Machine Learning for Aggregate Computing: a Research Roadmap

Machine Learning for Aggregate Computing: a Research Roadmap

Gianluca Aguzzi, Roberto Casadei, Mirko Viroli

ALMA MATER STUDIORUM—Università di Bologna, Cesena, Italy
Email: {gianluca.aguzzi, roby.casadei, mirko.viroli}@unibo.it

***Abstract*—Aggregate computing is a macro-approach for programming collective intelligence and self-organisation in distributed systems. In this paradigm, a single "aggregate program" drives the collective behaviour of the system, provided that the agents follow an execution protocol consisting of asynchronous sense-compute-act rounds. For actual execution, a proper aggregate computing middleware or platform has to be deployed across the nodes of the target distributed system, to support the services needed for the execution of applications. Overall, the engineering of aggregate computing applications is a rich activity that spans multiple concerns including designing the aggregate program, developing reusable algorithms, detailing the execution model, and choosing a deployment based on available infrastructure. Traditionally, these activities have been carried out through ad-hoc designs and implementations tailored to specific contexts and goals. To overcome the complexity and cost of manually tailoring or fixing algorithms, execution details, and deployments, we propose to use machine learning techniques, to automatically create policies for applications and their management. To support such a goal, we detail a rich research roadmap, showing opportunities and challenges of integrating aggregate computing and learning.**

***Index Terms*—aggregate computing, machine learning, multi-agent learning, self-organising systems**

I. INTRODUCTION AND BACKGROUND

Aggregate Computing (AC) [1]–[3] is an approach for programming *Collective Intelligence (CI)* [4], [5] in distributed systems, having its roots in field-based coordination [2], [6] and spatial computing [7]. AC is based on a macro-programming model [7]–[9] whereby the collective behaviour of a system of neighbour-interacting agents is expressed through a single "aggregate program" specifying both data processing and data exchange by a global perspective. Aggregate programs are written using AC languages, such as ScaFi [10], that are implementations based on the *field calculus* [2], a core language which provides the means for formal study of AC and aggregate programs. Indeed, aggregate programs express global, adaptive computations in terms of *fields*, namely distributed data structures that associate a value to each device over time. For collective adaptive behaviour to unfold, the full program has to be played by all the agents of the system through asynchronous *sense-compute-act* rounds, hence somewhat mimicking self-organisation processes in natural systems [11]. That is, AC builds on a flexible, best-effort *aggregate execution model* whereby every device in the system repeatedly (i) senses its local context, for environment data and incoming messages from neighbours; (ii) evaluates the aggregate program against its local context, and (iii) acts upon its local context as prescribed by the program by running actuations and sending a coordination message to neighbours. For actual execution, a proper AC middleware or platform has to be deployed across the nodes of the target distributed system, to support the services needed for the execution of applications [12], [13]. It has been shown that the approach is especially effective for programming large-scale homogeneous distributed systems and multi-agent systems such as crowds of device-equipped humans [1], robot swarms [3], sensor-actuator networks [14], and smart cities [12]. The major benefits of AC as a programming approach include (i) reliance on formal models; (ii) *declarativity*, by abstracting over several execution details; and (iii) abstraction of collective behaviours into reusable functions of fields and *compositionality*. The reader can refer to [2] for an overall survey of AC research and a discussion on its relationship with related works across the fields of coordination [15], spatial computing [7], ensemble computing [16], collective adaptive systems [17].

In summary, the engineering of AC applications is a rich activity that spans multiple concerns including designing the aggregate program, developing reusable algorithms [14], [18], detailing the execution model [19], and choosing a deployment based on available infrastructure [12] (see Figure 1 for the full AC "stack"). Traditionally, these activities have been carried out through ad-hoc designs and implementations created by developers and tailored to specific contexts and goals, leading to, e.g., self-organisation algorithms that are very reactive under certain network assumptions [14], or round execution frequencies tailored to the velocity of change of underlying environment phenomena [19]. To overcome the complexity and cost of manually tailoring or devising general but inefficient algorithms, execution details, and deployments, we propose to use *Machine Learning (ML) techniques*. In particular, we observe that automated design driven by learning can be applied at different levels of the AC "engineering stack". The integration of AC and ML, which we refer to as Aggregate Computing + Machine Learning (AC+ML), is expected to provide a lot of opportunities and challenges, fostering a long-term record of research contributions. Accordingly, in this paper, we provide and discuss a research roadmap for achieving the vision of AC+ML.

II. Motivating Example: an Internet of Things (IoT) system for Monitoring & Control

Consider an IoT system comprising a sensor network and other mobile nodes for the monitoring of hazardous situations in a natural park, e.g., wildfires [20]. In this case, the nodes should constantly assess local environmental data (e.g., humidity, temperature, etc.) and collaborate to collectively perceive macro-level phenomena. This may involve coordinating and achieving a consensus regarding what geographical zones are at risk or currently experiencing wildfires. Such collective perceptions could then be used to issue warnings and promote intervention. Indeed, using only point-wise perceptions could incur in inaccurate warnings and false alarms [20], due to the possible presence of local noise, variation, and sensor failures. So, the nodes may use self-organising distributed algorithms such as information flows [21], gradients [18], and dynamic area formation [22], to achieve the desired global behaviour. However, such algorithms usually admit diverse implementations, based on different assumptions on the network structure, reliability, and dynamics, and they may also have different associated energy and cost profiles. Additionally, the system should be able to opportunistically self-manage itself and its execution to promote desired non-functional properties like reactivity, efficiency, and resilience. Addressing all these details manually can be very complex. A way to mitigate this issue might lie in using approaches and methodologies that combine manual design and automatic design—since the latter, using methods like those of *soft computing* [23], can help to achieve tractability of complex problems and maintainability of solutions.

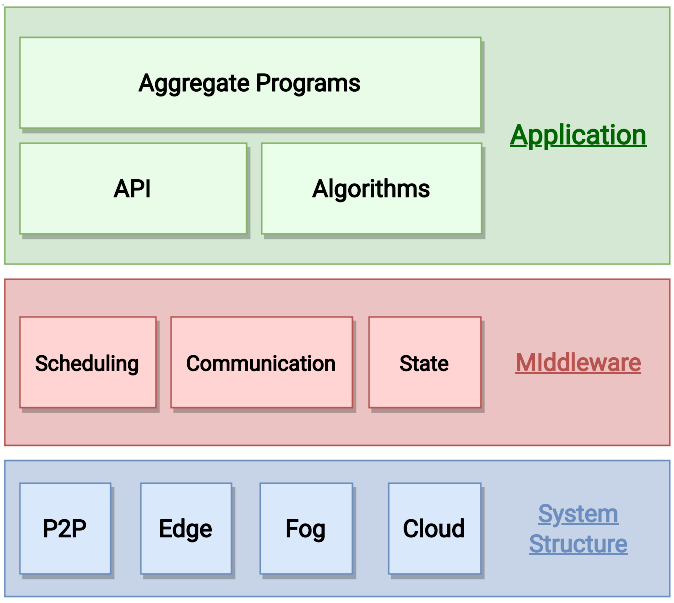

Fig. 1. The entire Aggregate Computing stack. The application level (composed of API, algorithms and programs) needs to be supported by an execution platform/middleware, to decouple the application from systems structures. Then, this execution platform should be deployed in a particular architecture. In our vision, Machine Learning could enhance all of these layers.

III. Machine Learning for Aggregate Computing: a Research Roadmap

This section details motivation and directions for AC+ML research. A summary is provided Figure 2. This section builds on AC background; for a comprehensive introduction of AC, the reader can refer to [2]. First, we specify goals and means to achieve CI in aggregate computing (Section III-A); then, we focus on the possibility of applying learning to learn algorithms (Section III-B), execution strategies (Section III-C), and system structures (Section III-D).

A. Goals and Means

To systematically analyse the ways in which ML can promote the development of AC applications, it is important to consider the *goals* and *means* of the AC framework.

The goals include:

1) *functionality* — achieving some collective behaviour (e.g., environment monitoring and control through sensor-actuator networks [12], [19], and matching and coordinating collective tasks with worker ensembles [3]);
2) *non-functionality* — concerning with the cost associated to the functionality.

In particular, the latter can be further divided into multiple sub-goals from which application-specific *trade-offs* can be made:

1) *time efficiency*: refers to the time needed to converge to the desired state-of-affairs;
2) *communication efficiency*: refers to the amount of communication performed (e.g., measured in terms of messages or bandwidth);
3) *execution efficiency*: refers to the number of rounds of computations performed;
4) *energy efficiency*: e.g. by combining communication and execution efficiency;
5) *dependability*: concerns e.g. reliability or safety of a collective and its products.

In the AC approach, these goals can be addressed through three main means:

1) algorithms;
2) execution strategy;
3) system structure (deployment).

Now, it turns out that ML could be a powerful technique to replace or augment those traditionally human-engineered means.

B. Learning AC algorithms

AC algorithms take collective inputs and use computational mechanisms to produce collective outputs. In the *field calculus* [2], the minimal formal framework that underpins AC, collective inputs and outputs are denoted by *computational fields* (or *fields* for short), namely distributed data structures mapping each device of the aggregate system to a value. The field calculus, then, provides a set of operators for manipulating fields: essentially, operators specifying how fields

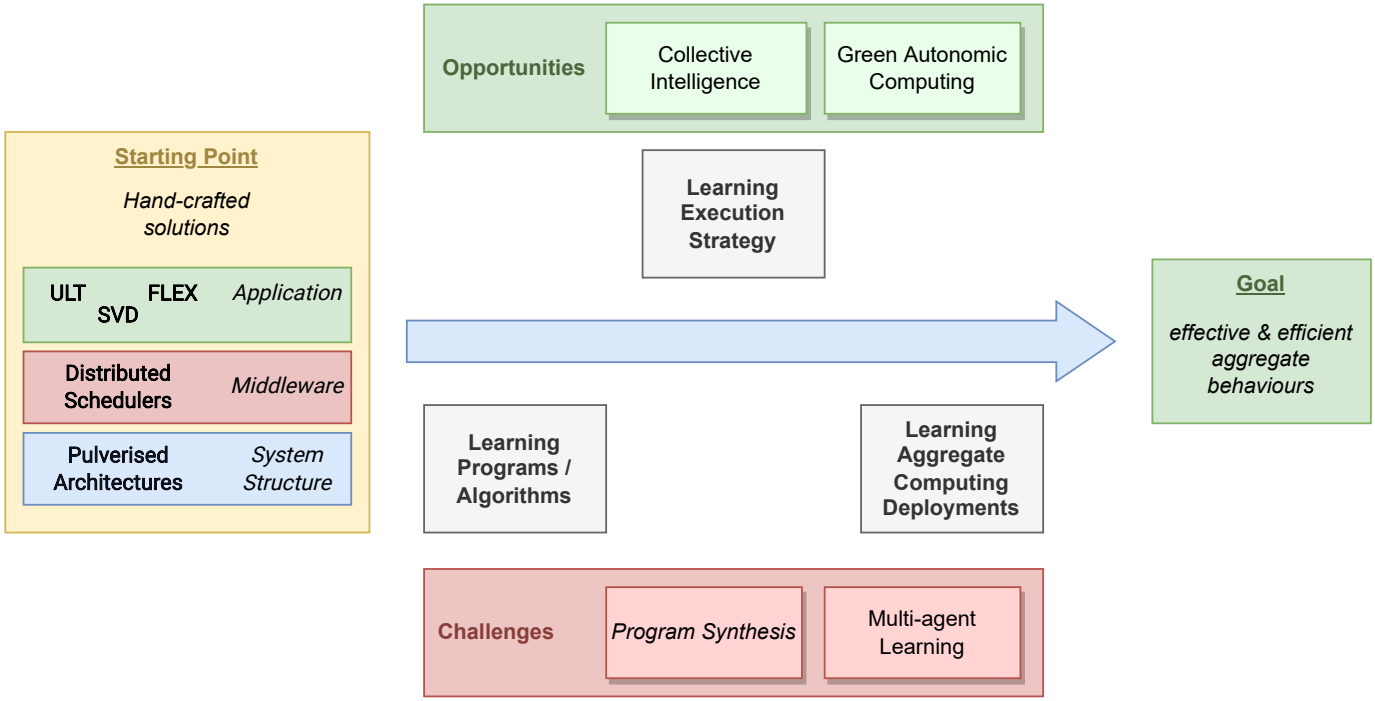

Fig. 2. Overview of the research roadmap aiming at efficient and effective aggregate computations. The starting point is current research, based on hand-crafted solutions. The goal is addressed through application of ML at the program/execution/deployment levels.

evolve over time (round after round), and operators specifying interactions with neighbours (which can be reified through *neighbouring fields*). The field calculus is implemented by so-called *aggregate programming languages* such as ScaFi [10], a Scala internal Domain-Specific Language (DSL). So, in AC, a collective adaptive behaviour is the result of an algorithm (a function from fields to fields) expressed e.g. in ScaFi and the concrete execution of the algorithm in a system of devices.

A typical example is the *gradient algorithm* [18], which is essentially a function mapping a Boolean field of sources (i.e. the devices where the field is *true*) to the field of minimum distances from those sources. Usually, algorithms are *progressive* – they take time (computation rounds and communications) to converge to the "correct" value – and *self-healing*, i.e., they can adjust their output following changes in their inputs and the system topology. For instance, a gradient algorithm progressively corrects the field of distances after the set of sources change, or nodes move (hence changing the distances between neighbour nodes) or enter/leave the system.

So, different gradient algorithms may achieve the same functionality (i.e., the eventual computation of the field of minimum distances from sources) with different non-functional outcomes. Indeed, they may: (i) take different time or a different number of rounds to converge, for the same initial condition; (ii) require a different amount of data to be exchanged; (iii) take different trade-offs e.g. regarding reactivity and smoothness (cf. the stability of values during change) [14], [18]; or (iv) take different assumptions regarding the execution model or the environment [14], which may affect applicability.

Designing efficient and versatile AC algorithms can be complex [14], [18]: therefore, it is interesting to explore whether algorithms can be learnt or synthesised given high-level functional goals. Program synthesis [24] goes in this direction. It consists of a set of techniques in which a model explicitly generates programs from a high-level specification. Particularly, these research areas gains interest in the last years, when novel techniques consider using as model neural networks (e.g. GitHub Copilot / Codex [25]). In our case, an idea could be to combine the program synthesis technique called *sketching* [26] with machine learning. With this approach, the designer could provide an algorithm template with holes corresponding to actions to be learnt by the agents or the whole collective, e.g., through reinforcement learning or/and evolutionary algorithms. In this case, learning would be used to *search* for an optimal policy. The resulting algorithm, then, would need extensive testing (e.g. by simulation) in a representative set of environments and dynamics. First efforts in this direction exploit Multi-Agent Reinforcement Learning (MARL) [27], [28] (i.e., learning algorithms where multiple agents learn a distributed policy through interaction with the environment) approaches to learn an improved gradient algorithm that self-heals faster than the classical baseline solution [29]. However, research is needed to identify what synthesis techniques, learning algorithms, frameworks, and methodologies can support the learning of algorithms able to achieve performance similar to or better than state-of-the-art solutions.

C. Learning execution strategies and adaptations

For a given AC program or algorithm, multiple execution strategies can be applied, affecting aspects like the scheduling of computation rounds, the scheduling of communications, the retention of messages from neighbours. In particular, a first distinction can be made between *static* and *dynamic* execution strategies. The latter approaches adapt the execution choices at runtime depending on factors which may include the speed of environmental change, the energy level of a device, incentives in volunteering settings, or the desired Quality of Service (QoS). Moreover, these factors may be diverse in diverse portions of the system; so, it is in general important to also consider the local context of each device or set of devices.

Note that adaptive behaviour could be achieved via static execution strategies, e.g., by using reactive approaches triggering behaviours when specific context conditions apply [19]. However, again, since it is in general hard to design static or dynamic execution strategies able to adequately take into account all the factors and goals, it could make sense to let a system (and its components) learn how to efficiently execute algorithms according to a set of given high-level objectives. Indeed, true adaptiveness comes from changing the behavioural rules, and learning is a premier tool for changing for the better. In this context, also bio-inspired optimisation techniques, such as evolutionary computing [30], could be leveraged to devise controllers that optimise for multiple objectives, e.g., related to a set of QoS metrics. This is a long-standing approach in swarm robotics, lately called automatic design. The emphasis on improving efficiency by optimising execution of aggregate systems, hence promoting sustainability of collective computations, could be the opportunity to open up a vision of *green autonomic computing*.

D. Learning system structures and re-structuring

A logical AC system consists of a logical network of logical devices operating as per the aforementioned execution protocol. It is the *collective digital twin* [31] of a target set of application-level physical devices (e.g., robots of a swarm, or workers in a computing ecosystem). As shown in recent work on *pulverised architectures* [12], it turns out that different application partitioning schemas and implementations of the *digital thread* associated with the aggregate system are possible, as well as different deployments of application components onto the available Information-Communication Technology (ICT) infrastructure. For instance, it is possible to embed evaluations of the aggregate program into the devices themselves, to move the entire computational part on the cloud (leaving devices as thin hosts dealing only with sensing and actuation), or spread these onto a layer of edge-fog infrastructural devices. Different deployments may lead to different efficiency trade-offs and non-functional outcomes [12], [31], which, crucially, may also change dynamically due to application and infrastructure dynamics (cf. addition or removal of new nodes, blackouts, etc.).

In previous research, aggregate application partitioning and deployment have been done manually at design time [12]. However, for a given set of infrastructures, it is not easy to determine an effective mapping. The usual approach consists of manually generating different deployments, simulating these for the same set of applications, collecting various cost metrics, and evaluating results to determine trade-offs and guidelines. However, ML could be injected into such a methodology to have the system learn by itself what is a (locally) optimal deployment for an aggregate application. Moreover, the system could be induced to learn a strategy to self-adapt the deployment (i.e., by moving components opportunistically across the ICT infrastructure) while trying to preserve, e.g., certain QoS targets.

Additionally, it has been shown in [31], that changing the logical structure of an aggregate system at runtime (e.g., by injecting *virtual* devices) could be a further means for steering self-organisation processes, namely to improve the collective behaviour of a system (cf. [32]). In this respect, a challenge would be to determine how, when, and where virtual devices should be spawned or removed from the system. In the AC+ML vision, this problem should not be addressed through ad-hoc solutions, but the AC system should be trained in order to learn the best strategies for improving the efficacy and efficiency of aggregate applications.

IV. Opportunities and Challenges

The research delineated in Section III is rich in opportunities and challenges—the most significant ones discussed in the following.

Opportunities

A prominent opportunity of AC+ML research lies in potentially getting insights about the *automatic design of Collective Intelligence (CI)*, renewing the partial contributions given by Szuba's computational collective intelligence [4] and Tumer and Wolpert's COIN (COllective INtelligence) [5]. However, unlike previous work, the peculiar characteristic of aggregate computing of reifying CI into macro-level *programs* (which we may refer to as *CI programmability*) is expected to enable a synergic and gentle introduction of automatic design and learning. Additionally, the other crucial aspect of functional *compositionality* of aggregate behaviours (denoted by functions operating on fields), is also expected to help, e.g., by fostering learning processes whereby the goal is to find suitable compositions of elementary collective behaviours. Moreover, the ability to change execution strategies while guaranteeing the same behaviour could also be considered a form of CI.

Indeed, another key opportunity lies in the possibility of fostering *efficiency* in large-scale intelligent systems, which is more and more important for sustainability as advocated by important fields like *green computing* [33]. The significance of the problem is especially relevant nowadays because of the tension between the visions of pervasive computing [34], future-generation large-scale computational collectives [3], and autonomic computing (promoting smarter – i.e. more com-

putationally intensive – devices) [35] and the urge of limiting the impact of humans and technologies on the environment.

Technical Challenges

Applying learning through the AC stack (Figure 1) poses several challenges, many of which are implied by the nature of aggregate systems—cf. distribution, decentralisation, partial observability, many-agents coordination, and the eventual nature of collective computations,

Particularly, learning in *many*-agent networked system [36] is currently an open challenge. Indeed, extending the learning from one agent to many agents exponentially enlarges the policy search space, due to the combinatorial nature of multi-agent systems [28]. Consider the case study presented in Section II: to collectively detect wildfires, the system should be composed of hundreds of smart sensors/drones spread on a possibly large geographical zone. If each device performs learning to find a good policy for the given problem, the overall policy search space grows exponentially with the number of agents, which leads the problem to be completely intractable. A recurrent solution in MARL for such kind of situation is to consider the agents' policies homogeneous [37] in order to drastically reduce the policy search space. Moreover, the neighbouring-based system structure is not strict and could evolve in time, leading to time- and space-varying input spaces. Furthermore, the agents should consider that some information are perceived in a previously supervised zone (i.e. space varying), therefore they could be outdated since the nodes could be moved to another area far away from the previous one. This complexity is particularly challenging to handle ML methodologies, even if the novel neural models (such as RNN [38] and GNN [39]) aim at handling both space and time-varying information.

In many-agent systems, it is not appropriate to use a centralised controller that orchestrates the system as a whole, due to the typical large scale and resilience requirements. However, using only a local vision of the system could lead to the problem of *non-stationarity*, since each agent concurrently learns and modifies the environment in the eye of the other agents [40]. Another concern related to many-agent settings is the *multi-agent credit assignment* problem [41]. This is referred to in the difficulty of deriving local reward policy from a global utility that measures the system as a whole [41], [42]. In the example of wildfires, consider the condition in which the system correctly identifies a hazard/fire situation. This behaviour must be considered correct at the collective level, but it could be that some nodes have taken "incorrect" actions (e.g. moving away from a dangerous situation), and these should be penalised. However, especially when considering emergent dynamics, it is not easy to precisely evaluate what micro-level activities led to the collective good and which not. This problem is historical, treated partly in COIN approaches with difference rewards [43] and later in COMA [44], a recent MARL and Deep Learning methodology to derive local agent policies in cooperative settings. Besides, the reward received is typically very delayed and sparse in time, because an action taken in a point of large geographical space, could lead to an influence on the whole system only when it reaches all nodes.

These challenges are mitigated by an increasing track of research records on AC [2], the availability of formal tools for analysing and reasoning about aggregate computations and systems [2], and the support given by tools for developing and simulating aggregate systems [2].

V. Conclusion

Developing AC applications requires addressing various algorithmic, computational, system, and deployment concerns. The integration of ML across the AC development stack (cf. Figure 1) provides opportunities and challenges requiring significant research (cf. Figure 2). Indeed, AC research seems a fertile terrain for exploring techniques combining self-adaptive/self-organising computing with machine learning (especially MARL)—and its synergy with other research ideas in areas like soft computing and program synthesis. Therefore, the roadmap for AC+ML delineated in this paper is expected to provide results and insights on the engineering of collectively intelligent distributed systems.

Acknowledgements

This work has been supported by the EU/Italian MUR FSE REACT-EU PON R&I 2014-2022 (CCI2014IT16M2OP005).

References

[1] J. Beal, D. Pianini, and M. Viroli, "Aggregate programming for the internet of things," *Computer*, vol. 48, no. 9, pp. 22–30, 2015. [Online]. Available: https://doi.org/10.1109/MC.2015.261

[2] M. Viroli, J. Beal, F. Damiani, G. Audrito, R. Casadei, and D. Pianini, "From distributed coordination to field calculus and aggregate computing," *J. Log. Algebraic Methods Program.*, vol. 109, 2019. [Online]. Available: https://doi.org/10.1016/j.jlamp.2019.100486

[3] R. Casadei, M. Viroli, G. Audrito, D. Pianini, and F. Damiani, "Engineering collective intelligence at the edge with aggregate processes," *Eng. Appl. Artif. Intell.*, vol. 97, p. 104081, 2021. [Online]. Available: https://doi.org/10.1016/j.engappai.2020.104081

[4] T. Szuba, "A formal definition of the phenomenon of collective intelligence and its IQ measure," *Future Gener. Comput. Syst.*, vol. 17, no. 4, pp. 489–500, 2001. [Online]. Available: https://doi.org/10.1016/S0167-739X(99)00136-3

[5] D. H. Wolpert and K. Tumer, "Collective intelligence, data routing and braess' paradox," *J. Artif. Intell. Res.*, vol. 16, pp. 359–387, 2002. [Online]. Available: https://doi.org/10.1613/jair.995

[6] M. Mamei, F. Zambonelli, and L. Leonardi, "Co-fields: A physically inspired approach to motion coordination," *IEEE Pervasive Comput.*, vol. 3, no. 2, pp. 52–61, 2004. [Online]. Available: https://doi.org/10.1109/MPRV.2004.1316820

[7] J. Beal, S. Dulman, K. Usbeck, M. Viroli, and N. Correll, "Organizing the aggregate: Languages for spatial computing," in *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*. IGI Global, 2013, pp. 436–501.

[8] R. Casadei, "Macroprogramming: Concepts, state of the art, and opportunities of macroscopic behaviour modelling," *CoRR*, vol. abs/2201.03473, 2022. [Online]. Available: https://arxiv.org/abs/2201.03473

[9] R. Newton, G. Morrisett, and M. Welsh, "The regiment macroprogramming system," in *Proceedings of the 6th International Conference on Information Processing in Sensor Networks, IPSN 2007*. ACM, 2007, pp. 489–498. [Online]. Available: https://doi.org/10.1145/1236360.1236422

[10] R. Casadei, M. Viroli, G. Audrito, and F. Damiani, "FScaFi : A core calculus for collective adaptive systems programming," in *Leveraging Applications of Formal Methods, Verification and Validation: Engineering Principles - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part II*, ser. Lecture Notes in Computer Science, T. Margaria and B. Steffen, Eds., vol. 12477. Springer, 2020, pp. 344–360. [Online]. Available: https://doi.org/10.1007/978-3-030-61470-6_21

[11] F. Yates, *Self-Organizing Systems: The Emergence of Order*, ser. Life Science Monographs. Springer US, 2012. [Online]. Available: https://books.google.it/books?id=IiTvBwAAQBAJ

[12] R. Casadei, D. Pianini, A. Placuzzi, M. Viroli, and D. Weyns, "Pulverization in cyber-physical systems: Engineering the self-organizing logic separated from deployment," *Future Internet*, vol. 12, no. 11, p. 203, 2020. [Online]. Available: https://doi.org/10.3390/fi12110203

[13] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for internet of things: A survey," *IEEE Internet Things J.*, vol. 3, no. 1, pp. 70–95, 2016. [Online]. Available: https://doi.org/10.1109/JIOT.2015.2498900

[14] G. Audrito, R. Casadei, F. Damiani, D. Pianini, and M. Viroli, "Optimal resilient distributed data collection in mobile edge environments," *Comput. Electr. Eng.*, vol. 96, no. Part, p. 107580, 2021. [Online]. Available: https://doi.org/10.1016/j.compeleceng.2021.107580

[15] F. Damiani and O. Dardha, Eds., *Coordination Models and Languages - 23rd IFIP WG 6.1 International Conference, COORDINATION 2021, Held as Part of the 16th International Federated Conference on Distributed Computing Techniques, DisCoTec 2021, Valletta, Malta, June 14-18, 2021, Proceedings*, ser. Lecture Notes in Computer Science, vol. 12717. Springer, 2021. [Online]. Available: https://doi.org/10.1007/978-3-030-78142-2

[16] T. Bures, I. Gerostathopoulos, P. Hnetynka, F. Plasil, F. Krijt, J. Vinárek, and J. Kofron, "A language and framework for dynamic component ensembles in smart systems," *Int. J. Softw. Tools Technol. Transf.*, vol. 22, no. 4, pp. 497–509, 2020. [Online]. Available: https://doi.org/10.1007/s10009-020-00558-z

[17] R. D. Nicola, S. Jähnichen, and M. Wirsing, "Rigorous engineering of collective adaptive systems: special section," *Int. J. Softw. Tools Technol. Transf.*, vol. 22, no. 4, pp. 389–397, 2020. [Online]. Available: https://doi.org/10.1007/s10009-020-00565-0

[18] G. Audrito, R. Casadei, F. Damiani, and M. Viroli, "Compositional blocks for optimal self-healing gradients," in *11th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2017*. IEEE Computer Society, 2017, pp. 91–100. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/SASO.2017.18

[19] D. Pianini, R. Casadei, M. Viroli, S. Mariani, and F. Zambonelli, "Time-fluid field-based coordination through programmable distributed schedulers," *Log. Methods Comput. Sci.*, vol. 17, no. 4, 2021. [Online]. Available: https://doi.org/10.46298/lmcs-17(4:13)2021

[20] O. M. Bushnaq, A. Chaaban, and T. Y. Al-Naffouri, "The role of uav-iot networks in future wildfire detection," *IEEE Internet Things J.*, vol. 8, no. 23, pp. 16 984–16 999, 2021. [Online]. Available: https://doi.org/10.1109/JIOT.2021.3077593

[21] T. D. Wolf and T. Holvoet, "Designing self-organising emergent systems based on information flows and feedback-loops," in *Proceedings of the First International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2007, Boston, MA, USA, July 9-11, 2007*. IEEE Computer Society, 2007, pp. 295–298. [Online]. Available: https://doi.org/10.1109/SASO.2007.16

[22] D. Pianini, R. Casadei, M. Viroli, and A. Natali, "Partitioned integration and coordination via the self-organising coordination regions pattern," *Future Gener. Comput. Syst.*, vol. 114, pp. 44–68, 2021. [Online]. Available: https://doi.org/10.1016/j.future.2020.07.032

[23] D. Ibrahim, "An overview of soft computing," *Procedia Computer Science*, vol. 102, pp. 34–38, 2016. [Online]. Available: https://doi.org/10.1016/j.procs.2016.09.366

[24] S. Gulwani, O. Polozov, and R. Singh, "Program synthesis," *Found. Trends Program. Lang.*, vol. 4, no. 1-2, pp. 1–119, 2017. [Online]. Available: https://doi.org/10.1561/2500000010

[25] M. Chen, J. Tworek, H. Jun *et al.*, "Evaluating large language models trained on code," *CoRR*, vol. abs/2107.03374, 2021. [Online]. Available: https://arxiv.org/abs/2107.03374

[26] A. Solar-Lezama, *Program synthesis by sketching*. University of California, Berkeley, 2008.

[27] L. Busoniu, R. Babuska, and B. D. Schutter, "Multi-agent reinforcement learning: A survey," in *Ninth International Conference on Control, Automation, Robotics and Vision, ICARCV 2006, Singapore, 5-8 December 2006, Proceedings*. IEEE, 2006, pp. 1–6. [Online]. Available: https://doi.org/10.1109/ICARCV.2006.345353

[28] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "A survey and critique of multiagent deep reinforcement learning," *Auton. Agents Multi Agent Syst.*, vol. 33, no. 6, pp. 750–797, 2019. [Online]. Available: https://doi.org/10.1007/s10458-019-09421-1

[29] G. Aguzzi, R. Casadei, and M. Viroli, "Towards reinforcement learning-based aggregate computing," in *Coordination Models and Languages - 24th 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*, ser. Lecture Notes in Computer Science, M. H. ter Beek and M. Sirjani, Eds., 2022, accepted for publication.

[30] B. Li, J. Li, K. Tang, and X. Yao, "Many-objective evolutionary algorithms: A survey," *ACM Comput. Surv.*, vol. 48, no. 1, pp. 13:1–13:35, 2015. [Online]. Available: https://doi.org/10.1145/2792984

[31] R. Casadei, D. Pianini, M. Viroli, and D. Weyns, "Digital twins, virtual devices, and augmentations for self-organising cyber-physical collectives," *Applied Sciences*, vol. 12, no. 1, 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/1/349

[32] W. Li and W. Shen, "Swarm behavior control of mobile multi-robots with wireless sensor networks," *J. Netw. Comput. Appl.*, vol. 34, no. 4, pp. 1398–1407, 2011. [Online]. Available: https://doi.org/10.1016/j.jnca.2011.03.023

[33] S. Sarkar and S. Misra, "Theoretical modelling of fog computing: a green computing paradigm to support iot applications," *IET Networks*, vol. 5, no. 2, pp. 23–29, 2016. [Online]. Available: https://doi.org/10.1049/iet-net.2015.0034

[34] D. Saha and A. Mukherjee, "Pervasive computing: A paradigm for the 21st century," *Computer*, vol. 36, no. 3, pp. 25–31, 2003. [Online]. Available: https://doi.org/10.1109/MC.2003.1185214

[35] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003. [Online]. Available: https://doi.org/10.1109/MC.2003.1160055

[36] K. Zhang, Z. Yang, and T. Basar, "Decentralized multi-agent reinforcement learning with networked agents: recent advances," *Frontiers Inf. Technol. Electron. Eng.*, vol. 22, no. 6, pp. 802–814, 2021. [Online]. Available: https://doi.org/10.1631/FITEE.1900661

[37] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Auton. Agents Multi Agent Syst.*, vol. 11, no. 3, pp. 387–434, 2005. [Online]. Available: https://doi.org/10.1007/s10458-005-2631-2

[38] D. Rumelhart, G. Hinton, and R. Williams, "Learning internal representations by error propagation," in *Readings in Cognitive Science*. Elsevier, 1988, pp. 399–421. [Online]. Available: https://doi.org/10.1016/b978-1-4832-1446-7.50035-2

[39] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009. [Online]. Available: https://doi.org/10.1109/TNN.2008.2005605

[40] K. Tuyls and G. Weiss, "Multiagent learning: Basics, challenges, and prospects," *AI Mag.*, vol. 33, no. 3, pp. 41–52, 2012. [Online]. Available: https://doi.org/10.1609/aimag.v33i3.2426

[41] R. S. Sutton and A. G. Barto, *Reinforcement learning - an introduction*, ser. Adaptive computation and machine learning. MIT Press, 1998.

[42] A. K. Agogino and K. Tumer, "Unifying temporal and structural credit assignment problems," in *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*. IEEE Computer Society, 2004, pp. 980–987. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/AAMAS.2004.10098

[43] K. Tumer, A. K. Agogino, and D. H. Wolpert, "Learning sequences of actions in collectives of autonomous agents," in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems part 1 - AAMAS '02*. ACM Press, 2002. [Online]. Available: https://doi.org/10.1145/544741.544832

[44] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," *CoRR*, vol. abs/1705.08926, 2017. [Online]. Available: http://arxiv.org/abs/1705.08926