding replicas in list.
- Horizontally scale down workload by contracting replicas, removing replicas from the list.
- Rescheduling, namely how to reallocate workloads to different nodes to improve efficiency.

The above actions are discussed next.

Distribute or redistribute replicas present in the list of workloads across nodes in the list using a distribution strategy.
The distribute action is used to place new replicas across nodes. A scale-up action or simply deploying a new workload may create these replicas. The redistribute action moves existing replicas across nodes to achieve optimal placement. In addition, this action also utilizes a distribution strategy that defines how the list of replicas should be distributed across the list of nodes. The agents and swarm components will select the distribution strategy in ACES. Suppose there are no constraints and unfavorable circumstances in the cluster. In that case, one example of such a strategy that could be utilized in ACES represents a uniform distribution of replicas across nodes.

As simple as this action might seem initially, it can be used to realize many edge-cloud placement and scaling optimization scenarios. For example, if the ACES platform has identified that some workloads specific to some edge locations cannot handle the demand, some replicas could be offloaded to the cloud. To illustrate even further, let's consider a scenario where a particular edge location is deemed unsuitable for a specific type of workload. Consequently, the ACES platform resolves to relocate all replicas of that workload from this edge location to another edge location or possibly even to the cloud.

Horizontally scale up workload by adding replicas present in the list
It is of utmost importance that ACES adapts to the increase in demand by scaling up the number of replicas of some workloads. This action could be achieved by utilizing the horizontal scale-up and specifying a list of new replicas. The list size can depend on many things, but mainly on the scale of the demand increase. The decision of where these new replicas should be placed is determined by the previous action.

Horizontally scale down workload by contracting replicas and removing replicas from the list
To prevent resource wastage during periods of decreased demand, the ACES platform needs the capability to scale down the number of replicas. This involves identifying a list of replicas that should be deleted. In this scenario, the scale-down action expands into multiple replica-level delete actions.

Rescheduling
The assumption is that a pod has already been scheduled and is operational on a node. The question is how a pod can be relocated to a more efficient node and what the process entails. While the rescheduling procedure should be very similar to the scheduling procedure, this section focuses exclusively on the rescheduling aspect. Determining the optimal node and orchestrating the pod's migration in ACES can be achieved through two distinct approaches. One approach involves a centralized system, such as the Kubernetes rescheduler, which assumes responsibility for optimizing the placement of pods across the entire cluster. Alternatively, a decentralized method can be employed, where each pod or its respective agent independently makes rescheduling decisions, bearing responsibility only for its placement.

The list of workload placement actions can be applied to both centralized and decentralized rescheduling strategies. Furthermore, the ideas can be expanded to rescheduling in multiple inter-connected clusters. When the pod needs to be rescheduled to a more optimal placement, there are

several considerations that must be taken into account, including the selection of a more optimal set of nodes, as well as the possibility and suitability of rescheduling.

Nodes provide metadata and metrics information to the rescheduling entity. In centralized rescheduling, the nodes are connected to the control plane, and in decentralized rescheduling, the information is provided to deployment agents via supply agents. Pods can optionally express their requirements or preferences using node selectors and node affinities. For example, a pod can express a preference to schedule on the edge, or it may require a GPU, or it may require a certain minimum bandwidth to function, or it may want to schedule close to some other pods that it commonly communicates with. In order to provide the pods with as much information about the nodes, each node should provide the required metadata and metrics.

When selecting a more optimal set of nodes, the rescheduling entity should consider both the pod's specification, resource usage metrics, business metrics (which might be represented as SLIs and SLOs) and persistent disk locations, as well as the node's specification and resource metrics. What kind of business metrics a pod exposes can vastly differ from one type of workload to another. ACES will strive to provide a way to encode at a higher level (deployment, StatefulSet, application, etc.) what kind of metrics the pods expose and define how to determine the quality of the service as a whole (taking into consideration all metrics of the Pods)—SLIs and SLOs could be encoded as part of the specification. After considering the node selectors, node affinities, and resource requirements, the rescheduling entity should decide whether the rescheduling provides a significant enough benefit to the pod since the process might be costly. Moreover, after a set of more optimal nodes has been identified, the rescheduling entity has to consider any pod disruption budgets, topology spread constraints, encapsulating object's upgrade strategy (for example, deployment's rolling update), etc., in order to determine if the rescheduling is possible. If rescheduling would, for example, violate the disruption budget, then it should be attempted again later.

If all the requirements mentioned above have been satisfied, the rescheduling entity can edit the nodeName property in the pod's specs and terminate the pod. The pod will then be recreated on the desired, hopefully more optimal node. In some cases, according to the specification of the deployment or some other object that encapsulates the pod, the pod should wait until a replica of itself is created before terminating itself.

3.1.2 Storage management actions

ACES focuses on the management of workloads deployed as a set of microservices. However, these functional components frequently must manage their own internal information. These additional storage elements can be captured as black-box pods that are part of these microservices, e.g., by deploying an individual database that is consumed exclusively by that service. These pods would then fall under the same type of workload actions that have been described above. Nonetheless, we observe with more details that inside the selected Kubernetes architecture there are additional options for managing microservice storage.

In Kubernetes, managing storage for microservices can be approached through several strategies to ensure data persistence and efficiency. The use of Persistent Volumes (PVs) and Persistent Volume Claims (PVCs) allows for the decoupling of storage lifecycle from the pods, enabling independent management of storage resources. Storage Classes provide a way to define different types of storage (e.g., SSD-based for performance or HDD-based for cost-efficiency) that can be dynamically provisioned as needed. StatefulSets are another option, particularly suited for stateful applications, as they manage the deployment and scaling of a set of pods while maintaining the sticky identity and storage across pod rescheduling. ConfigMaps and Secrets offer mechanisms to manage configuration data and sensitive information without embedding it directly into the application code. Lastly, the integration of Container Storage Interface (CSI) plugins extends Kubernetes to support a wide range of external storage systems, allowing for more specialized and advanced storage solutions to be seamlessly integrated into the Kubernetes environment. Each of these alternatives provides different benefits and can be chosen based on the specific requirements of the microservice applications, such as performance needs, data durability, and the complexity of data management.

This set of potential Kubernetes resources would also be managed through native actions from the Kubernetes management interfaces, as described above.

Decentralized data management actions
Decentralized data management systems, where data is spread across various nodes to enhance resilience and scalability, bring the need for specific data management actions. This leads to two main characteristics of data in these systems: replication of data elements across multiple nodes to improve reliability, and data partitioning or fragmentation to improve scalability. We note that unlike the general data management concerns, these will be specific to the data management platform that is being adopted, therefore this subsection only explores potential approaches without detailing its actual implementation.

Data synchronization actions would improve consistency across different replicas, often employing consensus protocols like Raft or Paxos to maintain data integrity. There are several aspects that can be configured in this space, such as Consistency Tuning techniques, supported natively by systems such as Cassandra[1], that allow volves adjusting the consistency level of read and write operations, balancing between strict consistency and eventual consistency based on the application's tolerance for data staleness, and its underlying SLOs.

On the other hand, regarding partitioning there are direct actions that can modify the allocation of each data element to one of the shards or elements forming part of the distributed data management system. Data Sharding actions will have to strategically partitions data across different nodes, balancing the load and reducing latency by allowing queries to be processed by the node nearest to the data's location.

For all these actions an important aspect is the physical distribution of these replicas, either to improve performance of applications, or to achieve better fault tolerance, the information about the topology of the different nodes will be very important for all the decentralized data management decisions and actions.

3.1.3 Network management actions

ACES orchestrates various network management actions to ensure high performance and adaptability. At the heart of its network management strategy lies the integration of artificial intelligence and machine learning (AI/ML) to navigate the intricacies of the network fabric and the interaction between edge services. By leveraging AI/ML algorithms, ACES dynamically adjusts network configurations, optimizes resource allocation, and predicts network demands, contributing to a self-aware and adaptive network infrastructure.

For this purpose, the ACES network uses Software-Defined Networking (SDN) and programmable switching technology. These provide a set of native actions that can modify the behavior of the ACES network. This combination allows ACES to exert granular control over network traffic, dynamically rerouting flows based on real-time conditions and priorities. SDN's centralized control plane, coupled with programmable switches, empowers ACES to respond swiftly to changing network requirements, ensuring efficient data transmission, and reducing latency across the edge infrastructure.

The ACES management action also includes in-network monitoring as a vigilant guardian against potential threats and attacks. This proactive approach involves continuously analyzing network traffic within the system, enabling ACES to detect anomalies and raise alarms in the face of suspicious patterns. By integrating real-time monitoring, ACES enhances its resilience against network attacks, providing a robust defense mechanism to safeguard sensitive data and ensure the integrity of the edge computing environment.

Furthermore, ACES extends the collaborative power of its network switches into the realm of swarm intelligence mechanisms. ACES switches can eventually participate in swarm-based decision-making processes, contributing to the system's collective intelligence. By fostering collaboration among

[1] Cassandra DB, https://cassandra.apache.org/

network switches, ACES enhances its adaptability and problem-solving capabilities, creating a self-organizing and self-optimizing network infrastructure that dynamically responds to changes. The integration of swarm intelligence with network switches synergistically contributes to the autopoietic nature of the ACES edge computing system.

3.2 Actions to achieve non-functional goals

In this section, we present the actions related to the non-functional goals that underpin the ACES design, including performance and security. The performance objectives are geared towards accelerating network functions, aiming to optimize the system's responsiveness and efficiency. On the security front, ACES unfolds a multi-faceted approach, encompassing in-network intrusion detection, fortifying edge and container security, and developing defenses against potential attacks targeting machine learning algorithms. The actions towards these non-functional goals represent the intricate balance between speed, intelligence, and resilience that ACES aspires to achieve.

3.2.1 Accelerating network functions

Software network functions (NFs) trade-off flexibility and ease of deployment for an increased performance challenge. In ACES, we will investigate techniques to achieve both goals, by exploring both host-based and network-enhanced mechanisms to accelerate NFs. Unfortunately, accelerating network functions is hard. The NF developer needs intricate knowledge of the NF semantics and internals, needs to learn domain-specific languages or, more commonly, both. In ACES we leverage program synthesis principles to automate the process.

On the host, the ACES approach is to increase NF performance by distributing traffic to multiple CPU cores. However, this poses a significant challenge: *how to parallelize an NF without breaking its semantics?* Our approach is to use program analysis tools to analyze sequential implementations of an NF and automatically generate an enhanced parallel version that carefully configures the NIC's Receive Side Scaling mechanism to distribute traffic across cores, while preserving semantics. When possible, we aim to orchestrate a shared-nothing architecture, with each core operating independently without shared memory coordination, maximizing performance. Otherwise, we can resort to fine-grained read-write locking mechanisms.

The network-enhanced mechanisms involve taking advantage of heterogeneous platforms, including network switches and x86 CPUs to improve performance, efficiency, and resource consumption. Again, the challenge is that programming for multiple hardware targets is hard because developers must learn platform-specific languages and skills. Our goal in ACES is to work towards a compiler that explores a large search space of different mappings of functionality to hardware. This exploration can then be tuned for a programmer-specified objective, such as minimizing memory consumption or maximizing network throughput.

3.2.2 In-network malicious traffic detection

In ACES, we will develop a malicious traffic detector to identify network attacks. However, the most widely deployed rule-based systems are limited to known attacks. We target zero-day attacks, which can easily bypass their protection. As such, we will explore recent ML-based attack detectors that show promise as a complement to deployed systems. These detectors track deviations from regular traffic patterns to detect attacks, achieving high detection rates and enabling the detection of previously unknown attacks. However, they face a performance challenge: the overhead of ML processing results in orders of magnitude decreases in throughput compared to their rule-based counterparts, limiting their practicality.

We will develop a cross-platform malicious traffic detector. The key idea is to run the detection process *partially* in the data plane. Specifically, we plan to offload the ML feature computation to the data plane of a network switch. The goal is for the ACES switch to process tens of features of diverse types *per-packet, at Tbps line rates* to feed the ML-based detector that runs in the control plane. The

ACES offloading approach presents a distinct advantage. While, in practice, current systems need to sample traffic at low sampling rates to downscale from the data plane packet rates to the much lower ML detection processing speeds, in ACES sampling shifts to *after* feature computation. This essential trait makes ACES the first system that computes features *over all traffic*, which we expect will significantly improve detection performance in today's Terabit networks. Additionally, offloading this compute-heavy component can also save precious CPU cycles.

3.2.3 Privacy and trust at the edge

ACES services and applications will be deployed and executed in a zero-trust environment at the edge, with ACES nodes replicated across various heterogeneous and vulnerable edge sites. Due to their exposed locations, these edge sites incur significant security risks, including data leaks, attacks, corruption, and failures. To address these challenges, ACES needs to build trust in its replication scheme while preserving user privacy.

To ensure that ACES maintains its availability despite failures or attacks, and continues to respect its autopoietic behavior, the implementation of service and storage replication is critical. However, implementing replication is insufficient in a zero-trust environment like the edge. It is crucial to verify the correctness and integrity of the replication. Storage proofs offer a practical solution to this challenge, enabling edge nodes to establish mutual trust and implement necessary replication levels adequately. ACES will develop a novel storage proof capable of auditing whether an edge provider can retrieve data objects within a latency lower than a predefined SLA threshold.

Our cryptographic proof will be supported by Trusted Execution Environments like Intel SGX to guarantee that the proof originates from the node under audit and minimizes the communication required between the auditor and the audited node. The proof must be meticulously designed to mitigate any noise and variance from the edge network, attaining millisecond-level accuracy. This proof should precisely determine the data's location at the edge, verifying compliance with specified SLAs. This auditing tool is crucial for fostering trust among the diverse entities within the edge network.

Privacy concerns, amplified by the physical proximity of entities at the edge, are also paramount. Entity authentication, especially for clients, is crucial to guard against unauthorized access to data and services, but such proximity can jeopardize user privacy, potentially exposing sensitive information like location data. Offering anonymous authentication in ACES is imperative to protect client privacy and align with European laws such as the GDPR. Existing techniques for anonymous authentication, like group signatures and zero-knowledge proofs, are computationally demanding for heterogeneous edge devices. In ACES, we will introduce novel authentication schemes based on efficient public key encryption to ensure client privacy at the edge.

The scheme we will develop will provide client authentication anonymity based on pseudonyms. In addition, we require the anonymity to be preserved even after a potential revocation. We will craft our scheme to provide time-bound revocability without undermining the unlinkability of a user's actions pre- and post-revocation.

3.2.4 Container security

The adoption of container-based applications is surging, offering unparalleled efficiencies in software development, deployment, and operation. Within ACES, containers are integral for rolling out intelligent agents and services. However, alongside their rising popularity, many security and privacy concerns are escalating in containerized environments. Vulnerabilities that lead to privilege escalation and remote code execution attacks pose a significant threat, potentially jeopardizing entire container ecosystems.

To counteract these risks, we will propose dedicated container security solutions to detect and protect against sophisticated threats. This includes, but is not limited to, privilege escalation, exposure of sensitive credentials, and denial of service (DoS) disruptions. Our defense framework will

utilize cutting-edge vulnerability scanning and a dynamic, deep learning-based approach to anomaly detection. These proactive measures ensure the timely identification of security weaknesses and the real-time interception of active threats, maintaining the integrity and reliability of our containerized deployments.

3.2.5 AI/ML security

Machine Learning (ML) is crucial for enhancing the capabilities of ACES intelligence agents and services, including at end host and network levels. Unfortunately, ML algorithms and systems are susceptible to several security and privacy concerns. To safeguard against these vulnerabilities, the ACES framework will integrate robust security protocols that protect against data and model poisoning, as well as inference attacks, which are particularly pertinent in distributed systems, such as federated learning. Our defense strategy against backdoor attacks will combine state-of-the-art techniques like model clustering, strategic parameter clipping, and the introduction of noise into model parameters. We will adopt a suite of privacy-preserving technologies to mitigate inference attacks, including Secure Two-Party Computation (STPC), trusted execution environments, and blockchain technology. These interventions are designed to limit exposure to local model updates, significantly reducing the risk of successful inference attacks.

4 Actions for the emergent workload scheduler

This section presents an initial proposal to develop workload scheduling actions in the ACES project. The following sections detail the implementation of algorithms and agent-based models designed to manage and distribute tasks across the edge-cloud infrastructure. This chapter explains how both swarm algorithms can be adapted to these workload scheduling problems, and later on how ML methods can be integrated to further tune the actual performance of a fully decentralized algorithm in the edge-cloud computing environments.

4.1 Swarm algorithms for the emergent scheduler

Central to our approach is the integration of autopoietic characteristics that include the emergent intelligence of self-organization, regeneration, and regulation. These characteristics enable the system to dynamically adapt and optimize in response to changing conditions. AI-driven optimization methods (including swarm intelligence) in cloud infrastructure are successfully being researched (see Deliverable D2.1 for more details). Among recent notable examples of utilization of swarm intelligence to optimize complex systems, is our work in Schranz et al. [19], where we successfully utilize bottom-up job shop scheduling applying swarm intelligence algorithms for optimizing a large production plant. Thus, we propose the edge continuum with its characteristics and limitations as a novel field of application for swarm intelligence.

Key to our approach is the use of swarm agents, representing demand and supply entities. Demand swarm agents represent workload behaviors at the pod level, ensuring pod-level optimization. On the other hand, supply swarm agents represent node dynamics. These agents collaborate within an Edge Micro Data Center (EMDC) environment, orchestrating processes such as workload placement, storage management, and caching optimization. The interaction between demand swarm agents and supply swarm agents is orchestrated through swarm intelligence algorithms. Demand swarm agents autonomously seek out the most suitable node for workload placement, while supply swarm agents determine the optimal workload to process based on available resources and capacity. This collaborative decision-making process enables the system to efficiently allocate workloads to nodes, optimizing processing, latency, and resource utilization. Exemplary swarm algorithms that we present in this deliverable are the hormone and ant algorithms to accomplish the desired functionality of the system. For example, demand swarm agents deploy synthetic hormones to communicate their requirements and priorities. Supply swarm agents, detect these hormones to make informed allocation decisions. The ant algorithm dynamically optimizes workload-node-assignments by simulating the foraging behavior of ants, depositing pheromones to guide subsequent decisions [20].

Our agent-based model is designed to exhibit autopoietic characteristics, fostering self-organization, regeneration, and regulation within the edge continuum. As demand and supply agents interact and adapt to changing workloads and resource availability, the system displays emergent behaviors that contribute to its resilience and efficiency.

In the following section, we introduce two candidate algorithms for the edge continuum. These algorithms adopt a bottom-up approach, mirroring real-world entities in this context as digital twins. These digital twins are already equipped with attributes that enable them to interact within their virtual environment. By applying swarm intelligence principles to these digital twins, akin to the concept of digital pheromones, we enhance their capabilities. This allows them to make context-aware decisions by drawing from both local and global information. This approach embraces decentralized decision-making, promising effective resource management in the complex edge-cloud continuum.

4.1.1 Hormone algorithm

Artificial hormone systems draw inspiration from the biological endocrine system, which regulates various metabolic processes within our bodies [42]. This creates a self-organizing system characterized by scalability, adaptability, and robustness. In our simulation, supply swarm agents correspond to nodes within the continuum, and demand swarm agents represent pods seeking optimal node placement [20].

Demand swarm agents release synthetic hormones into the environment based on their resource requirements and preferences. These hormones carry information about the demands and priorities of the pods. Supply swarm agents, representing nodes, detect these hormones and adjust their behavior accordingly. Nodes release their own hormones indicating resource availability and capacity.

The concentration of hormones guides demand swarm agents toward nodes that match their requirements, fostering autonomous and informed decision-making. The communication of synthetic hormones replaces traditional centralized control mechanisms with decentralized coordination, allowing the system to adapt to pod variations and resource fluctuations.

The underlying principle is inspired using artificial hormones for reorganizing agents in self-organizing systems for technical applications [28], which can be extended to the dynamic edge environment. In our framework, we will implement the artificial hormone system as a software layer distributed across the processing nodes within the edge continuum as inspired by the applications in production plants (see [43] for more details). The hormone algorithm used for optimization in the edge continuum can be dissected into six key mechanisms:

Production: Supply swarm agents, representing nodes, produce hormones in response to the number of demand swarm agents, pods, in the EMDC. Nodes that are currently processing fewer pods produce more hormone. Each node as well as a pool of resources (e.g., CPU, storage) may produce a distinct type of hormone with

$$H_i^r = \frac{1}{|Q_i^r| + \beta}$$

where H_i^r is the hormone corresponding to the node N_i^r, β is a smoothing factor, and $|Q_i^r|$ is the number of waiting workloads in the EMDC for the node N_i^r.

Evaporation: The hormone levels at each node gradually decrease over time through a process of evaporation, controlled by a parameter given with

$$H_{i,t+1}^r = H_{i,t}^r \cdot (1 - \alpha)$$

where $H_{i,t+1}^r$ and $H_{i,t}^r$ represent the state of hormone at the node N_i^r before and after a discrete evaluation step.

Diffusion: Hormones diffuse from one node to another based on the compilation of resources per pod that also connects the resources similar to hormone propagation in biological systems. Hormones move upstream, following the reverse of this resource graph by calculating

$$\Delta H = H_i^r \cdot \gamma$$

$$H_i^r -= \Delta H$$

where ΔH is the amount of hormone moving upstream from the node N_i^r, and γ is a parameter setting the motility of hormone.

The link strength $l^{r,p}$ between two nodes N_i^r and N_i^p is equivalent to the number of compilations of resources R_t containing processes P^r and P^p in direct succession. Each node connected upstream receives a proportional part of the upstream hormone with

$$H_j^p \mathrel{+}= \Delta H \frac{l^{r,p}}{\sum l^{r,c}}$$

where $\sum l^{r,c}$ represents the sum of all upstream links from P^r.

Diffusion through pod movement: When pods move between nodes within the EMDC, they carry hormones with them, influencing the hormone levels at both the initial and destination nodes.

$$\Delta H = H_i^r \cdot \delta$$

$$H_i^r \mathrel{-}= \delta H$$

$$H_j^p \mathrel{+}= \delta H$$

where ΔH defines the amount of hormone that moves with the pod, calculated from the amount of available hormone H_i^r at the node N_i^r.

Attraction: Pods are attracted by the nodes whose processing capabilities match the pods' requirements from the corresponding compilation of resources. The amount of attraction decreases exponentially based on the order of the node. The attraction force is applied to pods as soon as they enter the processing queue Q_i^r and it can make the pod move towards a distinct node.

$$attraction = \sum H_i^r \cdot \varepsilon^n$$

where H_i^r is the hormone amount at a node that is n edges away, and ε^n is a factor <1 defining the degradation of the hormone attraction over edge distance in the graph G.

Each mechanism comes with a parameter indicating the strength of each part, that is evaporation rate α, hormone production factor β, upstream diffusion factor γ, hormone distribution factor δ, and attraction factor ε. A possible configuration of these parameters is stated in [43]. Due to the interaction between each of the mechanisms forming feedback control loops, the algorithm can operate with a broad set of possible parameter settings.

4.1.2 Ant algorithm

Ant algorithms draw inspiration from the decentralized foraging behavior of ants, a natural phenomenon where ants can efficiently find near-optimal paths to food sources without relying on global knowledge. They achieve this by leaving pheromone trails to communicate with other ants. In our simulation within the edge continuum, this concept will be applied to optimize the allocation and processing of pods by supply swarm agents, analogous to ants, representing nodes within the continuum.

In the following, an ant algorithm adaptation to the edge continuum is presented [20]:

Trail Following: In our context, we frame the allocation of pods as a routing problem in the edge continuum. Pods probabilistically select the next suitable node N_i^n from the set of potential nodes N^n based on both local pheromone values associated with that node and a local heuristic considering the node's current pod, which can be assessed by metrics like queue length or resource utilization. The probability $P_{i,j}$ of selecting node N^n is computed as follows:

$$P_{i,j} = \frac{\tau_{i,j,d} + \alpha\eta_{i,j}}{1 + \alpha(N_i - 1)}$$

with

$$\eta = 1 - \frac{q_{i,j}}{\sum q}.$$

In this equation, η represents the relative queue length of node with N_i as the number of possible nodes. The parameter α allows for fine-tuning the influence of pheromone τ (see update rules below) versus the local pod heuristic. In our adaptation, the destination d corresponds to the next step in the pod's compilation of resources within the EMDC, rather than a specific destination node.

Trail Laying: Pheromone values are updated after a pod has been processed on a node within the EMDC. However, unlike traditional ant algorithms where backward ants are used to update pheromone values, we utilize communication and coordination among nodes within the continuum. Each pod maintains a memory of the processing, effectively measuring the time it waited for resources. When a pod moves from one node to another, this information is used to update the pheromone values.

For a chosen node N_x^n, the pheromone value is updated as follows:

$$\tau_{x,d} \leftarrow \tau_{x,d} + r\left(1 - \tau_{x,d}\right)$$

For all potential nodes N_n^n that were not chosen, the pheromone values are updated according to:

$$\tau_{n,d} \leftarrow \tau_{n,d} - r\tau_{n,d}.$$

The reinforcement r depends on the processing time of the pod, which reflects the waiting time and resource utilization. This approach ensures that nodes with shorter pod processing times and lower resource utilization become more attractive for incoming workloads.

Evaporation: Periodically, pheromone values are subject to evaporation with a rate ρ. This process simulates the natural fading of pheromone trails and helps remove paths that may have become less optimal due to changes in resource availability or demand

$$\tau(t+1) = \tau(t)(1-p)$$

This adaptation effectively models and optimizes the allocation and processing of pods within the EMDC, drawing inspiration from the decentralized behavior of ants.

4.2 Actions to modify swarm behavior

As outlined in Sections 4.1.1 and 4.1.2, swarm algorithms depend on hyperparameters that significantly impact the collective behavior of the coalition. These hyperparameters include settings governing the rate of hormone evaporation, hormone mobility, the strength of hormone attraction, to just name a few.

Typically, hyperparameters are determined through methods such as trial-and-error, random/grid searches, and heuristics [37]. Once these values are established, it is infrequent to adjust them during the execution of the swarm algorithm.

As an innovative feature within ACES, the hyperparameters of the swarm algorithm will undergo self-tuning using ML techniques. These ML methods will additionally facilitate real-time updates of hyperparameters, enabling the coalition's behavior to dynamically adjust to significant environmental changes. This can be achieved by utilizing previously discovered hyperparameter configurations as starting warm-up of the algorithms for adjustments and refinements, without doing all the computation and the exploration from scratch.

To achieve this objective, we will implement and assess two primary ML approaches: Bayesian learning and Reinforcement Learning (RL).

4.2.1 Bayesian learning

Following the principles of Bayesian learning, the swarm algorithm will be regarded as a black-box tool. It will receive a particular set of hyperparameter configurations as input and, following an experimental process, yield specific quantitative performance indices, namely the system's KPIs. This black-box can be considered an unknown objective function that we seek to maximize. We can submit hyperparameter configurations to it, and in return, it provides the corresponding KPI values.

Unfortunately, this objective function is very time-consuming to evaluate. For this reason, it is of interest to minimize the number of function evaluations by replacing the function to minimize with a surrogate function [38]. The surrogate is then used to solve a (much cheaper) global optimization problem that decides the new point where the original function must be evaluated. A better-quality surrogate is then created by also exploiting the new sample and the procedure is iterated.

Bayesian learning is a popular class of global optimization methods based on surrogates that, by modelling the black-box function as a Gaussian process, enables one to quantify in statistical terms the discrepancy between the true objective and the surrogate, an information that is considered to drive the search of the optimal hyperparameters.

More specifically, assume that we collected a dataset $D = \{x_j, y_j\}_{j=1}^{M}$ of length M, where y_j is a noisy observation $f(x_j)$ of the KPIs for an hyperparameter configuration x_j, and f is the function that we want to estimate, then the following linear combination of Radial Basis Functions (RBFs) [39] can provide a reliable surrogate:

$$\overline{f}(x) = \sum_{j=1}^{M} \beta_j \, \phi\left(\varepsilon d(x, x_j)\right),$$

Through where $\phi: \mathbb{R} \to \mathbb{R}$ is an RBF, $d(x, x_j)$ being any distance function between x and x_j, and $\varepsilon > 0$ is a scalar parameter defining the shape of the RBF. The unknown coefficients $\beta_j, j = 1, \dots, M$ are determined by fitting the model $\overline{f}(x)$ to the dataset D through the minimization of the regularized square error:

$$\sum_{j=1}^{M} \| \, y_j - \sum_{k=1}^{M} \beta_j \, \phi\left(\varepsilon d(x_j, x_k)\right) \|^2 + \gamma \, \| \, \beta_j \, \|^2,$$

Where the quadratic regularization term is added to guarantee strict convexity. Some RBFs commonly used are $\phi(\varepsilon d) = \frac{1}{1+(\varepsilon d)^2}$ (inverse quadratic) and $\phi(\varepsilon d) = e^{-(\varepsilon d)^2}$ (squared exponential kernel).

As underlined by several authors (see, e.g., [40]), purely maximizing the surrogate function may lead to converge to a point that is not the global maximum of the black-box function. To consider the fact that the surrogate and the true objective function differ from each other in an unknown way, the surrogate is typically augmented by an extra term that considers such an uncertainty. The resulting *acquisition function* is therefore maximized, instead, for generating a new sample of the optimization vector, trading off between seeking for a new vector where the surrogate is big and looking for regions of the feasible space of hyperparameters that have not yet been visited.
The acquisition function $a(x)$ used to select the query point x at each iteration is then constructed as a weighted sum of the surrogate and of an Inverse Distance Weighting (IDW) function that is used to promote exploration of the input space: $a(x) = \frac{\overline{f}}{\Delta\overline{f}} + \delta z(x)$, where $\Delta\overline{f}$ is a normalization factor, $\delta \geq 0$ is the exploration parameter, and $z(x)$ is an IDW function needed to promote exploration of the hyperparameter space. Possible choice for the IDW is:

$$z(x) = \begin{cases} 0, \ x \in \{x_1, \ \dots, \ x_M\}, \\ \tan^{-1}\left(\frac{1}{\sum_{j=1}^{M} w_j(x, \ x_j)}\right), \ otherwise \end{cases}$$

where $w_j(x, x_j) = \frac{1}{\|x - x_j\|^2}$. Clearly, $z(x) = 0$ for all inputs already tested, and $z(x)$ increases as the distances between x and the already tested inputs increases, thus promoting exploration towards the regions of the hyperparameter space where most of the uncertainty lies.

4.2.2 Reinforcement learning

Reinforcement learning [41] is a ML paradigm that simulates the process of learning through trial and error. In this approach, the swarm agents explore the space of hyperparameters taking various actions and receiving feedback in the form of rewards (i.e., the KPIs). By continually adjusting the hyperparameters based on this feedback, the agents learn to discover optimal solutions to complex problems. It's akin to a learning method inspired by how humans and animals learn from their experiences. This trial-and-error process allows agents to navigate the uncertainties of their environment, enabling them to adapt and find solutions that may not be immediately obvious but are ultimately the most effective in improving the KPIs over time.

4.3 Tool evaluation for swarm intelligence and ML

Finally, we briefly present the most promising tools that have been selected in the ACES project to begin with the prototypes of the presented hybrid ML and swarm intelligence model for action decision making.

4.3.1 NetLogo for agent-based modeling simulation

One of the most widely used free agent-based modeling (ABM) simulation platforms is NetLogo [44]. It has a good documentation, a mature code base that is actively maintained, and thus, many extensions appear on a regular basis.

NetLogo is very well known in the education of ABM and complex systems. Besides education, NetLogo has also been shown to be a sophisticated platform that can perform simulations involving several thousand of agents in feasible computation time [45][46]. The NetLogo homepage lists more than 3000 research publications from the last 10 years that have used NetLogo as an ABM simulation platform.

NetLogo offers an interactive user interface including an easy possibility for visualization to allow rapid prototyping. To perform mass simulations, it comes with the so-called BehaviorSpace, an easily configurable batch mode to configure any desired number of simulations runs with multiple parameter settings. The simulation results are logged to files and can then be post-processed with a tool of choice (R, Excel, etc.) for statistical evaluation. Additionally, NetLogo supports co-simulation offering interfaces to other programming languages such as Python [47] or R [48].

NetLogo uses a discrete scale for simulation time called ticks. Using so-called breeds NetLogo allows to implement different types of agents. They can interact either directly (based on proximity or their connection via a network topology), or indirectly (based on stigmergic information in the environment) [15].

4.3.2 Python for Machine Learning

Python is renowned for its exceptional flexibility, ease of use, and comprehensive capabilities when it comes to implementing ML algorithms. Its versatility shines through as it supports a wide range of

libraries and frameworks dedicated to machine learning, such as TensorFlow, PyTorch, scikit-learn, OpenFL.io, and more.

In the Python ecosystem, we have access to excellent open-source libraries for Bayesian optimization/learning and RL. For Bayesian optimization, libraries like GPyOpt and scikit-optimize provide robust and user-friendly solutions for RBFs generation and optimization tasks. When it comes to RL, frameworks like OpenAI's Gym and Stable Baselines offer comprehensive environments and pre-implemented algorithms for training agents. These libraries greatly expedite the development process and are suitable for most use cases.

However, Python's strength lies in its adaptability, and it allows us to tailor solutions to our exact needs. Should ACES demand more flexibility or specific customizations, Python's extensibility makes it possible to develop custom libraries and algorithms. This adaptability empowers developers to fine-tune the ML process, addressing unique challenges and requirements effectively. So, while Python offers exceptional open-source resources, it also offers the freedom to craft tailor-made solutions when the need arises, ensuring that we can tackle any machine learning task with precision and efficiency.

5 Conclusion

This deliverable reports the main technical achievements of the first year of the ACES project of WP4, focusing on agent-based components and their actions. Autopoiesis, a novel concept in this domain, has been central to our exploration. The deliverable delineates how these principles adapt to the edge-cloud continuum infrastructure, presenting a mapping of these concepts to relevant AI and ML techniques. Key highlights from the deliverable include:

Autopoietic Principles in Computing: Autopoiesis, originally a biological concept, has been successfully contextualized in computing for the ACES project. This approach helps to address the challenges of the edge-cloud continuum, demonstrating a unique application of self-sustaining systems in a technological framework.

The **initial version of the ACES action library** has been developed. This is a significant milestone in the project. This library forms the backbone for the various agent-based models and algorithms used in the project, providing a structured approach to executing tasks within the ACES architecture. The action library also addresses ACES's crucial and comprehensive **non-functional aspects**, ranging from network performance to security and privacy. In particular, it highlights how performance objectives are set to improve network functions, thereby enhancing system responsiveness and efficiency. Regarding security and privacy, ACES employs a comprehensive strategy that includes detecting intrusions within the network, strengthening container security, and creating safeguards against attacks on machine learning algorithms. These actions embody ACES's non-functional goals, aiming to optimize speed, intelligence, and resilience that ACES aspires to achieve.

Swarm Intelligence and ML Integration: Preliminary concepts highlight the effective combination of swarm agent algorithms and ML that can overcome some of these limitations inherent to these approaches. The main aspects from the ACES environment are modeled using a novel approach that identifies supply and demand swarm agents. Moreover, several swarm algorithms, such as hormone and ant colony algorithms, are adapted to the needs of this environment. On top of that, the ML component is proposed to apply bayesian reasoning technique to modify the swarm hyperparameters to further improve its performance based on the observed effect of its actions. This integration enables the ACES platform to leverage the whole collected knowledge, translating it into a set of reasoning components that address key management challenges within the edge-cloud infrastructure.

The deliverable concludes with a forward-looking perspective, identifying areas for future work: These include the continued refinement of the ACES action library, further integration of AI and ML techniques, and ongoing assessment of the system’s performance and scalability. In summary, the ACES project's first year has laid a solid foundation for future advancements in autopoietic computing within the edge-cloud continuum. The project has successfully demonstrated the potential of integrating biological principles like autopoiesis with advanced computing concepts, paving the way for innovative solutions in edge computing environments.

6 References

[1] Maturana, H. R., & Varela, F. J. (1980). Autopoiesis and Cognition: The Realization of the Living. D. Reidel Publishing Company.
[2] Briscoe, G., & Dini, P. (2010). Towards autopoietic computing. In Digital Ecosystems: Third International Conference, OPAALS 2010, Aracuju, Sergipe, Brazil, March 22-23, 2010, Revised Selected Papers 3, 199-212.
[3] Straussfogel, D., von Schilling, C. (2009). Systems Theory, International Encyclopedia of Human Geography, (pp 151-158). Elsevier.
[4] Schatten, M., & Bača, M. (2010). A critical review of autopoietic theory and its applications to living, social, organizational and information systems. Društvena istraživanja, 108(109), 4-5.
[5] Letelier, J. C., Marín, G., & Mpodozis, J. (2002). Computing with autopoietic systems. Soft Computing and Industry: Recent Applications, 67-80.
[6] Mingers, J. (1995). Self-producing systems: Implications and applications of autopoiesis. Plenum Press.
[7] IBM. (2005). An architectural blueprint for autonomic computing. IBM White Paper.
[8] Modha, D. S., et al. (2011). Cognitive computing. Communications of the ACM, 54(8), 62-71.
[9] Parashar, M., & Hariri, S. (2004). Autonomic computing: An overview. In International workshop on unconventional programming paradigms, 257-269.
[10] NASA. Information Technology And Software Autonomic Autopoiesis (GSC-TOPS-97), Patent. Retrieved from https://technology.nasa.gov/patent/GSC-TOPS-97 .
[11] Chen, M., Herrera, F., & Hwang, K. (2018). Cognitive computing: architecture, technologies and intelligent applications. IEEE Access, 6, 19774-19783.
[12] Demirkan, H., Earley, S., & Harmon, R. R. (2017). Cognitive computing. IT professional, 19(4), 16-20.
[13] Keenan, B. (2022). Niklas Luhmann: What is Autopoiesis? Retrieved from https://criticallegalthinking.com/2022/01/10/niklas-luhmann-what-is-autopoiesis/
[14] Escher, M.C.: Drawing hands (1989)
[15] Gheibi, O., Weyns, D., & Quin, F. (2021). Applying machine learning in self-adaptive systems: A systematic literature review. ACM Transactions on Autonomous and Adaptive Systems (TAAS), 15(3), 1-37.
[16] Ghahremani, S., Adriano, C. M. and Giese, H. (2018). Training prediction models for rule-based self-adaptive systems. In Proceedings of the IEEE International Conference on Autonomic Computing (ICAC'18). 187–192. https://doi.org/10.1109/ICAC.2018.00031
[17] G. Tesauro, G., Jong, N., Das, R., & Bennani, M. (2007). On the use of hybrid reinforcement learning for autonomic resource allocation. Cluster Computing. 10 (3), 287–299.
[18] Saputri, T. R. D., & Lee, S. W. (2020) The Application of Machine Learning in Self-Adaptive Systems: A Systematic Literature Review.
[19] Schranz, M., Umlauft, M., & Elmenreich, W. (2021). Bottom-up Job Shop Scheduling with Swarm Intelligence in Large Production Plants. In SIMULTECH (pp. 327-334).
[20] Schranz, M., Harshina, K., Forgacs, P., & Buining, F. (2024). Agent-based Modeling in the Edge Continuum using Swarm Intelligence. In ICAART (under review).
[21] Yu, L. and Liu, H. (2003). Feature selection for high-dimensional data: A fast correlation-based filter solution. In Proceedings of the 20th international conference on machine learning, pages 856–863.
[22] Khaire, U. M. and Dhanalakshmi, R. (2022). Stability of feature selection algorithm: A review. Journal of King Saud University-Computer and Information Sciences, 34(4):1060–1073.
[23] Aggarwal, C. C., Hinneburg, A., and Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional space. In Database Theory—ICDT 2001: 8th International Conference London, UK, January 4–6, 2001 Proceedings 8, pages 420–434. Springer.
[24] Benabdellah, A. C., Benghabrit, A., and Bouhaddou, I. (2019). A survey of clustering algorithms for an industrial context. Procedia computer science, 148:291–302.
[25] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark: Cluster computing with working sets. In 2nd USENIX Workshop on Hot Topics in Cloud Computing.
[26] Verbraeken, J., Wolting, M., Katzy, J., Kloppenburg, J., Verbelen, T., and Rellermeyer, J. S. (2020). A survey on distributed machine learning. ACM Computing Surveys (csur), 53(2):1–33.
[27] Pham, Q., Nguyen, D., Mirjalili, S., Hoang, D., Nguyen, D., Pathirana, P., and Hwang, W.-J. (2021). Swarm intelligence for next-generation networks: Recent advances and applications. Journal of Network and Computer Applications, 191:103141.

[28] Elmenreich, W., D'Souza, R., Bettstetter, C., & de Meer, H. (2009, December). A survey of models and design methods for self-organizing networked systems. In International Workshop on Self-Organizing Systems (pp. 37-49). Berlin, Heidelberg: Springer Berlin Heidelberg.
[29] Lin, Y.-H. and Hu, Y.-C. (2018). Residential consumer-centric demand-side management based on energy disaggregation-piloting constrained swarm intelligence: Towards edge computing. Sensors, 18(5):1365.
[30] Feng, S., Chen, Y., Zhai, Q., Huang, M., and Shu, F. (2021). Optimizing computation offloading strategy in mobile edge computing based on swarm intelligence algorithms. EURASIP Journal on Advances in Signal Processing, 7(36):1–24.
[31] Lee, C., Huo, Y., Zhang, S., and Ng, K. (2020). Design of a smart manufacturing system with the application of multi-access edge computing and blockchain technology. IEEE Access, 8:28659–28667.
[32] Mahenge, M., Li, C., and Sanga, C. (2022). Energy-efficient task offloading strategy in mobile edge computing for resource-intensive mobile applications. Digital Communications and Networks, 8(6):1048–1058.
[33] Bacanin, N., Antonijevic, M., Bezdan, T., Zivkovic, M., Venkatachalam, K., and Malebary, S. (2023). Energy efficient offloading mechanism using particle swarm optimization in 5g enabled edge nodes. Cluster Computing, 26:587–598.
[34] Attiya, I., Elaziz, M., Abualigah, L., Nguyen, T., and El-Latif, A. (2022). An improved hybrid swarm intelligence for scheduling iot application tasks in the cloud. IEEE Transactions on Industrial Informatics, 18(9):6264–6272.
[35] Singh, H., Bhasin, A., and Kaveri, P. R. (2021). Qras: Efficient resource allocation for task scheduling in cloud computing. SN Applied Sciences, 3:1–7.
[36] de Melo Menezes, B. A., Kuchen, H., and Buarque de Lima Neto, F. (2022). Parallelization of swarm intelligence algorithms: Literature review. International Journal of Parallel Programming, 50:1–29.
[37] Weerts, H. J., Mueller, A. C., & Vanschoren, J. (2020). Importance of tuning hyperparameters of machine learning algorithms. arXiv preprint arXiv:2007.07588.
[38] Cannelli, L., Zhu, M., Farina, F., Bemporad, A., & Piga, D. (2023). Multi-agent active learning for distributed black-box optimization. *IEEE Control Systems Letters*.
[39] Wu, Y., Wang, H., Zhang, B., & Du, K. L. (2012). Using radial basis function networks for function approximation and classification. *International Scholarly Research Notices*, *2012*.
[40] Bemporad, A. (2020). Global optimization via inverse distance weighting and radial basis functions. *Computational Optimization and Applications*, *77*(2), 571-595.
[41] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.
[42] Sobe, A., Elmenreich, W., Szkaliczki, T., & Böszörmenyi, L. (2015). SEAHORSE: Generalizing an artificial hormone system algorithm to a middleware for search and delivery of information units. Computer Networks, 80, 124-142.
[43] Elmenreich, W., Schnabl, A., & Schranz, M. (2021). An Artificial Hormone-based Algorithm for Production Scheduling from the Bottom-up. In *ICAART (1)* (pp. 296-303).
[44] Wilensky, U. (1999). Netlogo. http://ccl.northwestern.edu/netlogo/.
[45] Railsback, S., Ayllón, D., Berger, U., Grimm, V., Lytinen, S., Sheppard, C., and Thiele, J. C. (2017). Improving execution speed of models implemented in Netlogo. Journal of Artificial Societies and Social Simulation.
[46] Railsback, S. F. and Grimm, V. (2019). Agent-based and individual-based modeling: a practical introduction. Princeton university press, 2nd" edition.
[47] Gunaratne, C. and Garibay, I. (2021). NL4Py: Agent-based modeling in Python with parallelizable NetLogo workspaces. SoftwareX, 16:100801.
[48] Thiele, J. C. (2014). R marries NetLogo: introduction to the RNetLogo package. Journal of Statistical Software, 58:1–41.