applied sciences

MDPI

*Article*

# FLaMAS: Federated Learning Based on a SPADE MAS

**Jaime Rincon †, Vicente Julian *,† and Carlos Carrascosa †**

Valencian Research Institute for Artificial Intelligence, Universitat Politècnica de València, 46022 Valencia, Spain; jrincon@dsic.upv.es (J.R.); carrasco@dsic.upv.es (C.C.)

* Corresponding: vjulian@upv.es

† These authors contributed equally to this work.

**Abstract:** In recent years federated learning has emerged as a new paradigm for training machine learning models oriented to distributed systems. The main idea is that each node of a distributed system independently trains a model and shares only model parameters, such as weights, and does not share the training data set, which favors aspects such as security and privacy. Subsequently, and in a centralized way, a collective model is built that gathers all the information provided by all of the participating nodes. Several federated learning framework proposals have been developed that seek to optimize any aspect of the learning process. However, a lack of flexibility and dynamism is evident in many cases. In this regard, this study aims to provide flexibility and dynamism to the federated learning process. The methodology used consists of designing a multi-agent system that can form a federated learning framework where the agents act as nodes that can be easily added to the system dynamically. The proposal has been evaluated with different experiments on the SPADE platform; the results obtained demonstrate the benefits of the federated system while facilitating flexibility and scalability.

**Keywords:** artificial intelligence; federated learning; multi-agent systems; agent platforms; edge computing



**Citation:** Rincon, J.; Julian, V.; Carrascosa, C. FLaMAS: Federated Learning Based on a SPADE MAS. *Appl. Sci.* **2022**, *12*, 3701. https://doi.org/10.3390/app12073701

Academic Editor: Juan Pavón

Received: 11 March 2022
Accepted: 4 April 2022
Published: 6 April 2022



## 1. Introduction

By 2023, there will be 29.3 billion electronic devices connected to the Internet in the world, and half of them will be Internet of Things (IoT) objects, according to the latest report by Cisco [1]. In recent years, IoT solutions have become popular for remotely tracking, monitoring, and maintaining industrial manufacturing devices that are part of the production value chain by assessing equipment conditions and remotely diagnosing equipment failures before they happen, by analyzing the real-time data stream produced by the machine. In a related way, remote health monitoring solutions that use IoT technology to improve quality of life and care through precise and focused home medical monitoring have also become widespread.

A typical IoT architecture is essentially made up of a series of different devices interconnected with each other and with higher-level nodes. Often, the lower-level devices, mainly in charge of data acquisition, are limited in computational resources. In such cases, the information obtained is typically transferred through the network for processing and analysis at centralized nodes with greater computational power. This architecture, therefore, makes use of the cloud as a data repository and computational cluster [2].

There are numerous drawbacks to this type of architecture, such as a clear increase in the amount of information transferred, an increase in the total processing time, and an obvious problem of unauthorized access to data. This last problem has led to the appearance of different regulations, both at the European level [3], and in other countries [4,5].

In this context, the federated learning paradigm [6], proposed by Google, allows obtaining models based on machine learning through multiple data sources without the need to exchange all of the data. In the IoT domain, this paradigm assumes that each IoT node can train an individual model using its own data and only shares the model

parameters. Usually, a central node will adequately aggregate the different parameters to obtain a single global model.

Many proposals based on the federated learning paradigm have appeared in recent years. In the literature, there have been several surveys that discuss possible open challenges in the area, such as those presented in [7–9]. Among the different problems that are still open, current proposals often raise issues of flexibility and dynamism in large-scale IoT systems. Recognizing this, the main objective of this study was to demonstrate how multi-agent system (MAS) technology can be exploited to improve necessary data exchange and the aggregation process in a federated learning system [10] by improving its flexibility and dynamism.

To achieve the proposed objective, we present the development of a multi-agent approach that gives support to a federated learning framework. This framework allows agents to perform training tasks locally and transfer only the trained model to a server agent in charge of generating the global model and sharing it with the rest of the agents. In addition, the agents in charge of training can enter or leave the system dynamically, which allows for a completely open system. Thus, the contributions of this investigation are the following: (i) design of a framework for federated learning based on multi-agent systems; (ii) development of the proposed framework on the SPADE [11] (Smart Python Agent Development Environment) multi-agent systems platform; and (iii) development of the necessary experiments to demonstrate the feasibility of the proposed development.

The results obtained confirm how the proposed framework makes it possible to have an open multi-agent system where agents in charge of training can enter or leave the system without affecting the overall performance of the federated learning process. This results in greater flexibility and dynamism of the system. Moreover, the experiments undertaken showed that the proposed framework achieved higher accuracy in a shorter time and with a smaller number of epochs than a centralized solution.

The rest of the paper is structured as follows. Section 2 discusses several related previous studies; Section 3 describes in detail the federated learning framework based on the SPADE platform; Section 4 presents several experiments and the results obtained; and, finally, Section 5 discusses some conclusions and future research.

## 2. Related Work

There has been a huge increase in the number of mobile devices that can be used as providers of different kinds of data (e.g., sensory data, such as the global positioning system (GPS), accelerometers, any kind of sensor data, and even images).This data can be provided to deep learning processes to be used in intelligent applications.

However, this idea has two very important problems associated with it—one is related to data privacy (as the mobile device has to communicate all the data to the server), while the other is related to bandwidth limitations. So, it is more appropriate only to have such data available on the device.

That is the idea behind federated learning, as defined by Google in [6]; that is, a learning algorithm that benefits by sharing models of trained data with no central model. Thus, the learning is produced by a federation of *clients* that are coordinated by a *server*.

In traditional machine learning, data is obtained from different sources or devices and is sent to the server, where it is used to train the machine learning model. So, this is a *centralized learning* model. In *federated learning*, the training is made in a distributed and edge fashion. So, the model training is carried out in the same device where the input data is obtained. This enables the use of sensible data for training, as data privacy is preserved by not sharing the data, only the trained model.

Federated learning has not been the only approach to distributed learning preserving data privacy proposed in recent years. Another interesting approach is, for instance, *MIT split learning* (https://splitlearning.github.io (accessed on 28 February 2022)) [12], where, in its simplest configuration, a deep learning network is split between two entities; the first computes the data until the cut layer, and the output of this layer (*smashed data)* is sent to the

other entity for computation to be continued through the network. This can be performed for both training and execution. In this way, the privacy of the input data is preserved. One of the main open issues of this technique is parallelization over all IoT devices.

In recent years, many studies have been undertaken to progress federated learning. In this respect, we have identified several review papers that have analyzed progress made in this area, such as of [7,8], which provide an overview of existing developments and approaches and outline a number of future research directions. These papers summarize some applications in federated environments and discuss areas of development with considerable potential for the application of federated learning. In [13,14], the authors classify federated learning research in terms of possible new designs, applications, and challenges. These reviews discuss the unique properties and challenges of federated learning compared with traditional distributed data center computing. In addition, they examine several open problems worthy of future research effort.

One of the areas most closely related to federated learning is the IoT area. This is because, in most of the proposals, the learning occurs at the edge and, more specifically, in devices that directly perform the sensing of the data.

The authors of [15] provide a review and comparative analysis of different existing open source federated learning frameworks. In addition, the paper considers their applicability in IoT systems, taking into account aspects such as analysis capabilities, accuracy, deployment, development, and performance. Another interesting review is provided in [16], where the authors analyze recent advances in federated learning that can enable IoT applications. The paper introduces a number of metrics, such as sparsification, robustness, quantization, scalability, security and privacy, to compare the analyzed proposals. A further review is presented in [17], which discusses the opportunities and challenges of federated learning in IoT platforms, as well as how federated learning can enable different IoT applications. The paper also identifies critical challenges of federated learning in IoT platforms, highlighting some recent promising approaches to address them.

Regarding specific studies, we highlight the framework presented in [18], where the authors propose the use of blockchain and federated learning approaches to build a secure architecture for privacy-preservation, oriented to smart healthcare. In this case, the idea is to enhance security and privacy by employing the two technologies in a combined manner. In a related way, the research presented in [19] makes use of blockchain and federated learning by designing a distributed architecture for data sharing between multiple parties oriented to the industrial IoT. In [20], the authors present *FedHealth*, a federated transfer learning framework for wearable healthcare, which performs data aggregation using federated learning, and then builds personalized models through transfer learning. In this case, the application area is again healthcare and the main challenges are privacy and security. Finally, another interesting study is presented in [21], where a new algorithm is proposed to implement consensus techniques for federated learning in large scale networks, which was validated on an industrial IoT scenario.

Large mobile companies are not only researching federated learning, but are also using this algorithm in development; not only is *Google* using it in the *Gboard* mobile keyboard [22], in *Android* messages [13], and in some features in the *Pixel* [23] mobile phone, but *Apple* also is using it in its mobile keyboard *Quicktype*, and even in the voice classifier for "Hey, Siri", all in its iOS 13 [24].

As can be seen, there are many recent studies that have focused on proposing frameworks or mechanisms to facilitate the use of federated learning techniques. Many of these proposals have focused on the IoT area given the close relationship between the two technologies. In most cases, the idea is to improve crucial aspects, such as security and privacy of the information obtained by IoT devices. However, most federation approaches are still rather rigid and may not perform well in situations involving frequent node changes, connectivity problems, etc. Thus, the following section presents a new flexible tool for the development of federated learning systems based on the use of the SPADE platform, which is specially designed for the creation of multi-agent systems.

## 3. Multi-Agent Learning Based on Federated Learning

This section presents a real implementation of a federated learning algorithm in a multi-agent system based on SPADE agents. Firstly, we include a brief description of SPADE, followed by a description of the federated learning algorithm, as has been implemented in SPADE agents, that we have called *FLaMAS* (Federated Learning based on MAS).

### 3.1. SPADE

*SPADE* (http://spade-mas.readthedocs.io/ (accessed on 28 February 2022)) [11] (Smart Python Agent Development Environment) is a framework for the development of multi-agent systems based on Python and having as a main feature its usage of the XMPP (extensible messaging and presence protocol) instant messaging protocol (https://xmpp.org (accessed on 28 February 2022)).

The programming model for SPADE agents is based on behaviors, including not only classical bahaviors, such as one-shot, periodic or finite-automata, but also BDI (belief desire intention) behavior [25], allowing the mixing of procedural, object-oriented and logic programming in the same agent.

### 3.2. FLaMAS

*FLaMAS* is a new federated learning tool for multi-agent systems based on SPADE. *FLaMAS* brings together the benefits of deep learning and multi-agent systems to create a new distributed learning tool. The system allows the training of deep learning models from anywhere in the world, following the main features of a federated learning algorithm, being distributed (enhanced by the implementation in SPADE agents), and ensuring privacy of the data (as only the model being learned is shared and not the data used for the training). These features are enhanced by the inclusion of the learning algorithms in a MAS, such as that formed by SPADE agents, i.e., a distributed framework designed for distributed applications, with easy and rapid adaptation to failures in nodes, or new nodes entering the learning process during its execution.

In the *FLaMAS* system, there exist two different agent roles:

- Client Role: its main goal is to receive the input data from the device where it is embedded, train the model and send it to the agent playing the Server role.
- Server Role: Its main function is to be in charge of receiving at each iteration the different trained models of the agents playing the Client role in the system. The agent playing the Server role takes the average of all those models and this new unified model is sent to the different agents playing the Client role.

### 3.3. Agent Interaction Model

This section details the different agent interaction stages of the presented system. These stages are the normal execution cycle stage and the new agent entering the system stage.

#### 3.3.1. Normal Execution Cycle

The normal execution cycle stage of an agent in *FLaMAS* is made up of four states, as shown in Figure 1 (independently of the role it is playing):

1. SETUP STATE: In this state the agent configures parameters, such as “agent name”, “XMPP Server IP”, “name of the database to train”, and any other configuration parameters required by the developer.
2. TRAIN STATE: In this state, if the agent is playing the Client role, the training of the deep learning model is performed, and the weights and the losses parameters of the model are extracted. If the agent is playing the Server role, it calculates the average of the weights of the different models received from the Client agents. These weights are then used by the next state.
3. SEND STATE: This state is responsible for encapsulating the different weights (weights and losses), using the XMPP message structure, and sending them. If the agent is

playing a Client role, they will be sent to the Server agent, otherwise, the Server agent will send them to the agents playing the Client role.

4. RECEIVE STATE: If the agent is playing the Client role, this state receives the message coming from the Server agent, which has the new weights. This same state is in charge of introducing these new weights into the model that is being trained. On the other hand, if the agent is playing the Server role, in this state it has to wait for all the agents playing the Client role, until the Presence state indicates they are active.

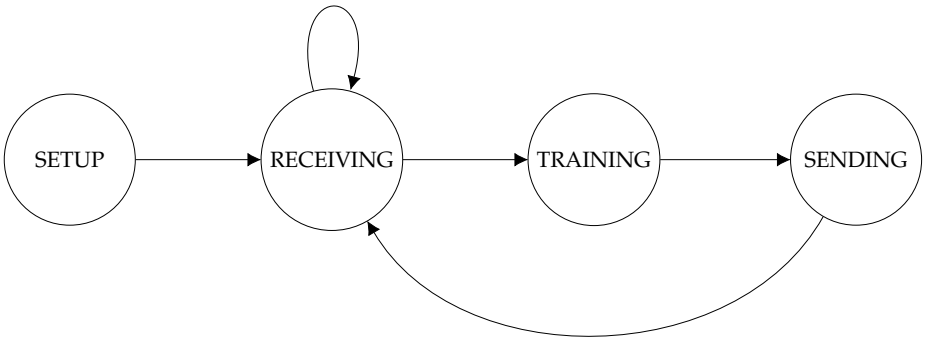


**Figure 1.** Finite state machine for the agents in the *FLaMAS* system.

At each iteration of the normal execution cycle of the *FLaMAS* system, there are the following steps and messages (the last ones are shown in Figure 2):

1. The Server agent uses the presence of all the Client agents to wait for the model trained by all the active Client agents.
2. Each active Client agent trains its model.
3. Each active Client agent sends its trained model to the Server agent.
4. After receiving all the trained models of the active Client agents, the Server agent calculates the average of all the models.
5. The Server agent sends the average model to each active Client agent.
6. Each active Client agent deploys the average model received.

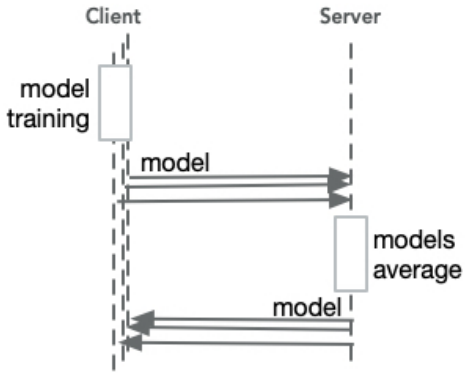


**Figure 2.** Agent interaction protocol for an iteration in the normal execution cycle.

#### 3.3.2. New Agent Entering the System

When a new agent enters the system, it will be playing the Client role, and will be linked to an IoT device, acquiring all the information related to the IoT device sensors.

In this way, the new agent will communicate with the agent playing the Server role, to communicate that it is going to play a new Client. Figure 3 shows the protocol or sequence of messages between the two agents in this stage:

1. The new agent sends a message to the agent playing the Server role indicating that it wants to *join* the system as another Client.
2. The agent playing the Server role will subscribe to the Presence of the new Client agent.

3. When the Server agent receive the confirmation of the new agent Presence subscription, it will add it to its clients list for the normal execution cycle.
4. The Server agent will send the *init* message with the parameters needed to initialize the model in the Client agent.
   As the Server agent receives this message, it will begin the other protocol commented above (normal execution cycle) in both agents.

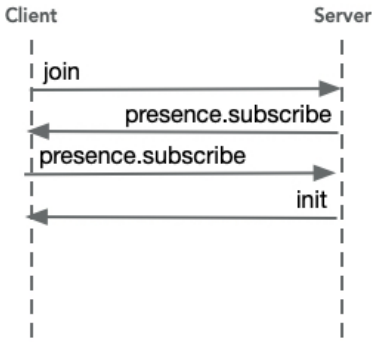


**Figure 3.** Agent interaction protocol for a new agent entering the system.

In this section, we have presented *FLaMAS* as a federated learning algorithm developed using SPADE agents as a way to have all the advantages of both developments: an enhanced learning process that uses traditional isolated learning, but which allows for preservation of the privacy of the local data used for each agent to train their local model (federated learning main advantages), along with a scalable distributed solution that adapts quickly to failures in agents (the Server only waits for currently available Client agents through their Presence feature), or even to new agents entering the process when it is still running.

Figure 4 shows a brief schema summarizing how *FLaMAS* works: After all agents (Clients and Server) are created and registered in a XMPP Server, the system enters in a cycle where the following steps are executed:

1. All Client agents that are active in the system (so their Presence would indicate it), will get input data from the environment.
2. All active Client agents will train their local models.
3. If the Server agent is active (indicated by its Presence), all active Client agents will send their trained local model parameters to the Server agent.
4. The Server agent calculates a de-aggregated global model.
5. The new global model is sent by the Server agent to the active Client agents, which put it in their systems as the new local model.

To conclude this summary, we would like to underline the use of the Presence in the SPADE agents, which not only ensures that the system will adapt to agents failing in the system, and recovers from fails after a while, but also allows new Client agents to enter in the system during its execution.

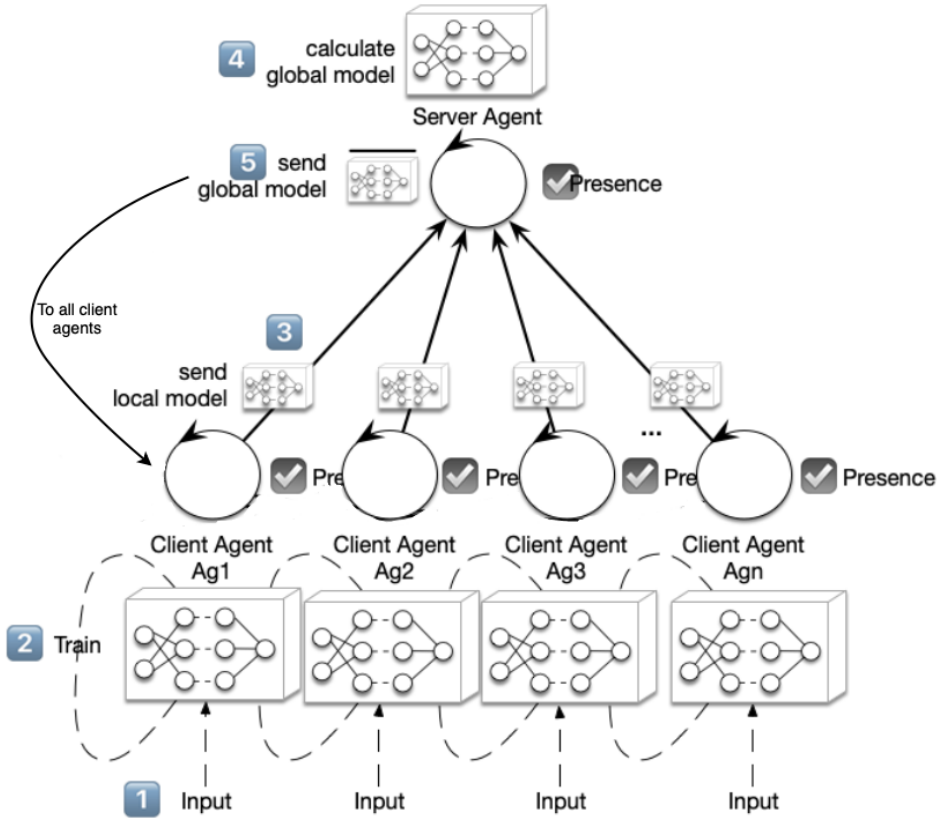


**Figure 4.** *FLaMAS* framework description.

## 4. Experiments

In this section, we describe the different experiments performed to determine the advantages of the federated learning tool *FlaMAS* using multi-agent systems. To determine whether federated learning is of more interest than normal learning, a series of experiments were performed using the *MNIST* (Modified National Institute of Standards and Technology) database (Figure 5). *MNIST* is a database of handwritten digits, and is one of the most common and widely used in machine learning. It has a training set of 60,000 examples and a test set of 10,000 examples. It is a subset of a larger set available from NIST (https://www.nist.gov/ (accessed on 28 February 2022)) (National Institute of Standards and Technology). The digits have been normalized in size and centered on a fixed-size image.

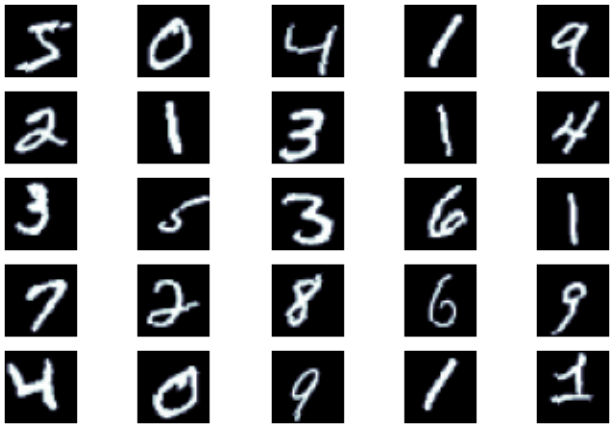

**Figure 5.** Examples of *MNIST* database.

To evaluate the advantages of this federated learning approach compared to traditional centralized learning, a number of parameters were defined to determine which of the two methods is more suitable. The parameters selected were accuracy, training time, and the number of epochs needed to obtain an optimal classification model. For the different experiments a convolutional neural network (CNN) network was trained.

The network used in the experiments was configured using the following hyper-parameters:

- Input Size: 784 ($28 \times 28 = 784$).
- Hidden Size: 600
- Batch size: 128
- N-Classes: 10
- Model Optimizer: SGD
- Learning Rate: $1 \times 10^{-3}$

Figure 6 shows the structure of the compiled model. It is composed of four layers, so two hidden layers are included.


**Figure 6.** Structure of the network model (left module is the input layer, right module is the output layer, and the other two layers are hidden).

The definition of the internal structure of the model is shown below.

- First layer (input layer) is a Convolutional 2d, with the following configuration:
  - Input channels = 1 (binary image)
  - Output channels = 10
  - Kernel size = 5
- Second layer (hidden layer) is a Convolutional 2d, with the following configuration:
  - Input channels = 10
  - Output channels = 20
  - Kernel size = 5
- Dropout2d (parameter for reducing the dimensionality of the data at this level) = 0.5
- Third layer (hidden layer) is the first linear layer, with the following configuration:
  - Input features = 320
  - Output features = 50
- Fourth layer (output layer) is the second linear layer, with the following configuration:
  - Input features = 50
  - Output features = 10

The first experiment consisted of training the CNN in a centralized manner. In the second experiment, the same network architecture was trained, but in a distributed manner, applying the *FLaMAS* approach. Being the same type of network, some parameters were the same in both experiments, but in the distributed learning network only, we added a parameter denoted N-local epochs. To compare the performance of our system in both experiments, we decided to train both experiments with the number of epochs necessary to obtain an accuracy of better than 90% in both situations. The third experiment examined the effect of increasing the number of epochs for local training between two communications with the Server agent. Lastly, the fourth experiment assessed how the system would adapt to new agents entering the system during its execution.

### *4.1. Experiment 1: Centralized Learning*

As mentioned above, the first experiment was a typical centralized training using a CNN.

In this experiment, 20 training epochs were needed to reach the goal of an accuracy of over 90%. The time needed to obtain this classification rate was approximately 5 min. Once the system had been trained, the best models obtained were stored for evaluation. The evaluation of the model was performed on 10% of the database. The result of this validation process can be seen in the confusion matrix of Figure 7.

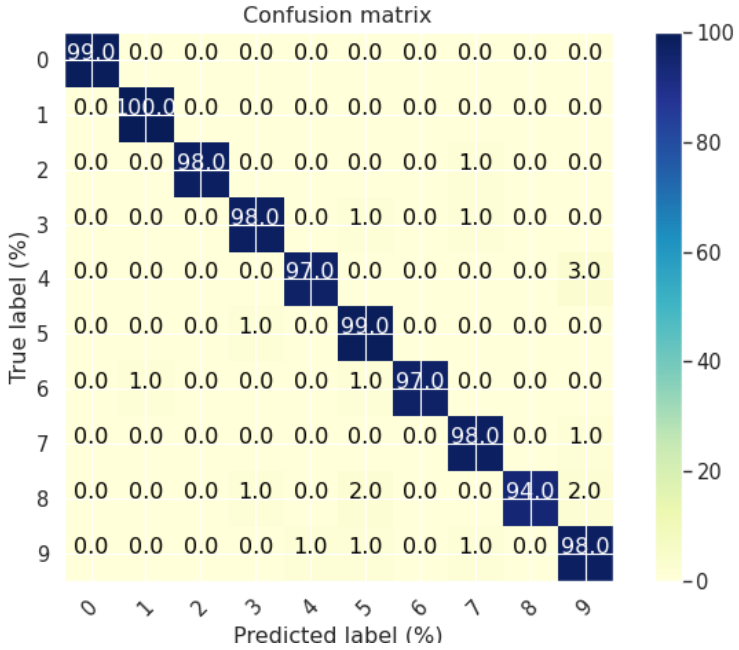


**Figure 7.** Confusion matrix for Experiment 1: centralized training.

### *4.2. Experiment 2: Distributed Learning (FlaMas)*

In the second experiment using our *FlaMas* tool, we sought to decrease the training time, as well as the number of epochs needed to obtain a model with an accuracy higher than 90%. To find out if it was possible to decrease these two variables, we decided to distribute the same dataset among three SPADE agents using the *FLaMAS* development playing the Client role. Each of them had access to 100% of the MNIST database. A fourth SPADE agent behaved as a Server agent, to receive the messages from the three agents. This agent had the task of calculating the average of the weights, which are network parameters that transform the input data in the hidden layers of the network. These parameters were sent by each of the Client agents at the end of a training epoch.

As commented above, the same network used in the first experiment was used in the second; thus, the same hyper-parameters were applied to this experiment, with one exception—the number of epochs. Instead of this parameter, we added two new hyper-parameters. These new hyper-parameters were the number of local epochs (N-local epochs), and the number of global epochs (N-global epochs). N-local epochs corresponded to the number of training epochs each agent had and N-global epochs corresponded to the number of global iterations the central agent performed before stopping training. The parameter N-global epochs corresponded to the epochs of centralized learning.

Once the system had been trained, the best models obtained were stored for evaluation. The evaluation of the model was performed on 10% of the database; the result of this validation process can be seen in the confusion matrix of Figure 8.

For this experiment, we defined *N-Local Epochs* as 1, and we reached the goal after 5 *N-Global Epochs*.

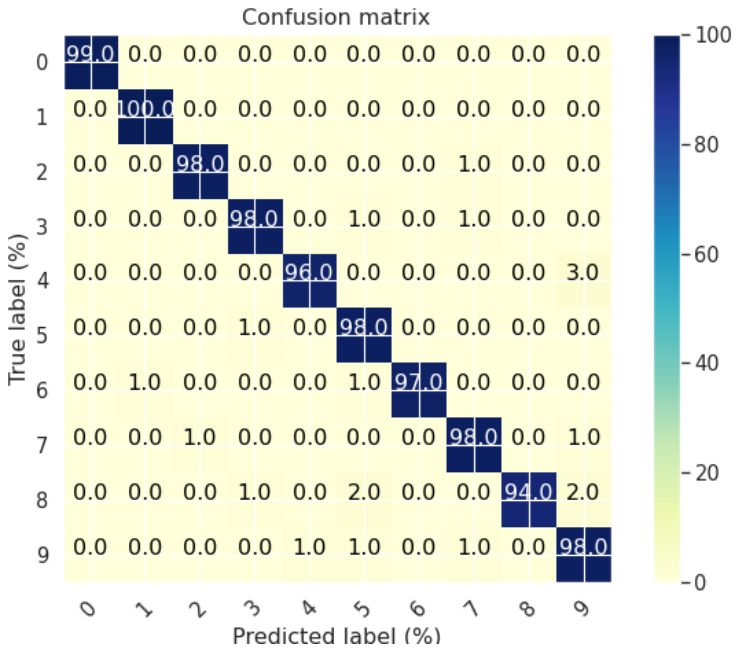


**Figure 8.** Confusion matrix of trained Client agent for Experiment 2.

### *4.3. Experiment 3: Local Learning Effect*

In the third experiment, using our *FLaMAS* tool, we sought to analyze the effect of delaying communication with the Server agent giving more epochs for local training in the Client agents. To analyze this effect, we tried three different deployments, with Client agents training one local epoch (see Figure 9), two local epochs (see Figure 10), or five local epochs (see Figure 11).

It is worth underlining how easily this mechanism benefited from the *force of numbers*, as can be observed in the first deployment with only one epoch for each local training. Figure 9 shows how the average accuracy increased hugely between the first global epoch (the result of one epoch of local isolated training), and the second global epoch (where the Client agent had received the average model from the Server agent and added another local training epoch to this new model).

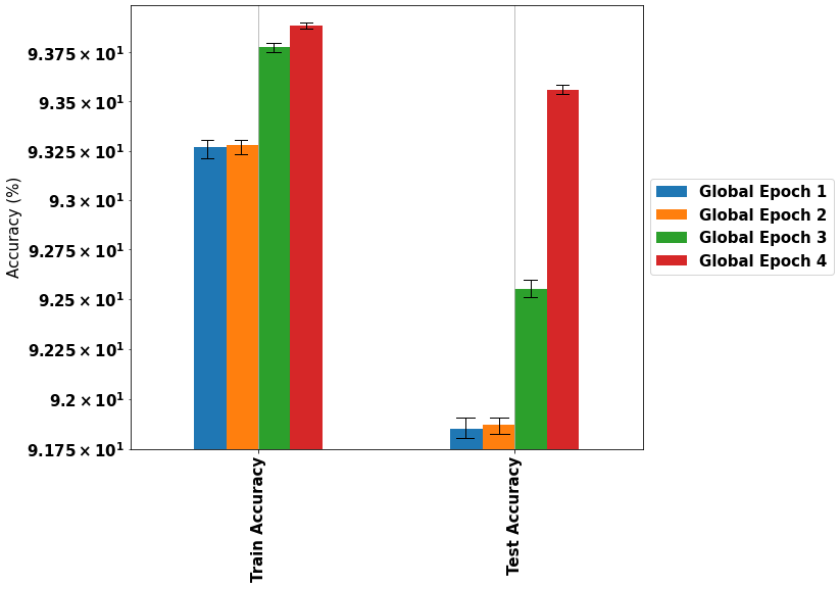


**Figure 9.** Experiment 3.1: Local learning effect with three agents, one local epoch, and four global epochs.

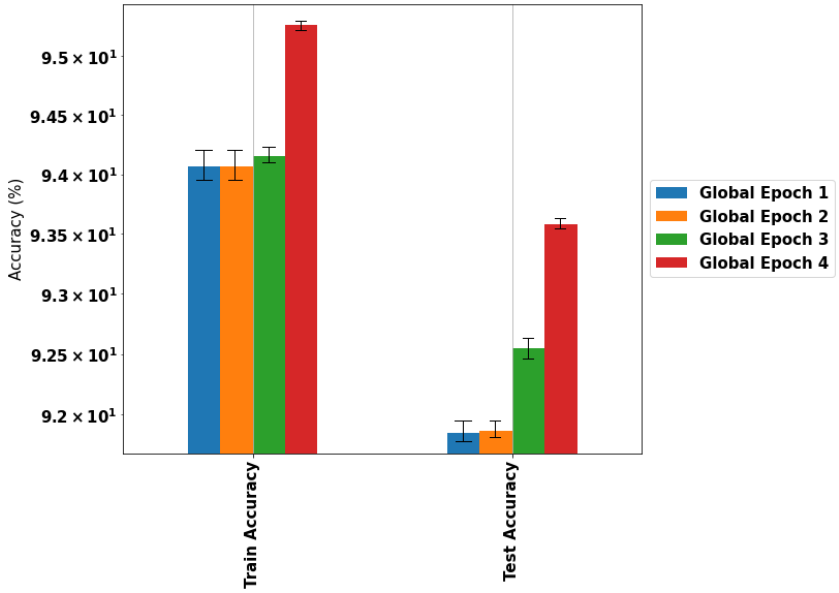


**Figure 10.** Experiment 3.2: Local learning effect with three agents, two local epochs, and four global epochs.

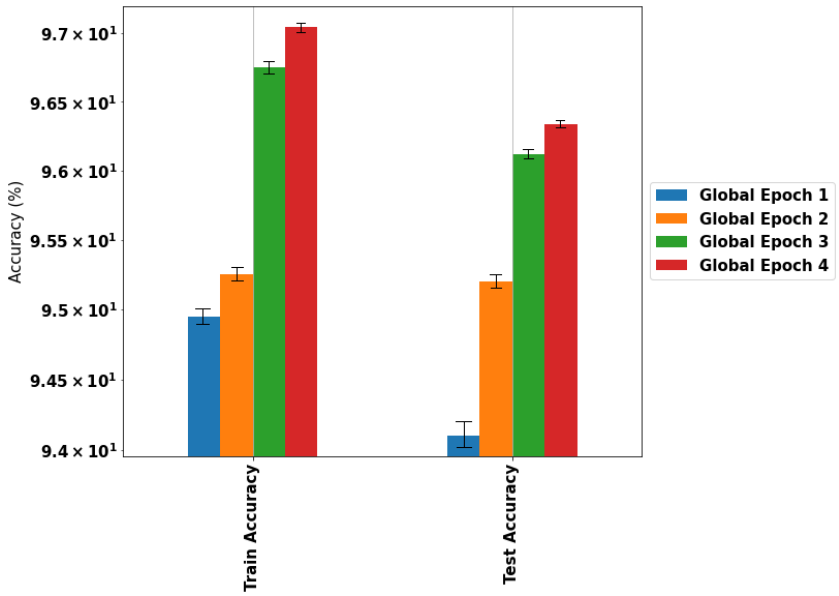


**Figure 11.** Experiment 3.3: Local learning effect with three agents, five local epochs, and four global epochs.

### *4.4. Experiment 4: Adding New Agents*

In the fourth experiment, using our *FLaMAS* tool, we sought to analyze the effect of adding new agents to the learning process during the execution at different times.

As can be seen in Figure 12, this experiment used three Client agents that entered the system at different moments (initially, there was only agent 1 active, then after a while, agent 2 entered the system, and lastly, agent 3 incorporated itself into the learning process.

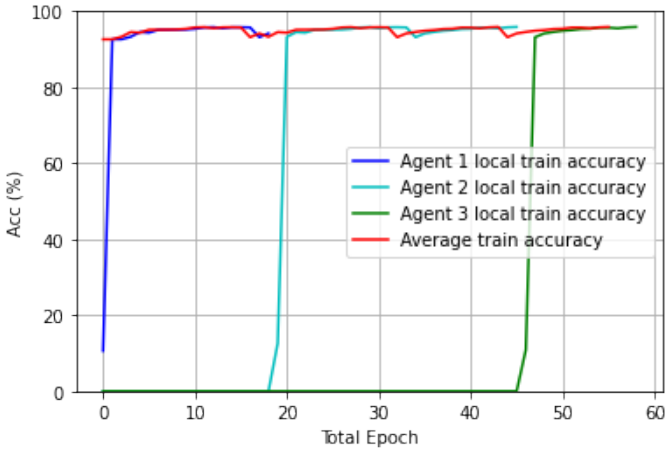


**Figure 12.** Experiment 4: Effect of adding new agents.

According to the results obtained, the dynamic entry of new agents into the system was performed without affecting the system's performance, and they were immediately adjusted to the appropriate weights. In this way, higher flexibility was achieved, since the failure of any client could be easily solved without affecting the global performance. In addition, the scalability of the system was facilitated, if necessary.

To conclude, a record was taken of the different parameters in each experiment to be compared (accuracy, training time and the number of epochs); the results obtained are summarized in the following Table 1. The distributed learning performed with *FLaMAS* achieved the goal proposed (with even more accuracy than the centralized learning) in lower epochs than the centralized experiment, and in a meaningfully lower amount of time. These numbers could be improved if we increased the number of Client agents. Moreover, as commented above, the usage of *FLaMAS* not only facilitated an easy communication mechanism, but also enabled adaptation to agents failing and to new agents entering the system.

**Table 1.** Comparison table of experiment results.

| | **Experiment 1** | **Experiment 2** | **Experiment 3** | | | **Experiment 4** |
|---|---|---|---|---|---|---|
| | | | 3.1 | 3.2 | 3.3 | |
| Accuracy (%) | 95 | 96.5 | 93.56 | 93.58 | 96.49 | 97.1 |
| Training Time (minutes) | ≈5 | ≈3 | ≈1 | ≈3 | ≈7 | ≈10 |
| N-Global Epochs | 20 | 5 | 4 | 4 | 4 | 4 |
| Mean (%) | 94.69 | 95.10 | 94.68 | 94.93 | 94.95 | 95.05 |
| Variance | 1.17 | 0.52 | 1.16 | 0.83 | 0.78 | 0.58 |

A statistical significance test was used. Ten experiments were conducted to determine whether the results obtained were non-random between the centralized training and one of the distributed training agents. A total of 211 accuracy values were collected. Before testing, we considered whether the two accuracy populations had equal variances. We can assume that the populations have equal variances if the ratio of the variance of the larger sample to the variance of the smaller sample is less than 4:1. The ratio of the variance of the larger sample to the variance of the smaller sample was 12.46/8.48 = 1.46, which is less than 4. This means that we can assume that the variances of the populations were equal. The t-test statistic was 2.29 and the corresponding two-sided p-value was 0.022. The p-value of the test was 0.022, which is below the alpha significance level (e.g., 0.05). This means that

we can conclude that the accuracy of the distributed model was statistically different from that of the centralized training.

## 5. Conclusions and Future Work

This paper has presented FlaMAS, a federated learning tool using multi-agent systems, in order to study the advantages of distributed learning. FlaMAS is a real distributed learning tool, where each of the agents is located in a different machine. Using message-based communication supported by SPADE, the agents communicate the weights to a central agent, which calculates the average weights. This average is then communicated and introduced into the model of each of the client agents. In comparison to other approaches, such as [26], in which the agents are simulated within the same machine, this real distribution capability enables consideration of how to improve communication timings, which are crucial in federated learning. The implementation of the proposed framework is available at GitHub (https://github.com/jarain78/FlaMASv0.1 (accessed on 28 February 2022)) which should be run on the SPADE platform.

Several experiments were conducted to compare the advantages of distributed versus centralized learning. Three variables were taken into account for this comparison: accuracy, training time and N-epochs. The results obtained showed that distributed learning achieved higher accuracy in a shorter time and with a smaller number of epochs. However, when the number of agents increased, the training time was affected, as more messages were required between the agents and the central agent. The dynamic entry of agents is a significant advantage as it facilitates the flexibility and scalability of the system.

Future work will focus on how to reduce the training time, as well as on how to optimize the messages passing between agents.

**Author Contributions:** Conceptualization, J.R.; formal analysis, J.R.; investigation, J.R.; methodology, C.C.; project administration, V.J.; supervision, C.C. and V.J.; writing—original draft, J.R.; writing—review and editing, J.R., C.C. and V.J. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was partially supported by the MINECO/FEDER RTI2018-095390-B-C31 project of the Spanish government.

**Conflicts of Interest:** The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

## References

1. Cisco Systems Inc. *The Role of Technology in Powering an Inclusive Future*; Cisco: San Francisco, CA, USA, 2020.
2. Ray, P.P. A survey of IoT cloud platforms. *Future Comput. Inform. J.* **2016**, *1*, 35–46. [CrossRef]
3. Voigt, P.; Von dem Bussche, A. The eu general data protection regulation (gdpr). In *A Practical Guide*, 1st ed.; Springer International Publishing: Cham, Switzerland, 2017; Volume 10, pp. 10–5555.
4. Goldman, E. An introduction to the california consumer privacy act (CCPA). *Santa Clara Univ. Leg. Stud. Res. Pap.* **2020**. http://dx.doi.org/10.2139/ssrn.3211013. [CrossRef]
5. Yong, W.; Quan, B. Data privacy law in Singapore: The personal data protection act 2012. *Int. Data Priv. Law* **2017**, *7*, 287–302.
6. Brendan McMahan, H.; Moore, E.; Ramage, D.; Hampson, S.; Agüera y Arcas, B. Communication-efficient learning of deep networks from decentralized data. *arXiv* **2016**, arXiv:1602.056292.
7. Li, T.; Sahu, A.K.; Talwalkar, A.; Smith, V. Federated learning: Challenges, methods, and future directions. *IEEE Signal Process. Mag.* **2020**, *37*, 50–60. [CrossRef]
8. Li, L.; Fan, Y.; Tse, M.; Lin, K.Y. A review of applications in federated learning. *Comput. Ind. Eng.* **2020**, *149*, 106854. [CrossRef]
9. Nguyen, D.C.; Ding, M.; Pathirana, P.N.; Seneviratne, A.; Li, J.; Poor, H.V. Federated learning for internet of things: A comprehensive survey. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 1622–1658. [CrossRef]
10. Savazzi, S.; Nicoli, M.; Bennis, M.; Kianoush, S.; Barbieri, L. Opportunities of federated learning in connected, cooperative, and automated industrial systems. *IEEE Commun. Mag.* **2021**, *59*, 16–21. [CrossRef]
11. Palanca, J.; Terrasa, A.; Julian, V.; Carrascosa, C. SPADE 3: Supporting the New Generation of Multi-Agent Systems. *IEEE Access* **2020**, *8*, 182537–182549. [CrossRef]
12. Vepakomma, P.; Gupta, O.; Swedish, T.; Raskar, R. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv* **2018**, arXiv:1812.00564.

13. Kairouz, P.; McMahan, H.B.; Avent, B.; Bellet, A.; Bennis, M.; Bhagoji, A.N.; Bonawitz, K.; Charles, Z.; Cormode, G.; Cummings, R.; et al. Advances and open problems in federated learning. *arXiv* **2019**, arXiv:1912.04977.
14. Rahman, K.J.; Ahmed, F.; Akhter, N.; Hasan, M.; Amin, R.; Aziz, K.E.; Islam, A.M.; Mukta, M.S.H.; Islam, A.N. Challenges, applications and design aspects of federated learning: A survey. *IEEE Access* **2021**, *9*, 124682–124700. [CrossRef]
15. Kholod, I.; Yanaki, E.; Fomichev, D.; Shalugin, E.; Novikova, E.; Filippov, E.; Nordlund, M. Open-Source Federated Learning Frameworks for IoT: A Comparative Review and Analysis. *Sensors* **2021**, *21*, 167. [CrossRef] [PubMed]
16. Khan, L.U.; Saad, W.; Han, Z.; Hossain, E.; Hong, C.S. Federated learning for internet of things: Recent advances, taxonomy, and open challenges. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 1759–1799. [CrossRef]
17. Zhang, T.; Gao, L.; He, C.; Zhang, M.; Krishnamachari, B.; Avestimehr, S. Federated Learning for Internet of Things: Applications, Challenges, and Opportunities. *arXiv* **2021**, arXiv:2111.07494.
18. Singh, S.; Rathore, S.; Alfarraj, O.; Tolba, A.; Yoon, B. A framework for privacy-preservation of IoT healthcare data using Federated Learning and blockchain technology. *Future Gener. Comput. Syst.* **2021**, *129*, 380–388. [CrossRef]
19. Lu, Y.; Huang, X.; Dai, Y.; Maharjan, S.; Zhang, Y. Blockchain and federated learning for privacy-preserved data sharing in industrial IoT. *IEEE Trans. Ind. Inform.* **2019**, *16*, 4177–4186. [CrossRef]
20. Chen, Y.; Qin, X.; Wang, J.; Yu, C.; Gao, W. Fedhealth: A federated transfer learning framework for wearable healthcare. *IEEE Intell. Syst.* **2020**, *35*, 83–93. [CrossRef]
21. Savazzi, S.; Nicoli, M.; Rampa, V. Federated learning with cooperating devices: A consensus approach for massive IoT networks. *IEEE Internet Things J.* **2020**, *7*, 4641–4654. [CrossRef]
22. Hard, A.; Rao, K.; Mathews, R.; Ramaswamy, S.; Beaufays, F.; Augenstein, S.; Eichner, H.; Kiddon, C.; Ramage, D. Federated learning for mobile keyboard prediction. *arXiv* **2018**, arXiv:1811.03604.
23. McMahan, B.; Ramage, D. Federated Learning: Collaborative Machine Learning without Centralized Training Data. *Google A Blog*. 2017. Available online: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html (accessed on 10 March 2022).
24. Peterson, D.; Kanani, P.; Marathe, V. Private federated learning with domain adaptation. *arXiv* **2019**, arXiv:1912.06733.
25. Palanca, J.; Rincon, J.; Julian, V.; Carrascosa, C.; Terrasa, A. Developing IoT Artifacts in a MAS Platform. *Electronics* **2022**, *11*, 655. [CrossRef]
26. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-efficient learning of deep networks from decentralized data. In Proceedings of the Artificial Intelligence and Statistics, PMLR, Fort Lauderdale, FL, USA, 20–22 April 2017; pp. 1273–1282.