AI4PM: Distributed and Intelligent Programmable Matter

Benoît Piranda, Mohammad Ali Nemer, Abdallah Makhoul and Julien Bourgeois

Abstract *Programmable Matter* (PM) is composed of materials that can be programmed to modify their physical characteristics, such as alterations in shape and other properties. This paper proposes the implementation of an Artificial Neural Network (ANN) on a modular robot and PM system in order to enhance its computational and self-reconfiguration capabilities. The idea is to imbue these systems with computational intelligence, enabling it to adapt itself dynamically according to its internal constraints and/or its surrounding environment. Indeed, programmable matter is a distributed system that suffers from several challenges, like high number of modules, limited energy and computational capabilities, storage and communication. Deploying Artificial Intelligence (AI) techniques directly on modules will increase the ability of the systems to autonomously and dynamically optimize the use of the available resources. However, most of existing AI solutions need high computational capabilities and rely on centralized servers which are not suitable for PM systems with limited resources. In this paper, we study the feasibility of fusing AI and PM by implementing a distributed artificial neural network model directly on PM systems. The objective of this model is to let the system dynamically and in a distributed manner to determine its current shape. Our approach is demonstrated both by simulations and by implementation on a real *Blinky Blocks* platform. The conducted experimentation shows that the results are exactly the same as an ANN executed in a centralized manner.

1 Introduction

Programmable Matter is a matter that can be reprogrammed to modify its physical properties or transform itself on demand or dynamically in response to its surround-

Benoit Piranda, Mohammad Ali Nemer, Abdallah Makhoul, Julien Bourgeois
Université Marie et Louis Pasteur, CNRS, institut FEMTO-ST, 25200 Montbéliard, France.
e-mail: first.name@femto-st.fr

ing environment. When implemented as a modular robot, it is composed of a high number of autonomous and independent connected elements called modules or particles, whose connections to one another form the overall shape of the system. In addition to sensing, processing, and communication abilities, each module possesses actuation and mobility features enabling it to dynamically alter the shape of the system through the rearrangement of connections between its constituent modules. This is known as the self-reconfiguration problem [1, 2]. Modular robots implementing programmable matter tends to be small and have, therefore, limited capacities like it is the case for the memory space. Indeed, it is often supposed that the different modules are homogeneous, having low computational, memory and energetic resources due to space constraints, and they lack global configuration knowledge, hindering individually decision-making. Therefore, introducing computational intelligence capabilities to *Programmable Matter* systems can improve their functionality and self-configuration. This can be achieved by embedding Artificial Intelligence (AI) and neural networks techniques within modular robots.

In the past decade, AI methods have seen widespread application across a multitude of tasks, such as regression, prediction, classification, etc., in various domains. Several models and algorithms have been proposed and used in several fields including mathematics, statistics, computer science and computer vision [3, 4, 5]. However, the majority of these methods has been developed for centralized computing platforms as they require heavy computational and storage resources [6] and therefore, they are not suitable for scarce resources. Most existing AI solutions for these systems are based on edge computing [7, 8], where the nodes send their compressed or aggregated data to a central point to do the computing and wait for the results or the decisions [9, 10]. These solutions suffer from several drawbacks and are not suitable for PM. First, the PM system is fully distributed and it is not easy to connect it to a central point or to the edge. Second, the amount of communication and transmission messages will increase in the system leading to battery depletion. Third, the data latency will increase and the system must wait for the edge to respond. Therefore, embedding neural networks within PM would allow the system to execute AI techniques locally without the need of centralization. However, the main challenge of implementing AI techniques on PM is the computation and memory limitation of resource constrained modules (particles).

In the literature, one can find some works which are dedicated to implementing neural networks models directly on resource-constrained embedded IoT or mobile devices [11, 12, 13]. For instance, in [14] the authors explore various distributed neural network architectures, analyzing the amount of communication and storage usage. They introduce centralized, horizontal, and vertical decomposition methods to implement distributed neural networks within a Wireless Sensor Network (WSN). The authors in [15] propose a technique to optimize neural network models by reducing the number of connections between neurons while maintaining or improving performance compared to fully connected networks. This approach facilitates the discovery of smaller Artificial Neural Network (ANN) models that offer comparable performance to fully connected models. [13] introduces a decentralized architecture for deploying Distributed Artificial Intelligence on Internet of Things (IoT) hard-

ware platforms. This approach leverages decentralized, self-managed blockchain technologies to facilitate trusted interactions and information exchange among distributed neurons. The objective of the study in [11] was to investigate the effectiveness of a multi-layer perceptron artificial neural network in character recognition across three different implementation platforms: personal computers, cloud computing environments, and smart cyber-physical systems. Integrating AI techniques into IoT devices can be seen very similar to integrating AI into *Programmable Matter* (or modular robots) systems. Hence, we can observe that many of the issues and constraints are shared between IoT and modular robots: a large number of nodes (modules), limited memory storage, processing power and energy, etc. Nevertheless, there are significant differences as well, notably that the topology of modular robots will evolve continuously as the robot changes its morphology. The communications in modular robots are generally done only with the adjacent neighbor modules (i.e., communication through modules borders, as wireless communication is much more costly and prone to errors) which reduces the impact on messages loss and complicates the routing.

To the best of our knowledge, this is the first work that studies the implementation of a neural network directly on a *Programmable Matter* system. In this paper, our aim is to study the feasibility of such an implementation by proposing and implementing an ANN distributed algorithm on a real *Blinky Block* platform [16]. The idea is to train the model allowing a modular robot system to recognize the shape that it forms dynamically. Several challenges arise from this integration from computing and communication sides. For instance, the definition of the different layers, the routing of the messages in the networks, the communication failures, the memory space, etc. In our study we proposed a distributed algorithms that takes into account all these constraints.

The reminder of this paper is organized as follows. In Section 2 we present our research study and methodology. Section 3 details the simulation and the real implementation of an ANN on a *Blinky Block* platform and discusses the obtained results. Section 4 concludes the paper and gives some directions for future research work.

2 Research Study

The objective of our work is to integrate ANN into *Programmable Matter* systems. These two networks are very similar in terms of topology and characteristics. PM is composed of hundreds or thousands of modules or particles, each with some computational and storage capabilities. On the other hand, an ANN is composed of high number of neurons where each one has limited processing capability. From this similarity, we can deduce the correspondence between the two networks, where a module in a PM system can act as a neuron in an ANN, and attachment between two modules as the weighted connections among neurons.

In order to study the feasibility of our proposal, we applied our approach on a network composed of a set of 256 *Blinky Blocks* (a square of 16 by 16). Some of

the *Blinky Blocks* are lighted in white while the others are not lighted (their color is black). Using an ANN model, the set has to recognize the shape made by the lighted blocks among square, disk and triangle shapes. The result of the computation is visualized turning respectively the bottom left *Blinky Block* ($(0,0)$) to red if the shape is a square, the *Blinky Block* at $(1,0)$ to orange if the shape is a disk, and the *Blinky Block* at $(2,0)$ to yellow if the shape is a triangle.

We propose here an agent-based solution where each neuron implementation is performed by a dedicated agent. An agent is a program embedded in a robot, and we will associate a different agent to each class of neuron (Input, layer, Output...). Each module contains several agents running simultaneously which can be activated by a message.

To better understand how agents work, for example *Layer Agents* compute a summation of received numbers from *Input Agents* until they have received all the expected messages, then, they send a new message (embedding the previous treated result) to propagate the computation to *Output Agents*.

Considering that we get a pre-trained neural network shown Fig. 1 that is able to recognize three different shapes: a square, a disk and a triangle. We propose in this paper a method to map efficiently the neuronal network over a grid of robots.

To summarize, the recognition process applies the following equations to input data X_i:

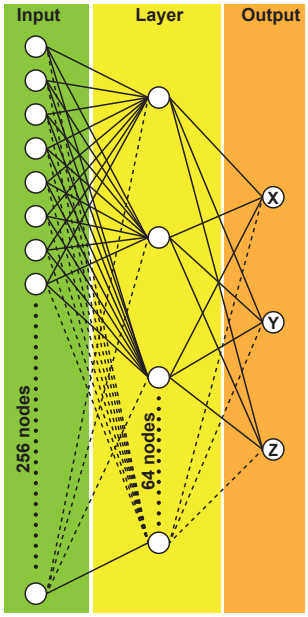


Fig. 1: Trained neuronal network used in this work.

$$A_j(X_i) = \omega_{ij} \times X_i \tag{1}$$

$$B_j = ReLU\left(\sum_{i=1}^{M_0} \omega_{ij} \times X_i + b_j\right) \tag{2}$$

$$\text{with } ReLU(a) = \begin{cases} a & \text{if } a > 0 \\ 0 & \text{if } a <= 0 \end{cases} \tag{3}$$

$$C_k = \exp\left(\sum_{i=1}^{M_1} \omega_{ik} \times B_i + b_k\right) \tag{4}$$

$$S = \sum_{k=1}^{M_2} C_k \tag{5}$$

$$D_i(S) = \frac{v_i}{S} \tag{6}$$

The distributed computation problem: To compute the sum at each node $B_j = \sum_{i=1}^{M} \omega_{ij} X_i + b_j$ in a distributed context, we create communications that transfer $\omega_{ij} X_i$ data from each module i to module j. Each receiver node must know M, in order to finish the summation (and the wait for messages) when the M^{th} message arrives. ***SoftMax computation*** To compute the SoftMax ($\frac{e^{v_i}}{\sum_i e^{v_i}}$), we have to work in two steps, first by computing the denominator, and then by re-sending the result to all the nodes in order to divide it by this sum.

We propose to define 5 different agents that can be activated in each *Blinky Block* to map these equations in a distributed system:

1. All M_0 *Input Agent* at node i send the local input X_i (multiplied by ω_{ij}) to node j over the network at start (cf. Eqn 1).
2. *Layer Agent* at node j computes the sum of values A_i received from i (M_0 nodes), and in second time computes B_j of Eqn 2.
3. *Output Agent* computes Eqn 4 with M_1 the number of nodes of the ANN layer.
4. *Super Node Agent* sum up components of the M_2 nodes of the output layer and return the result to the senders.
5. *Divider Agent* computes the final value of the output dividing the local value by a previous computed sum. At the end, this agent deduce that the shape is recognize if $Di(S)$ is higher than 80 %.

Transfer message to a destination: We need to define a distributed algorithm to transfer a message from one *Blinky Block* to another distant *Blinky Block* (placed at position *dest*). However, messages can only be passed from one module to its directly connected neighbor. It is therefore necessary to repeat several transfers from *Blinky Block* to *Blinky Block* to reach the destination. For the *Blinky Blocks* we use here (forming a 2D square), the only possible directions of message propagation are $\{North, South, East, West\}$. We will use the following algorithm, which simply tries to reduce the distance between the module with the message and *dest* at each jump.

The Algo 1 presents the method used to transfer a message to $(destx, desty)$ position. It uses a $XorY(dx, dy)$ function that returns the direction (X or Y) for the

Algorithm 1: Distributed path following algorithm

```
   Data: Agents: list of local agents.
   Data: (x,y): position of the current host.
 1 Function StartUp():
 2   | Send({agentId, data, destx, desty}) ;
 3 Function Send(msg):
 4   | if (msg.destx, msg.desty) = (module.x, module.y) then
 5   |   | agent ← {a ∈ Agents | a.id = msg.agentId} ;
 6   |   | agent.consume(msg.data) ;
 7   | else
 8   |   | dx ← msg.destx − module.x ;
 9   |   | dy ← msg.desty − module.y ;
10   |   | if XorY(dx, dy)=X then
11   |   |   | if dx > 0 then
12   |   |   |   | sendMessage(msg,PlusDxConn);
13   |   |   | else
14   |   |   |   | sendMessage(msg,MinusDxConn);
15   |   | else
16   |   |   | if dy > 0 then
17   |   |   |   | sendMessage(msg,PlusDyConn);
18   |   |   | else
19   |   |   |   | sendMessage(msg,MinusDyConn);
20 Msg Handler PathMsg(msg, sender):
21   | send(msg);
22 Function XorY(dx, dy):
23   | if dx = 0 then
24   |   | return Y;
25   | if dy = 0 then
26   |   | return X;
27   | return (rand()%(|dx| + |dy|) < |dx|)?X : Y;
```

next hop according to the complete (dx, dy) displacement.
A naive solution consists in choosing the direction with the highest absolute value (X direction if $|dx| > |dy|$ and Y otherwise), but it produces lot of traffic along the same path. For example, two paths $(10,1)$ and $(10,2)$ from a same node use exactly the same intermediary modules. In order to reduce this traffic, we add some random parts in the path: we randomly draw a number between 1 and $(|dx| + |dy|)$ and if this number is less or equal to dx we choose the X direction otherwise the Y direction.

Complexity: The huge difference between message transmission time and computing time in a module means that the complexity in terms of time corresponds to the complexity in terms of number of messages.

The computation of the Neuronal network to detect the shape drawn by the module is a complex process in several stages. Each stage detailed Fig. 2a implies to send message from all agents of a layer to all agents of the next layer. This process generates lot of parallel transfer of messages along paths separating host blocks. In studying the longest path in each stage, we can express the complexity of our algorithm.

For our case, the longest possible path is the diameter $D = 31$. Then, according to the Neuronal Network graph, we can major the number of messages by:

$$N_{msg} < 256 \times D + 64 \times D + 3 \times D + 3 \tag{7}$$

To conclude, the complexity in time is $O(D)$.

Memory distribution: The space complexity in such integration of an artificial neural network and a *Programmable Matter* system is supposed to be low. Indeed, in this combination there's no need to generate a global or network-wide weight matrix within the *Programmable Matter* system. We suppose that each module (neuron) in the system maintains its weight vector locally. Communications between attached neighboring nodes (transmitting and receiving neurons) eliminate the need of storing this big size network matrix. At the end, each module stores only 2 vectors. The first one is the local weight vector and the second one is to store output values of particles connected to a it or another specific particle. Therefore, in the worst-case scenario where we consider a fully connected networks of N, where N is the total number of modules in the system, then the space or memory complexity is $O(2N)$ of float or double numbers.

However, in our case study, the total memory of the one layer neuronal network is given by the following relation:

$$M = \left((N_{input} + N_{output} + 1) \times N_{layer} + N_{output}\right) \times size \tag{8}$$

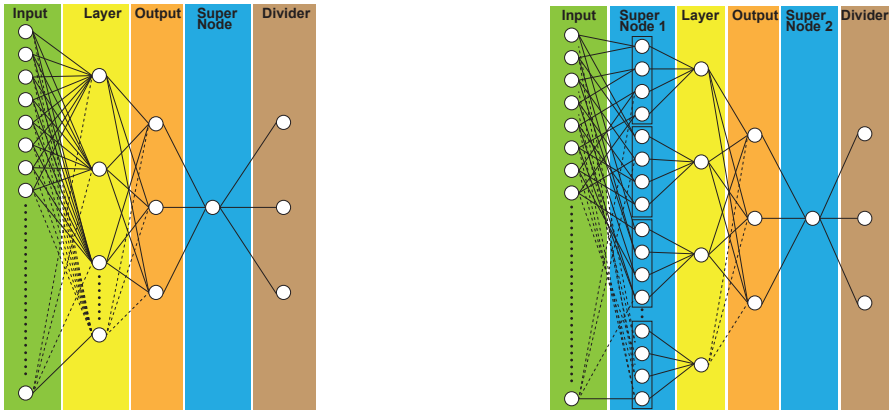
Where N_{input}, N_{layer} and N_{output} are respectively the number of Input, Layer and Output nodes and *size* is the size of a single number (8 B for a 'double'). In our context $N_{input} = 256$, $N_{layer} = 64$ and $N_{output} = 3$, it represents 130 kB which is too important for the memory capacity of the *Blinky Block*.
The neuronal network memory is naturally distributed over the modules. If we consider an isolated *Blinky Block*, it only needs the memory used in the embedded agents. Each *Input Agent* needs 64 weights only, each *Layer Agent* stores one bias and 3 weights, and an *Output Agent* store a bias only. Three modules embed this 3 agents, then in maximum, the memory used by a *Blinky Block* to store the neuronal network data is $(64 + 4 + 1) \times 8 = 552$ bytes only.

The important point is that the complexity in memory of our solution is $O(1)$.

3 Experiments

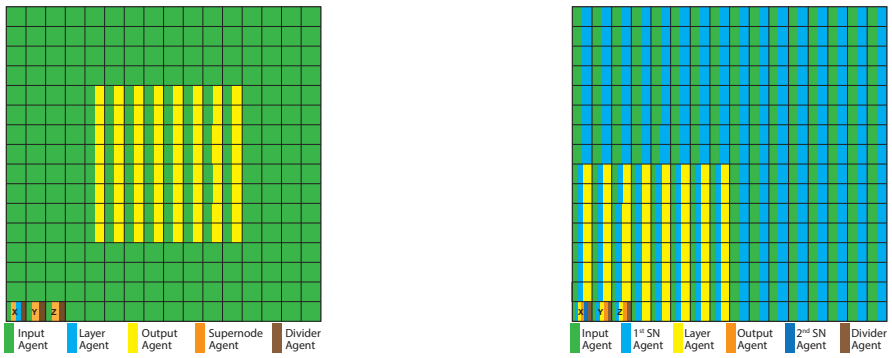
The evaluation of the algorithm was made in simulating a set of 256 *Blinky Blocks* with *VisibleSim* (a behavioral simulator [17]) using communication times proposed in [16] and in implementing the algorithm on real *Blinky Block* hardware as well.

Comparing two methods on simulator: We first propose a method with direct mapping of the neuronal network agents over the set of *Blinky Blocks*. This neuronal network presented Fig. 1 is made of an input layer with 256 nodes, an intermediary layer with 64 nodes and an output layer with 3 nodes.



(a) Initial basic graph. (b) Optimized graph.

Fig. 2: Designs of agents for the two versions in *VisibleSim*.



(a) Reference configuration. (b) First optimization.

Fig. 3: Design of agents over the set.

For our first distributed implementation with agents shown Fig. 2(a), we place an *Input Agent* in each 256 *Blinky Blocks* of the set and a *Layer Agent* in the 64 *Blinky Blocks* placed at the center of the square as shown Fig. 3(a). We add the three *Output Agent* at the bottom left corner of the square set, in cells marked ($\{X,Y,Z\}$) in Fig. 3(a). To compute Equ. 6 we add two layers, the first one is only made of a *Super Node Agent* placed in the *X Blinky Block* only, and the second layer is made of three *Divider Agent* placed in ($\{X,Y,Z\}$) *Blinky Blocks*.

To apply the neuronal network and recognize the shape each *Input Agent* sends a message with the data $\omega_i \times v_i$ to all *Blinky Blocks* with *Layer Agent* (where v_i is the sender value, according to its color and ω_i the pre-computed weight of the link). As a consequence, each *Layer Agent* has to wait for all 256 messages before computing

Eq.(2) and send the result to the three modules ($\{X,Y,Z\}$) running the *Output Agent* at the top left of the set.

The advantages of this method are that it implements an exact mapping of the neuronal network (cf. Fig. 1) on the set of *Blinky Blocks* that make easy the determination the number of messages waited by each agent. But the simulation shows that it produces a huge number of messages: 144,005 representing a total computing time of 1.051 s.

The second version is an optimization that reduces the average distance between *Blinky Blocks* that contain connected agents Fig. 2(b). We try to distribute the 64 nodes of the first layer of the Neuronal Network graph in all the 256 *Blinky Blocks*. As a consequence all nodes of the first layer are duplicated 4 times in the first layer of *Super Node Agent* presented Fig. 3(b). That implies that we add a new treatment to sum the 4 components into a *Layer Agent* placed in the bottom left corner, to be closer as possible to the final module with *Output Agent*.

This method reduces the average distance between a module and modules that embed the agents for layer 1. The drawback is the addition of a virtual layer that sum the contributions of the 4 agents treating the same node.

The new number of messages with this optimization is 79,046, representing a total simulation time of 0.401 s for a gain of 62 %. All the simulation results are presented in Table 1.

We apply the previous algorithm on three different shapes to check that they detect the good shape. We can notice on Fig. 4 that *Blinky Block* $X(0,0)$ is red for the square shape, *Blinky Block* $Y(1,0)$ is orange for the disk shape and finally *Blinky Block* $Z(2,0)$ is yellow for the triangular shape. Finally, they are all grey for the "question mark" shape, which is not detected as one of the three processed shapes.

Implementation on real Blinky Blocks required a few simplifications: First of all, the grid size is 8×8, and we are using single-precision 4-byte reals (float) to reduce the data size. We then carried out a new training session to define a neural network that recognizes all three shapes at this new scale. The neural network we implemented accepts the color of 64 Input nodes, then 64 Layer nodes and finally 3 Output nodes. The model presented in Fig. 3a is the one implemented in the agents.

Table 1: Simulation results.

Method	nb of messages	time for square (s)	time for disk (s)	time for triangle (s)
Centered Layer	144007	1.10534	1.10534	1.103899
with rand path		1.071082	1.071082	1.089041
Distributed Layer	79046	0.413323	0.413323	0.413313
with rand path		0.374945	0.374945	0.370397
Shape	"?"	Square	Disk	Triangle
X rate	6.04 %	98.29 %	2.12 %	0.39 %
Y rate	26.61 %	1.58 %	97.87 %	0.23 %
Z rate	67.35 %	0.12 %	0.002 %	99.37 %

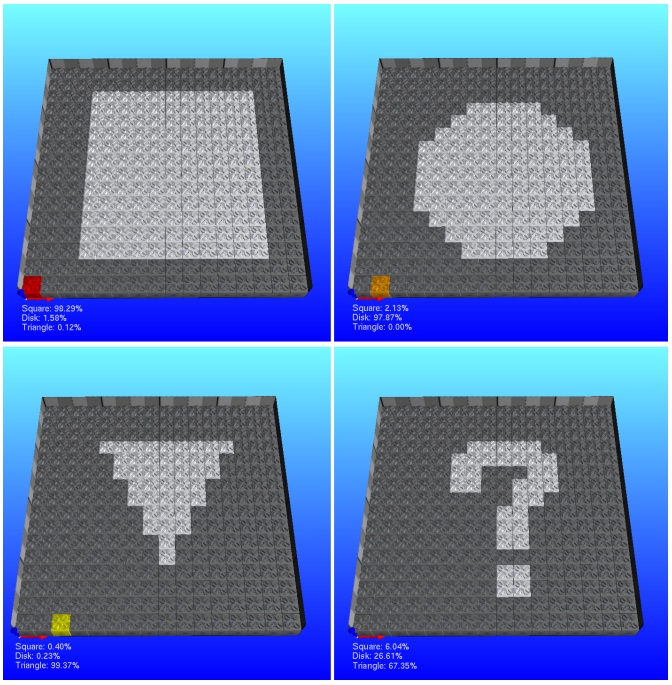


Fig. 4: Simulation of the distributed algorithm on *VisibleSim* for four different configurations. Red block is for square detection, orange for disk and yellow for triangle shape.

Each *Blinky Block* must run the same program, that follows an algorithm in 3 steps: At start, the leader placed at $X = (0,0)$ build a coordinate system relative to its position and orientation. This method takes into account the random orientation of the *Blinky Blocks* in the set using the distributed method proposed in [18]. Then each module creates a routing table, enabling each *Blinky Block* to know which connector provides access to the module with coordinate (x_g, y_g). To do this, each robot broadcasts its position in a message to all the others. When a module receives such a message, it memorizes the direction of arrival as the direction to follow to send it a message. The video[1] shows the complete detection of each shape by the 64 *Blinky Blocks*.

The third step is the running of the computation of the recognition process from the initial values of the *Blinky Blocks*. We can observe on the video and on Figure 5 that the three shapes described in the color of the *Blinky Blocks* are well detected by the distributed system.

[1] Youtube video: https://youtu.be/q13-Kz4_clc

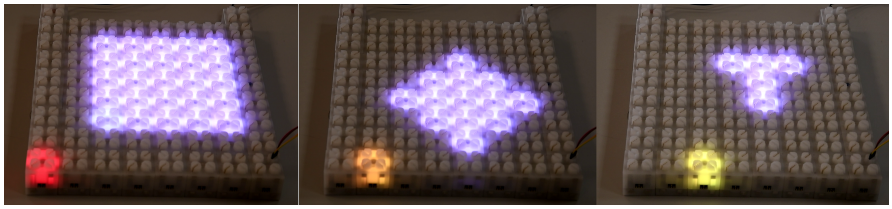


Fig. 5: Three pictures from the video capture of our experiments.

4 Conclusion

In this paper we studied the integrating of an artificial neural network into a *Programmable Matter* based modular robot system. The aim of this study is to improve the capabilities of adaptation and self-reconfiguration of such systems. To do this, we considered that each module in a PM system replaces one neuron in an ANN, and attachment between two modules are the weighted connections among neurons. A case study was conducted and based on implementing an ANN to let a modular robot system recognize its shape automatically and in a distributed manner. The proposed algorithm takes into consideration the network constraints, low computational and memory capabilities and communications between attached modules. The complexity of this algorithm is studied showing $O(1)$ for the space complexity and $O(D)$ for the time complexity. The efficiency of the algorithm was tested via simulations and real implementation on the *Blinky Block* platform. We compared the results to those obtained by a model implemented on a PC, and the results show clearly the feasibility of our proposed approach.

For future work, we intend to test and implement other types of neural networks directly on *Programmable Matter* systems. For instance, ANN can be used for clustering, or Hopfield networks like mean-field annealing or Boltzmann machines can be used for optimization. Furthermore, a main study will be conducted to distribute and test the training phase as well directly on the *Blinky Block* platform.

ACKNOWLEDGMENT

This work has been supported by the EIPHI Graduate School (contract "ANR-17-EURE-0002").

References

1. P. Thalamy, B. Piranda, and J. Bourgeois, "A survey of autonomous self-reconfiguration methods for robot-based programmable matter," *Robotics and Autonomous Systems*, vol. 120,

p. 103242, 2019. [Online]. Available: https://doi.org/10.1016/j.robot.2019.07.012
2. J. Bassil, A. Makhoul, B. Piranda, and J. Bourgeois, "Distributed size-constrained clustering algorithm for modular robot-based programmable matter," *ACM Trans. Auton. Adapt. Syst.*, vol. 18, no. 1, pp. 1:1–1:21, 2023.
3. N. Sharma, R. Sharma, and N. Jindal, "Machine learning and deep learning applications-a vision," *Global Transitions Proceedings*, vol. 2, no. 1, pp. 24–28, 2021.
4. L. Herrmann and S. Kollmannsberger, "Deep learning in computational mechanics: a review," *Computational Mechanics*, pp. 1–51, 2024.
5. C. Abou Akar, R. Abdel Massih, A. Yaghi, J. Khalil, M. Kamradt, and A. Makhoul, "Generative adversarial network applications in industry 4.0: A review," 2024. [Online]. Available: https://doi.org/10.1007/s11263-023-01966-9
6. S. Sharma and P. Chaudhary, "Machine learning and deep learning," *Quantum Comput. Artif. Intell. Train. Mach. Deep Learn. Algorithms Quantum Comput*, pp. 71–84, 2023.
7. A. Hennebelle, L. Ismail, H. Materwala, J. Al Kaabi, P. Ranjan, and R. Janardhanan, "Secure and privacy-preserving automated machine learning operations into end-to-end integrated iot-edge-artificial intelligence-blockchain monitoring system for diabetes mellitus prediction," *Computational and Structural Biotechnology Journal*, vol. 23, pp. 212–233, 2024.
8. M. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–37, 2021.
9. J. Azar, A. Makhoul, M. Barhamgi, and R. Couturier, "An energy efficient iot data compression approach for edge machine learning," *Future Generation Computer Systems*, vol. 96, pp. 168–175, 2019.
10. A. Hazarika, N. Choudhury, M. M. Nasralla, S. B. A. Khattak, and I. U. Rehman, "Edge ml technique for smart traffic management in intelligent transportation systems," *IEEE Access*, 2024.
11. M. A. Torres-Hernández, M. H. Escobedo-Barajas, H. A. Guerrero-Osuna, T. Ibarra-Pérez, L. O. Solís-Sánchez, and M. d. R. Martínez-Blanco, "Performance analysis of embedded multilayer perceptron artificial neural networks on smart cyber-physical systems for iot environments," *Sensors*, vol. 23, no. 15, p. 6935, 2023.
12. C.-W. Peng, C.-C. Hsu, and W.-Y. Wang, "Mobile mapping system for automatic extraction of geodetic coordinates for traffic signs based on enhanced point cloud reconstruction," *IEEE Access*, vol. 10, pp. 117 374–117 384, 2022.
13. S. M. Alrubei, E. Ball, and J. M. Rigelsford, "The use of blockchain to support distributed ai implementation in iot systems," *IEEE Internet of Things Journal*, vol. 9, no. 16, pp. 14 790–14 802, 2021.
14. M. Holenderski, J. Lukkien, and T. C. Khong, "Trade-offs in the distribution of neural networks in a wireless sensor network," in *2005 International Conference on Intelligent Sensors, Sensor Networks and Information Processing*. IEEE, 2005, pp. 259–264.
15. D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," *Nature communications*, vol. 9, no. 1, p. 2383, 2018.
16. J.-P. Yaacoub, B. Piranda, F. Lassabe, and H. Noura, "Efficient Communication Protocol for Programmable Matter," in *38th IEEE International Conference on Advanced Information Networking and Applications (AINA 2024)*, 2016. [Online]. Available: https://www.researchgate.net/publication/286441618
17. P. Thalamy, B. Piranda, A. Naz, and J. Bourgeois, "Behavioral simulations of lattice modular robots with visiblesim," in *15th International Symposium on Distributed Autonomous Robotic Systems (DARS 2021)*, Kyoto, Japan, jun 2021. [Online]. Available: https://publiweb.femto-st.fr/tntnet/entries/17403/documents/author/data
18. B. Piranda, F. Lassabe, and J. Bourgeois, "Disco: A multiagent 3d coordinate system for lattice based modular self-reconfigurable robots," in *IEEE International Conference on Robotics and Automation (ICRA 2023)*, London, England, may 2023.