BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 73(3), 2025, Article number: e154062
DOI: 10.24425/bpasts.2025.154062

Improving testing of multi-agent systems: An innovative deep learning strategy for automatic, scalable, and dynamic error detection and optimisation

Nour El Houda DEHIMI[1]*, Zakaria TOLBA[1], Mehdi MEDKOUR[1], Anis HADJADJ[1], and Stéphane GALLAND[2]

[1] LIAOA Laboratory, Department of Mathematics and Computer Science, University of Oum El Bouaghi, Algeria
[2] UTBM, CIAD UR 7533, F-90010 Belfort, France

Abstract. In this paper, a novel method is introduced for automated, scalable, and dynamic identification of errors in various behavioural versions of a multi-agent system under test, employing deep learning techniques. It is designed to enable accurate error detection, thus opening new possibilities for improving and optimising traditional testing techniques. The approach consists of two phases. The first phase is the training of a deep learning model using randomly generated inputs and predicted outputs generated from the behavioural model of each version. The second phase consists of detecting errors in the multi-agent system under test by replacing the predicted outputs with which the model is trained with execution outputs. The envisioned strategy is put into action through a real case study, which serves to vividly showcase and affirm its practical efficacy.

Keywords: multi-agent systems; system-level testing; error detection; artificial intelligence techniques; deep learning.

1. INTRODUCTION

In the dynamic field of computing, novel programming paradigms continually emerge to address the escalating demands of computer systems. Among these, the agent paradigm [1] is a pivotal development technology for intricate and distributed systems [2–4]. It has not only proven itself, but has also reshaped perceptions, challenging the conventions of more traditional paradigms like object-oriented programming. This is attributed to the distinctive functional and behavioural characteristics inherent in this paradigm. Features such as autonomy, reactivity, proactivity, and the dynamic shifts in behaviour resulting from the acquisition or release of roles by these agents contribute to its uniqueness. Furthermore, the organisational concepts associated with this paradigm are relatively novel, further improving its appeal and innovative nature for the modelling of complex systems.

The new characteristics of this paradigm pave the way for several lines of research at different levels of design, in particular, software testing. Several agent-orientated methodologies with a clear organisational vision have been created in recent years to support modelling complex systems, such as agent, group and role (AGR) [5], agent-orientated software process for complex engineering systems (ASPECS) [6], extended gaia [7], engineering for software agents (INGENIAS) [8], Model of multi-agent systems organisation (Moise+) [9] and OperA [10]. All these methodologies provide methods and tools for the design and implementation of multi-agent systems (MAS). Within these methodologies, testing agent code and system behaviour is not extensively addressed. Testing systems developed using the agent paradigm is an important task in the quality assurance process. It can be very useful for identifying faults and validating the system under test [11, 12]. Its purpose is to examine or execute a program to reveal errors and thus increase confidence in the software. It is often defined as how it is possible to ensure that an implementation conforms to what has been specified. Unlike other types of system, testing MAS is a challenging task [13] due to various factors such as the autonomous nature of the agents, the nonreproducible effect, which implies that two system executions using the same input data may not produce the same state, the simultaneous and independent execution of multiple agents, the manipulation of a substantial amount of data by each agent, each with its objectives, the unpredictable evolution of the agents' behaviour, and the growing complexity associated with the distributed nature of applications composed of multiple agents. Consequently, despite the rapid evolution of MAS, there are few proposals for testing MAS in the literature. Furthermore, most of these proposals are related to unit-level testing, which consists of testing all units that comprise an agent [14–16], and agent-level testing,

*e-mail: dehimi.nourelhouda@univ-oeb.dz

Manuscript submitted 2024-11-21, revised 2025-01-12, initially accepted for publication 2025-02-26, published in May 2025.

which consists of *(i)* testing the integration of the different modules, tested at the unit level, within an agent, *(ii)* testing the agent's ability to achieve its goal in its environment [17, 18]. System-level testing, which ensures that all agents in the system operate according to specifications and interact correctly, has attracted little interest from researchers. In fact, in the literature, only a few approaches have been proposed for system-level testing [19–23].

Although these works have made considerable progress in the field of MAS testing by proposing new strategies, they are, on the other hand, very complex. This complexity primarily stems from their focus on developing test case generation algorithms aimed at covering the very sophisticated characteristics of MAS. However, this complexity also poses challenges in terms of generalisation and adaptability across different MAS contexts. Existing approaches often struggle to adapt to the unpredictable evolution of MAS behaviour. This limitation suggests the need for more flexible and adaptive testing methodologies that can accommodate the dynamic nature of MAS.

To overcome this problem, in this work, a new approach is proposed to automatically detect and diagnose software errors in MAS by replacing test case generation algorithms with a deep learning model [24]. This aims to provide a powerful and innovative solution capable of improving accuracy, adaptability, execution speed, and reducing dependency on expert knowledge, thus overcoming the problems imposed by traditional testing techniques.

Deep learning has already found promising applications in software testing, revolutionising traditional testing methodologies [25–29]. Deep learning models can be trained to automatically detect and diagnose software defects, vulnerabilities, and performance problems. These models excel at analysing considerable amounts of code, identifying patterns, and predicting potential failure points. The use of machine learning [30] and deep learning for software testing has been mentioned in certain studies [31–37]. These studies highlight the effectiveness of deep learning in detecting anomalies and automating the testing process. They demonstrate how neural networks can adapt to diverse software environments and provide accurate predictions, significantly reducing the time and effort required for testing. The application of deep learning to software testing not only improves the efficiency of the testing process, but also contributes to the overall improvement of software quality and reliability. Despite the numerous research on the use of deep learning for automatic software tests, none of them are specifically on the key properties of a MAS, such thatits distributed nature and the unpredictable evolution of its behaviour. The remainder of this paper is organised as follows: Section 2 provides a brief overview of the main related work. Section 3 describes the motivations and research gaps in this field. Section 4 describes the proposed approach and its different phases. Section 5 presents the multi-agent system chosen for testing. Section 6 illustrates the formation of the deep learning model. The testing of the multi-agent system using the deep learning model presented in Section 6 is demonstrated in Section 7. Section 8 presents the developed tool. Section 9 presents some conclusions and recommendations for future work.

2. RELATED WORKS

In the literature, a limited number of approaches have been proposed for testing multi-agent systems in recent years. Subsequently, we outline a selection of these approaches in the following.

Shafiq *et al.* [38] introduce an innovative method to test multi-agent systems using Prometheus design artefacts. In this proposed approach, various interactions between agents and actors are considered to evaluate the multi-agent system. These interactions encompass perceptions, actions, and message exchanges between agents, which are represented in a protocol diagram. Subsequently, the protocol diagram is transformed into a protocol graph, upon which various coverage criteria are employed to produce test paths covering agent interactions. To facilitate this process, a prototype tool has been developed to generate test paths from the protocol graph based on the specified coverage criterion.

In Dehimi [39], a novel model-based testing method is introduced for holonic agents. This method uses genetic algorithms and considers the evolution of an agent over successive versions. The process is divided into two main phases that are carried out iteratively. Initially, the focus is on identifying a new version of the agent under examination. Subsequently, the focus is on testing each newly identified version. The analysis of the new agent version aims to construct a behavioural model, facilitating the generation of test cases. Notably, the test case generation process emphasises the examination of the new or modified aspects of agent behaviour. This approach enables an incremental enhancement of test cases, addressing a pivotal concern in testing methodology.

Bakar and Selamat [40] thoroughly examined the testing methods applicable to agent systems, delineating the range of properties and faults detectable by current techniques. The primary objective was to pinpoint areas of research deficiency and outline future directions for the verification of agent systems.

Barnier *et al.* [41] conducted a comparison between testing methodologies for software and multi-agent systems, focusing specifically on embedded contexts. This research delved into various testing techniques for multi agent systems, analysing them with respect to the AEIO facets. The aim was to offer a tailored testing approach for multi-agent systems on embedded platforms. The suggested method consisted of three fundamental levels: individual agent testing, collective resources, and acceptance testing.

Winikoff [42] suggested a method to assess the testability of BDI agents (belief-desire-intention), comparing two distinct adequacy criteria: the 'all edges' criterion, which is met when a set of tests covers all edges in a control flow graph, and the 'all-paths' criterion, where a test set is adequate if it covers all paths in the control flow graph of the program. The level of testability achieved was used to determine the number of test cases required to validate a BDI program.

Gonçalves *et al.* [43] introduced a comprehensive framework to analyse and test the social aspect of MAS within the Moise+ organisational model. This framework employs a set of Moise+ specifications as an information artefact mapped into a coloured Petri net (CPN) model known as CPN4M, serving as a mecha-

nism for generating test cases. CPN4M operates with two distinct test adequacy criteria: all paths and state-transition path. The paper formalises the process of transitioning from Moise+ to CPN, outlines the procedures for generating test cases, and performs the tests in a case study scenario. The findings suggest that this methodology has the potential to enhance the accuracy of the social aspect in a multi-agent system specified by a Moise+ model, highlighting the system context that may lead to MAS failure.

Shafiq *et al*. [44] presented a model-based methodology to ensure comprehensive coverage of goals and plans within multi-agent systems. A fault model has been established to encompass faults related to the execution of the objectives and plans and interactions within MAS. This approach utilises Prometheus design artefacts, such as the goal overview diagram, scenario overview, and agent and capability overview diagrams, to construct a test model. Additionally, new coverage criteria have been formulated for fault detection purposes. Test paths are identified from the test model, and test cases are subsequently generated from these paths. The effectiveness of this technique is evaluated using the JACK Intelligent Agents framework, which is a Java-based platform for multi-agent system development. More than 100 different test cases are executed on the actual implementation of MAS, with code instrumentation used for coverage analysis and fault injection. The results demonstrate the successful detection of injected faults by applying test cases aligned with the coverage criteria paths during the execution of MAS. Particularly, the 'Goal Plan Coverage' criterion proves to be more effective in fault detection compared to scenario, capability and agent coverage criteria, which exhibit relatively lower effectiveness in identifying faults.

Huang *et al*. [45] implement semantic mutation testing (SMT) in three rule-based agent programming languages: Jason, GOAL, and 2APL. Describes various scenarios where SMT is beneficial for these languages. In addition, it introduces three sets of semantic mutation operators rules designed to induce semantic changes for each language and presents a systematic method to derive such operators for rule-based agent languages. The paper further demonstrates, through initial evaluation, that the proposed semantic mutation operators for Jason showcase the potential of SMT in evaluating tests, robustness, and reliability concerning semantic alterations.

In Dehimi *et al*. [46], we introduced a novel test case generation approach designed to address individual behavioural scenarios within a multi-agent system. The primary objective is to isolate the scenario responsible for any errors detected between concurrently running scenarios. To achieve this, our approach employs mutation analysis and parallel genetic algorithms in the initial stage. These techniques help identify situations in which agents execute interactions outlined in the sequence diagram of the target scenario exclusively, which then serve as inputs for the test case. In the subsequent stage, our approach utilises the activities depicted in the activity diagram to determine the expected outputs of the test case corresponding to its inputs. The generated test cases are subsequently utilised for error detection purposes.

In Dehimi *et al*. [47], we presented a novel approach to generating test cases capable of accommodating the new interactions introduced due to the unpredictable evolution of the behaviour of a multi-agent system under test. Our approach dynamically applies a model-based testing methodology for each new version of the system under examination. Specifically, it utilises *(i)* the AUML sequence diagram of each new system version to derive test cases capable of addressing the newly introduced interactions. *(ii)* Constraints expressed in Object Constraint Language (OCL) to ensure the execution of each interaction, thereby considering the specificities of interactions between agents, including inclusive, exclusive, or parallel execution. *(iii)* Genetic algorithms to assess the incorporation of new interactions into the system under test. This approach facilitates comprehensive test coverage of evolving MAS behaviour and enables the identification of potential issues arising from system updates. Although significant progress has been made in MAS testing through the introduction of new strategies, there are also significant shortcomings in these research efforts. The following section highlights these gaps and describes proposed contributions to address them.

3. RESEARCH GAPS AND OPEN ISSUES

Identifying research gaps in traditional testing techniques for MAS is imperative to advance the field and addressing inherent challenges. Traditional methods may encounter scalability issues, struggling to efficiently scale with an increasing number of agents within a system. The dynamics of interactions among numerous agents can create complex scenarios that these methods may not adequately address. Furthermore, many traditional testing approaches may lack adaptability to the dynamic and evolving nature of MAS, especially in real-world scenarios with changing conditions. Nondeterministic behaviour, inherent in some MAS, poses a challenge for conventional testing methodologies, as agents' actions can be influenced by external factors, leading to variations in system behaviour. Emergent behaviour challenges, where unexpected behaviours arise from agent interactions, may be overlooked by conventional testing, requiring enhancements to capture and validate these unexpected behaviours. The inadequate coverage of diverse interaction scenarios among agents further underscores the need to enhance existing testing methods. Furthermore, traditional testing approaches may not be well-equipped to handle learning-based agents, such as those employing reinforcement learning, as they adapt and evolve over time. The absence of standardised testing frameworks tailored for MAS is another gap that hinders a common ground for evaluating and comparing methodologies. Lastly, limited consideration of security aspects in traditional MAS testing techniques, especially in critical applications, necessitates the development of specialised testing approaches. Addressing these research gaps is crucial for the development of more effective and comprehensive testing methodologies specifically tailored to the unique challenges posed by MAS.

The main contribution of this research lies in the introduction of a novel deep learning approach that addresses critical research gaps in traditional MAS testing techniques. By systematically identifying and addressing these gaps, the study aims

to significantly improve the accuracy of error detection and optimisation processes within MAS. The proposed deep learning approach offers a solution to scalability issues, providing an efficient method for handling a growing number of agents within a system. It also focuses on adaptability to dynamic environments, accommodating the evolving nature of MAS in real-world scenarios. Moreover, the approach tackles the challenges posed by nondeterministic behaviour, emergent behaviour, inadequate coverage of interaction scenarios, and limited support for learning-based agents. By integrating these advancements, the research contributes to overcoming the limitations of traditional testing methodologies, paving the way for more robust, comprehensive, and adaptive testing techniques tailored to the intricate dynamics of MAS. This innovative deep learning approach holds promise for furthering the field and addressing the complex testing requirements inherent in MAS.

4. THE PROPOSED APPROACH

Our approach aims to automatically detect possible errors in the different behavioural versions V_i of a multi-agent system under test that are obtained following the acquisition and/or release of roles by the agents that comprise it. It consists of two main phases (Fig. 1):

The first phase consists of training a deep learning model whose training data set consists of n subsets of data. Each data subset is built primarily for a V_i behavioural version of the system under test. Indeed:

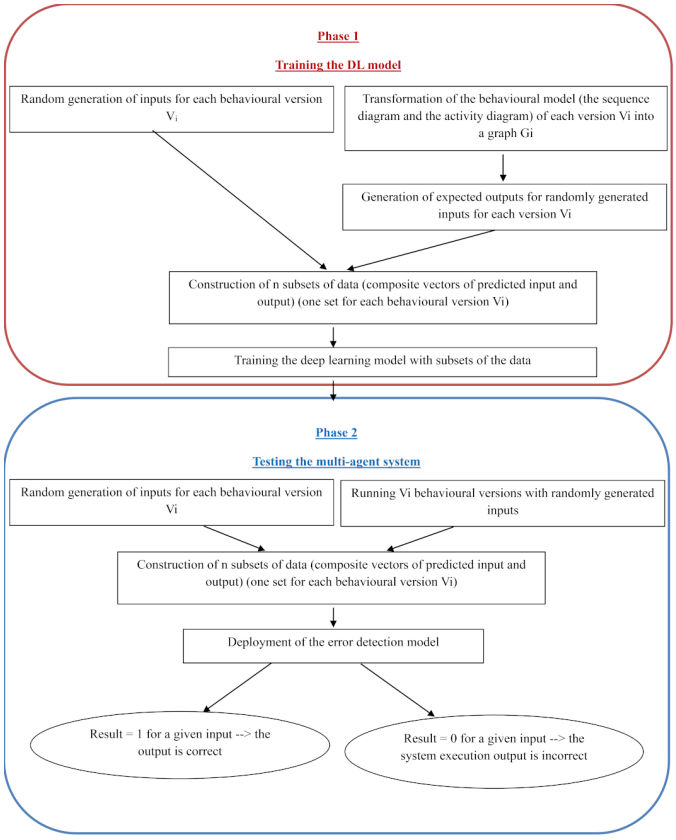

Fig. 1. The phases of the approach

- Each input for each subset of data is presented as a vector consisting of: *(i)* a possible input $Input_q$ to execute one of the possible behavioural scenarios S_j of the system under test in its behavioural version V_i. This entry is randomly generated from the execution interval of the behavioural scenario S_j, *(ii)* the expected output $Output_q$ for this input according to the V_i behavioural version of the system under test. This output is generated from the behavioural model of the V_i version of the system under test. The behavioural model used for this is the sequence diagram and the activity diagram of version V_i, which will be transformed into a G_i graph to automate this operation. In this graph, each path P_j represents a behaviour scenario S_j, with each node n_k representing an interaction between two agents in the sequence diagram.
  It contains the information needed to generate the expected outputs for the randomly generated inputs, namely, activities n_k to be executed by the receiving agent will execute following receipt of the interaction represented by that node. To generate the expected outputs for the inputs randomly generated using the graph G_i, suffices that, for each input $Input_q$ belonging to the execution interval of the behaviour scenario S_j, we simply traverse the path P_j of the graph G_i, node by node, and for each node n_k of the path P_j, we calculate the results of applying the activities n_k. Activities of that node on $Input_q$. The following algorithm (Fig. 2) summarises the process of generating expected outputs.
- The output of each input in each subset of data can take two values: '1' if the expected output of the randomly generated input is correct (generated from the behavioural model), meaning that the system behaves correctly, and '0' if the expected output of the randomly generated input is not correct, meaning that the system behaves incorrectly.

```
Algorithm Expected_Outputs_Generation
  Inputs: The graph Gi
  Outputs: Expected Outputs
  Begin
      For each path Pj∈P do
        For each Inputq generated from the execution interval of the
                  behavioural scenario Sj
                  Outputq = Φ ;
              For each nk∈ Pj
                Outputk= Φ
                  For each Activityl∈ nk.Activitys
                    Outputl=Activityl(Inputl) // Inputlis a part of
                    Inputqnecessary for the execution of the Activityl
                    Outputk = Outputk∪ Outputl;
                  End For;
                Outputq = Outputq∪Outputk;
              End For;
        End For;
    End For;
  End.
```

Fig. 2. Expected outputs generation algorithm

The second phase consists of testing the different V_i behavioural versions of the system under test using the deep learning model trained and validated in the first phase. To do this, this phase begins by generating, for each V_i behavioural version of

the system under test, a subset of inputs that will later be used to deploy the model to detect any errors in this version. Unlike the inputs used for training the model, each vector of these inputs is made up of a randomly generated input, possible for the execution of the system under test, and the execution output of the system under test in its V_i behavioural version with this input. The result of deploying the model with these inputs facilitates error detection. Indeed, if the result is 1 for a given input, this means that the system execution output is correct (conforms to the expected outputs with which the model is trained), which implies that there is no error; otherwise, if the result is 0, this means that the system execution output is incorrect (it does not conform to the expected outputs with which the model is trained), which implies that there is an error.

In what follows, we will first present the multi-agent system chosen for our study and its various behavioural versions. We will then present the first phase of our approach, which consists of training a deep learning model. To do this, we will address the following points: preparing subsets of data for each V_i behavioural version of the system under test, training, and validating the model. The second phase of our approach, which consists of testing the different V_i behavioural versions of the system under test using the trained model, will be presented at the end of this section.

5. PRESENTATION OF THE MULTI-AGENT SYSTEM UNDER TEST

The traffic control system chosen for our study aims to ensure traffic management by making decisions capable of minimising traffic disruption and delay, considering several variables such as weather conditions, road traffic, the presence of traffic jams, the presence of events, and accidents. In its first behavioural version V1, the system agents play the following roles:

- Weather agent: This agent is responsible for monitoring, in real time, weather conditions such as temperature, precipitation (particularly rain), and snowfall, using sensors located in key areas of the city and data supplied by climate monitoring services. Then it sends the collected data (temperature in degrees Celsius, amount of rain in millimeters, and amount of snow in millimeters) to the data collection agent to be used for effective traffic management.
- Environment agent: This agent is responsible for monitoring, in real time, congestion problems such as traffic jams or accidents. When the environment agent identifies a congestion problem, it sends the relevant information (accident line, accident type %, presence of traffic (binary value: 0 or 1), traffic line, traffic type) to the data collection agent.
- Event agent: This agent is responsible for monitoring, in real-time, events that may disrupt normal traffic flow, such as demonstrations, funeral processions, parades, the arrival of a democratic person, etc. It then sends the relevant information ((the presence of an event (binary value: 0 or 1), the line of the event, the type of event (unit not specified)) to the data collection agent.
- Data Collection Agent: This agent is responsible for gathering all important traffic information, including data provided by the Weather Agent, Environment Agent and Event Agent. Once it has collected all this data, the data collection agent sends it to other agents in the system that use this information to plan and manage traffic efficiently.
- Traffic light officer: This officer is responsible for controlling the traffic lights at an intersection or in a specific area. They use data collected by other agents to optimise signal times and reduce waiting times for drivers. These data can include information on current traffic conditions, the number of vehicles in each area, traffic density, etc. It sends to the decision agent to acquire additional information about the situation. These data can be used to improve the precision of traffic forecasts and to make more informed decisions about traffic management.
- Speed agent: This agent can also play a role in the speed of vehicles. It can receive information from the weather, environment, and event agent to adjust the recommended speed for drivers accordingly. By providing real-time information on the traffic situation, the speed agent can help regulate traffic flow and improve road safety.
- Lane agent: This agent plays an important role in the management of traffic lanes. It is responsible for monitoring the conditions of each lane in real time and making decisions about opening or closing lanes based on traffic demand. The lane agent can receive information from the weather, environment and event agent from the data collection agent to identify congestion areas and adapt the opening or closing of lanes accordingly. It can also inform drivers of the condition of each lane, which can help improve safety and traffic flow. Working in collaboration with other agents, the lane agent can help optimise road use and reduce driving times.
- Decision-making agent: This agent is responsible for making the final decision based on the information provided by the lane agent, the speed agent and the traffic light agent. Then sends its decision to the interface agent to be displayed.
- Interface agent (GUI): This is the interface agent that launches and supervises the system three main agents: the weather agent, the event agent and the environmental agent. Once the decision agent has decided, the interface agent displays this decision in a way that is clear and comprehensible to users. It provides detailed information on the measures taken, such as the adaptation of signaling, speed recommendations, or changes to lane opening. Thanks to this user-friendly interface, users can quickly understand the actions taken by the traffic management system to improve traffic flow and safety.

In the second behavioural version V2 of the system, the environment agent was given a new role, which enabled it to detect obstacles. This is due to the installation of a new sensor dedicated to obstacle detection. This enhancement has added crucial information to the list of transmitted data, such as the presence of obstacles, the line of the obstacle, and its identification number. These new data enable other agents to make informed decisions to avoid obstacles and ensure that traffic flows smoothly. As a result, data concerning obstacles have been added to the list of data sent to the data collection agent, including the presence of obstacles (binary value: 0 or 1), the line where the obstacle is located and the obstacle's identification number.

In the third version V3 of the system, the weather agent has acquired a new role thanks to the installation of a sensor dedicated to air detection. This new feature has added crucial data to the list of information transmitted to the data collection agent. These data include the presence of air (0 or 1) and the measurement of air density expressed in $PM_{2.5}$ up to 50 $\mu g/m^3$. This information is essential for other agents to make informed decisions and adopt appropriate measures to avoid potential air quality-related accidents on the roads. The integration of these new data strengthens the monitoring and risk management capabilities of the traffic control system.

6. FIRST PHASE (FORMATION OF THE DEEP LEARNING MODEL)

6.1. Generating data subsets

To take into consideration the three behavioural versions of the system under test, three subsets of data are generated. Each data item in each subset generated for the V_i behavioural version of the system is made up of an input (presented in the form of a vector made up of a randomly generated execution input and the expected output for this input generated automatically from the graph obtained following the transformation of the sequence and activity diagram for the V_i version using the **expected outputs generation algorithm**) and an output (. . . 0/1).

In our case, the input vector contains 37 integer elements, 17 of which represent the randomly generated input *(Temperature, Rain, Snow, Air_line, Air_density, Accidents, Acc_line, Acc_type, Obstacles, Obs_line, Obs_num, Traffic, Traffic_line, Traffic_type, Event, Event_line, and Event_type)*, and 20 elements represent the expected output (*Line1 to Line20: knowing that each line is open or closed, the maximum speed for each line, the traffic light, and the event time*).

For each version, we generated a data set of size = 4000. Therefore, the total size of the data set is 12 000.

6.2. Data set pre-processing

In this data pre-processing section, we start by importing the raw data set and then proceed to manage it for training the neural network. This management includes separating the data set into a training set and a test set, to evaluate the model performance on unknown data. Additionally, we apply scaling to the characteristics of the data set to facilitate computation when training the model. Scaling is an important step in data preparation, as it prevents certain features from dominating others in the learning process, which can distort the results. In our case, we trained our model with a data set comprising 9600 examples for training and 2400 examples for testing.

6.3. Model proposal

To carry out our study, we used several deep learning training models, including recurrent neural networks (RNN) [48, 49], artificial neural networks (ANN) [50], deep neural networks (DNN) [51], and convolutional neural networks [52]. Each of these models was tested on the data set to determine their respective effectiveness in the prediction task at hand. After careful evaluation, we found that the ANN model gave the most accurate and consistent results = 0.96 (Fig. 3). In what follows, we will describe in detail how we trained and evaluated our ANN model to obtain the best possible results. In the following, we will discuss model architecture, model training, model evaluation, and model improvement.

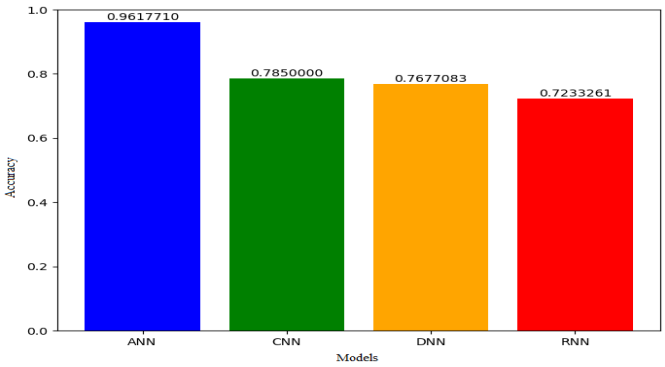

Fig. 3. Accuracy of the different models

– Model architecture: The architecture of our model (ANN) (Fig. 4) consists of an input layer, four hidden layers, and an output layer as follows:

- The input layer contains 37 nodes, one for each independent variable in the data set.
- Each hidden layer contains 19 nodes. The rectified linear rectification function (ReLU) is used as the activation function for these layers, which solves vanishing gradient problems and improves network convergence. A deactivation layer with a deactivation rate of 0.1 is added after each hidden layer to avoid overlearning.
- The output layer contains a single node, which returns a binary output (1 or 0) for the binary classification problem in question. The sigmoid function is used as the activation function for this layer.

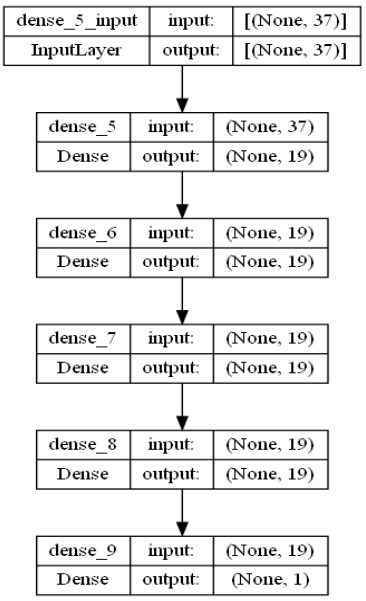

Fig. 4. Architecture of the ANN model

– Training the ANN: To train the model, we used the Keras library to compile the ANN by specifying the optimisation algorithm (Adam), the loss function (mean squared error), and the metrics (precision and recall). The ANN was trained on the training set with a predefined number of epochs and batch size. We also used a technique called cross-validation to evaluate the performance of the model on the validation set at each epoch. Visualising the training (Fig. 5) has allowed us to understand how the model classifies the different classes in the data space, which can help identify potential problems such as overfitting or underfitting. By visualising the results of the training set, it is also possible to check whether the model is capable of correctly separating the different classes in the data space. In addition, visualising the results of the training set can help to select the optimal hyperparameters for the neural network model, such as the number of hidden layers, the number of neurons in each layer, the learning rate, etc. The following figures (Figs. 6–9) demonstrate the error functions of the different models obtained during the training process.

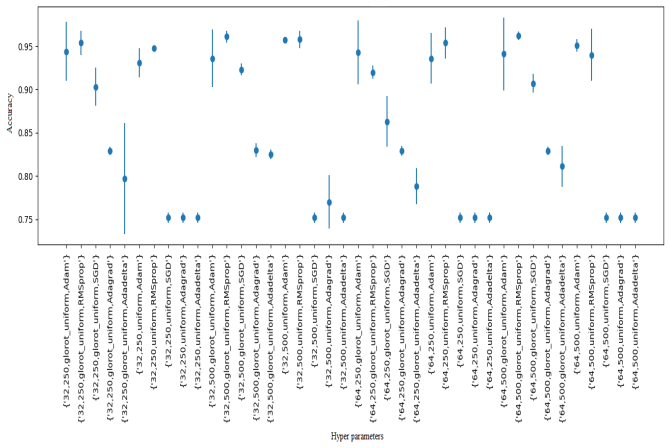

Fig. 5. Accuracy of the various hyper parameters

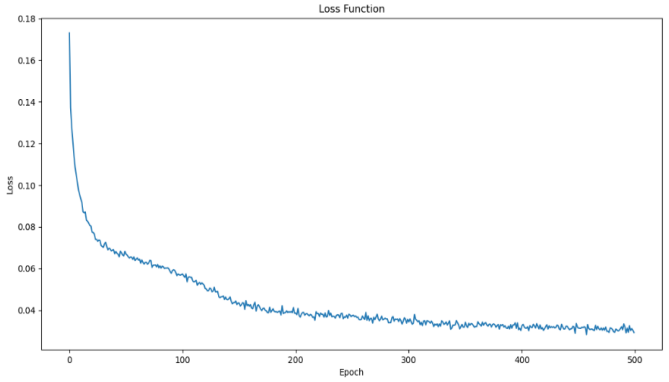

Fig. 6. Error function of the ANN model

– Model evaluation: In this section, we comprehensively evaluate the performance of the ANN model on the test set by calculating key evaluation metrics, including accuracy and loss, alongside additional performance measures such as the classification report and the confusion matrix. The classification report

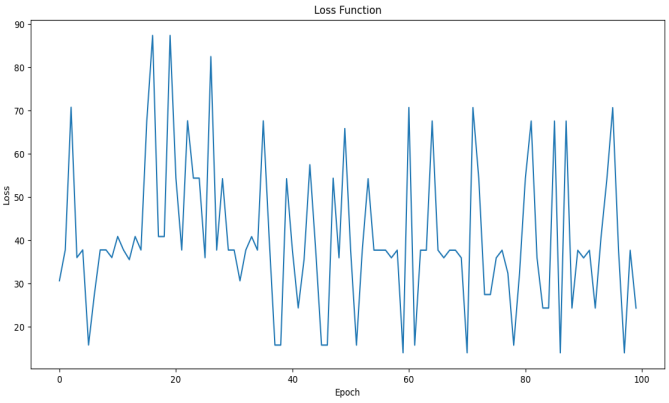

Fig. 7. Error function of the CNN model

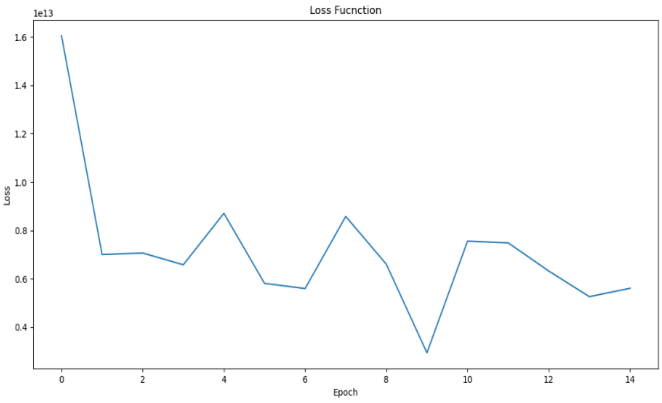

Fig. 8. Error function of the DNN model

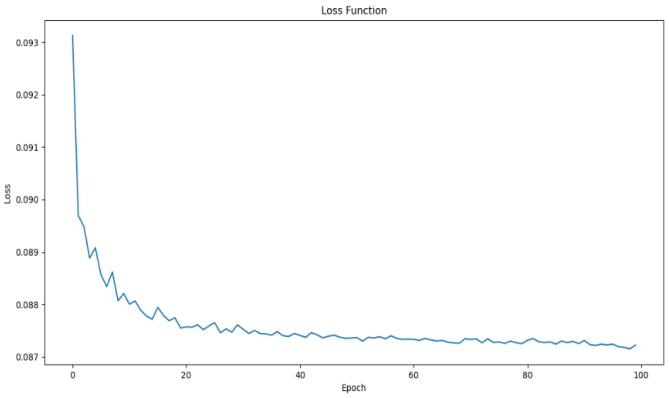

Fig. 9. Error function of the RNN model

provides detailed insights into the model behaviour, including precision, recall, F1-score, and support for each class, enabling a granular understanding of its strengths and weaknesses across different categories. The confusion matrix, on the other hand, offers a visual representation of the model classification errors, highlighting misclassifications and their distribution across classes. These metrics together provide a multi-faceted evaluation framework, ensuring a robust assessment of the model predictive capabilities. By integrating these results, we can critically analyze whether the ANN model is well-suited to the

data and meets the precision and reliability requirements for the specific task. Moreover, this multi-metric evaluation framework underscores the model performance not only in terms of overall accuracy but also in its ability to handle imbalanced datasets and diverse class distributions, thus validating its effectiveness and practical applicability.

– Model improvement: To improve the performance of the model, we use a grid search technique to explore different combinations of ANN hyperparameters (Fig. 10). This allowed us to find the optimal combination of hyper parameters, including the optimisation algorithm and the kernel initialisation method, which improved the performance of the model. We also evaluated the performance of the improved model on the performance of the test set to ensure that the model was significantly improved. Ultimately, this section optimises the performance of the model for better accuracy and generalisation.

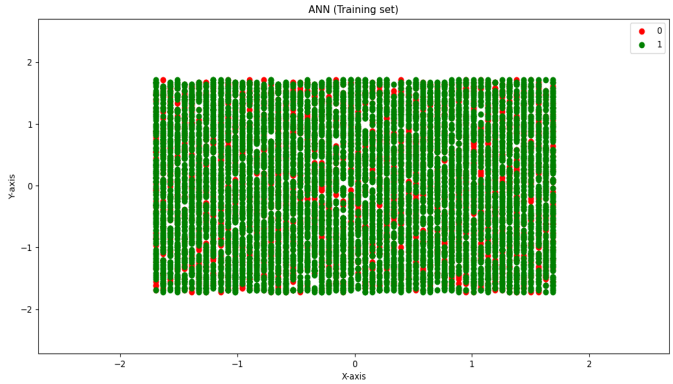

Fig. 10. ANN training set

7. THE SECOND PHASE (ERROR DETECTION USING THE TRAINED MODEL)

To detect possible errors in the traffic control system under test, we will present in this section the second phase of our approach which consists of testing the different behavioural versions V_i of this system using the deep learning model trained and validated in the first phase by replacing the part of the predicted outputs by execution outputs. In what follows, we will present for each behavioural version V_i of the system under test *(i)* part of the subset of inputs generated for the testing of this version, where each input consists of a randomly generated input, possible for the execution of the system under test, and the output of the execution of the system under test in its behavioural version V_i with this input, *(ii)* the results obtained by deploying the learning model with these inputs, where if the result is 1 for a given input, this means that the execution output of the system is correct (conforms to the predicted outputs with which the model is trained), which implies that there is no error; otherwise, if the result is 0, this means that the execution output of the system is incorrect (it does not conform to the predicted outputs with which the model is trained), which implies that there is an error. Finally, we will discuss the results obtained, highlighting the successes and limitations of our approach. The results shown in the following sections are obtained automatically using a software tool1 that we developed on Python.

7.1. Testing version V1

The following figure (Fig. 11) represents part of the subset of inputs generated for the first behavioural version V1 of the system under test and the results obtained by deploying the learning model with these inputs. The results of the deployment of the training model show that the V1 version of the system under test is error-free. This can be explained by the fact that the system behaviour is compatible with the expected behaviour with which the model was trained.

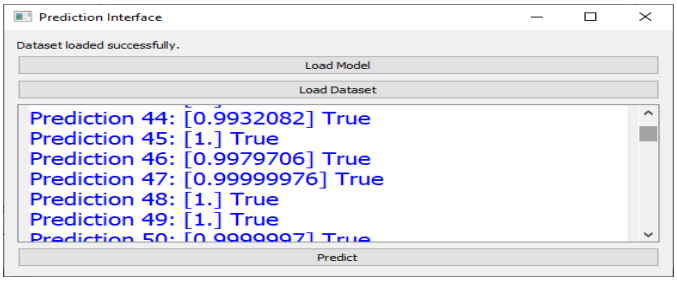

Fig. 11. Test results for Version 1

7.2. Testing version V2

In this version, the agents of the system under test were given new roles to consider the presence of obstacles. The following figure (Fig. 12) represents part of the subset of inputs generated for the behavioural version V2 of the system under test and the results obtained by deploying the learning model with these inputs.

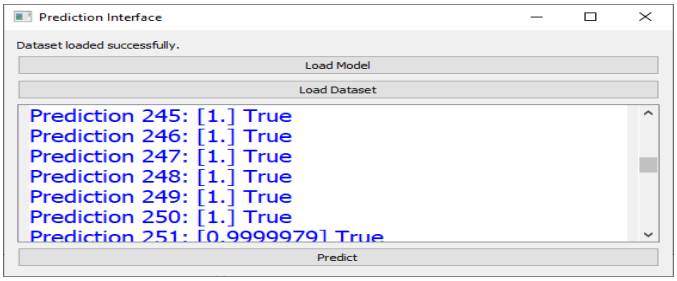

Fig. 12. Version 2 test results

The results of the deployment of the training model show that the V2 version of the system under test is error-free. This can be explained by the fact that the system behaviour is compatible with the expected behaviour with which the model was trained.

7.3. Testing version V3

In this version, the agents of the system under test were given new roles to consider the presence of air. The following figure (Fig. 13) represents part of the subset of inputs generated for the V3 behavioural version of the system under test and the results obtained by deploying the learning model with these inputs.

The results of the deployment of the training model show that the V3 version of the system under test contains errors. This can be explained by the fact that the system behaviour was not compatible with the expected behaviour with which the model was trained.

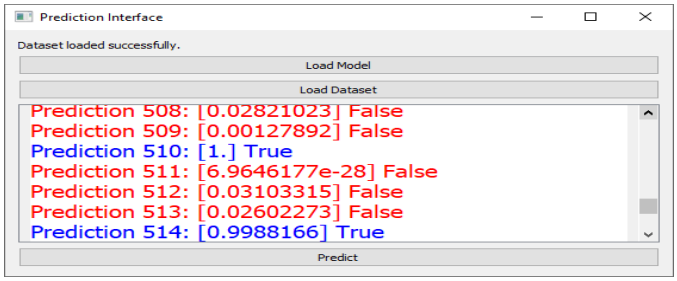

Fig. 13. Version 3 test results

8. DISCUSSION AND LIMITATIONS

The application of our approach to error detection in the various Vi behavioural versions of the traffic control system demonstrates several key advantages over existing approaches, directly responding to the challenges outlined in Section 3:

- Enhanced accuracy: Our deep learning model significantly improves error detection accuracy in MAS by leveraging the ability of deep learning techniques to identify subtle patterns and complex relationships within the data. This enables more precise error detection than traditional methods, particularly in dynamic and complex systems where traditional rule-based techniques often fall short.
- Adaptability to dynamic environments: One of the primary strengths of our approach is its adaptability. Deep learning models can be trained on system-specific datasets, allowing them to effectively handle the evolving nature of MAS. This adaptability is crucial for real-world scenarios, where agent behaviours and system configurations may change over time. Our approach provides real-time adaptability to these changes, ensuring consistent performance even in dynamic environments.
- Generalisation across MAS: The focus on behavioural models rather than system-specific features ensures that our approach can be generalised and applied to a wide range of MAS. This modularity allows our method to accommodate different configurations and behaviours commonly found in diverse multi-agent systems, making it a versatile solution for various domains such as traffic management, robotics, and industrial automation.
- Handling nondeterministic and emergent behaviours: Our deep learning-based approach effectively addresses the challenges posed by nondeterministic and emergent behaviours, which are often difficult to detect using traditional methods. By training the model on diverse interaction scenarios, it becomes capable of identifying deviations from expected outcomes, even in unpredictable or complex situations. This ability to manage emergent behaviours is a critical feature in dynamic MAS where unexpected interactions can arise.
- Reduced execution times: Through the use of parallel processing capabilities inherent in deep learning, our approach significantly reduces execution times. Models can be deployed on parallel architectures like GPUs, enabling faster processing and quicker error detection. This reduction in detection time is especially important when scaling to larger MAS, as it allows for real-time responses to detected issues.
- Scalability: Our approach was specifically designed with scalability in mind. As the number of agents in the system increases, deep learning models can handle large-scale MAS without sacrificing accuracy or efficiency. The ability to process large datasets and capture complex relationships between agents ensures that the approach remains robust even in systems with a growing number of agents and dynamic behaviours.
- Reduced reliance on expert knowledge: Unlike traditional methods that require experts to manually define error detection rules, our deep learning-based approach learns directly from the data. This reduces the dependency on specialised expertise, allowing the method to be applied across a variety of domains without requiring deep technical knowledge in each specific field.

By addressing the scalability, adaptability, and dynamic nature of MAS, our approach overcomes the limitations of traditional testing methodologies. It provides a comprehensive, scalable, and adaptive solution that can handle the challenges posed by nondeterministic behaviours, emergent dynamics, and large-scale systems. This makes it a powerful and innovative tool for error detection in MAS, paving the way for more robust testing techniques in the future.

Although deep learning strategies offer promising prospects to achieve automatic, scalable, and dynamic error detection and optimisation in MAS, several significant limitations need to be considered. One critical challenge lies in the limited interpretability of deep learning models, particularly complex neural networks, which often lack transparency. This opacity makes it difficult to understand the decision-making process, hindering the interpretability and explanation of detected errors. Consequently, stakeholders may be reluctant to rely on outcomes that they cannot fully comprehend, which can impact the trustworthiness of the error detection process. Another notable limitation is the dependence on large volumes of high-quality training data, which poses challenges in obtaining diverse and representative datasets for MAS.

Incomplete or biased training data may result in suboptimal performance, hampering the model ability to generalise effectively to real-world scenarios. Furthermore, the proposed approach relies heavily on the accuracy and completeness of the behavioural model used for training. If the model fails to capture all possible behaviours or edge cases, the approach may struggle to detect errors effectively, particularly in complex and unpredictable MAS environments. This dependency highlights the need for careful modeling and validation processes to ensure sufficient coverage of the system behaviours. Consequently, the approach may underperform in comparison toformal verification techniques [53] in scenarios where strict correctness guarantees are required, as deep learning methods inherently introduce probabilistic uncertainty rather than deterministic results. Additionally, scalability challenges may arise when applying deep learning models to real-world MAS with a vast number of agents and intricate interactions, potentially leading to increased computational requirements and longer training times. The adaptability of deep learning models to dynamic environments is another concern, as these models may struggle to

keep pace with evolving system behaviours if not continuously retrained or adapted. High computational costs, sensitivity to variations in input data, dependence on hyperparameter tuning, and ethical concerns related to biases inherited from training data further contribute to the limitations. To overcome these challenges, it is essential to adopt a comprehensive approach that integrates deep learning with other testing and formal methodologies. Ensuring regular updates, continuous monitoring, and a deep understanding of the specific characteristics and requirements of the multi-agent system is vital for addressing these limitations and enhancing the effectiveness of error detection and optimisation strategies. Additionally, incorporating explainable AI (XAI) techniques [54] can significantly improve the interpretability of the model decisions, offering clearer insights into error detection processes and system behaviour.

9. CONCLUSIONS AND PERSPECTIVES

Testing is an important task in the software quality assurance process. Despite the rapid evolution of MAS, testing these systems is still an under-researched area. In fact, only a few proposals for the testing of MAS have been put forward in the literature, although they have led to real progress in the field of MAS testing. However, most of these proposals are related to unit-level and agent-level testing. Moreover, they are very complex and difficult to apply in real cases. In this work, we have presented a novel approach to error detection in MAS using deep learning. This method aims to improve the reliability and performance of MAS by accurately and efficiently identifying errors that can compromise its proper functioning. The proposed approach, supported by the tool we have developed, has been validated on a concrete case study: 'Traffic control system'. The results obtained have enabled us to show that our approach to error detection in MAS using deep learning offers significant advantages in terms of reliability, performance, and automation. In fact, by exploiting the capabilities of deep learning, our approach enables accurate error detection, opening up new possibilities for improving and optimising traditional testing techniques. In the short and medium term, we plan to enhance the accuracy and reliability of error detection and optimisation in MAS by integrating formal and complementary methodologies with deep learning approaches.Additionally, we aim to explore the integration of XAI techniques to improve the interpretability of model decisions, offering clearer insights into error detection processes and system behaviour. To further strengthen error localisation, we plan to analyze error situations detected during the deployment of the deep learning model using advanced anomaly detection algorithms to identify abnormal behaviour or inconsistencies that could indicate the cause of errors.

Building on this, we will develop more precise error localisation methods by combining tracing techniques and in-depth analysis of message exchanges between agents in error situations to determine the specific agent responsible for the error, enabling faster and more targeted problem resolution. We also intend to explore the use of visualisation and graphical representation techniques to map agent states and interactions, providing visual insights into areas where errors propagate or concentrate, thus highlighting critical areas requiring attention. Furthermore, we will integrate machine learning techniques to enhance error localisation by analyzing agent data to detect abnormal patterns, which could pave the way for automatic error correction mechanisms. Finally, we plan to validate the generalizability of the proposed approach by applying it to other systems and domains, demonstrating its scalability, adaptability, and robustness across diverse application contexts.

REFERENCES

[1] S.T. Goonatilleke and B. Hettige, "Past, present, and future trends in multi-agent system technology," *J. Eur. Syst. Autom.*, vol. 55, no. 6, pp. 723–739, 2022, doi: 10.18280/jesa.550604.

[2] S. Bitimanova and A. Shukirova, "Agents and Multi-agent Systems in the Management of Electric Energy Systems," *Manage. Product. Eng. Rev.*, vol. 14, no. 2, pp. 99–110, Jun. 2023, doi: 10.24425/mper.2023.146027.

[3] B. Das, B. Subudhi, and B.B. Pati, "Formation control of underwater vehicles using Multi Agent System," *Arch. Control Sci.*, vol. 30, no. 2, pp. 365–384, Jun. 2020, doi: 10.24425/acs.2020.133503.

[4] P. Qaderi-Baban, M.B. Menhaj, M. Dosaranian-Moghaddam, and A. Fakharian, "Intelligent multi-agent system for DC microgrid energy coordination control," *Bull. Pol. Acad. Sci. Tech. Sci.*, vol. 67, no. 4, 2019, doi: 10.24425/bpasts.2019.130183.

[5] J. Ferber, O. Gutknecht, and F. Michel, "From agents to organizations: An organizational view of multi-agent systems," in *Proc. 4th Int. Workshop Agent-Oriented Softw. Eng.*, 2003, vol. 2935, pp. 214–230.

[6] M. Cossentino, N. Gaud, V. Hilaire, S. Galland, and A. Koukam, "ASPECS: An agent-oriented software process for engineering complex systems—How to design agent societies under a holonic perspective," *Int. J. Autonomous Agents Multi-Agent Syst.*, vol. 2, no. 2, pp. 260–304, 2010.

[7] M. Wooldridge, N.R. Jennings, and D. Kinny, "The GAIA methodology for agent-oriented analysis and design," *Int. J. Autonomous Agents Multi-Agent Syst.*, vol. 3, no. 3, pp. 285–312, 2000.

[8] J. Pavón, J. Gómez-Sanz, and R. Fuentes, "The INGENIAS methodology and tools," in *Agent-Oriented Methodologies*, 2005, pp. 236–276.

[9] M. Hannoun, O. Boissier, J.S. Sichman, and C. Sayettat, "MOISE: An organizational model for multi-agent systems," in *Advances in Artificial Intelligence, IBERAMIA-SBIA*, 2000, pp. 156–165.

[10] B. Putten, V. Dignum, M. Sierhuis, and S. Wolfe, "OperA and Brahms: A symphony?" in *Agent-Oriented Software Engineering IX. AOSE 2008, Lecture Notes in Computer Science*, 2009.

[11] A. Kiran, W. H. Butt, M. W. Anwar, F. Azam, and B. Maqbool, "A Comprehensive Investigation of Modern Test Suite Optimization Trends, Tools and Techniques," *IEEE Access*, vol. 7, pp. 89093–89117, 2019.

[12] S. Zardari *et al.*, "A comprehensive bibliometric assessment on software testing (2016–2021)," *Electronics*, vol. 11, no. 8, p. 1984, 2022, doi: 10.3390/electronics11131984.

[13] C.D. Nguyen, A. Perini, C. Bernon, J. Pavón, and J. Thangarajah, "Testing in multi-agent systems," in *Proc. Int. Workshop Agent-*

Oriented Software Engineering, Budapest, Hungary, 2009, pp. 180–190.

[14] Z. Zhang, J. Thangarajah, and L. Padgham, "Automated unit testing intelligent agents in PDT," in *AAMAS (Demos)*, 2008, pp. 1673–1674.

[15] E.E. Ekinci, A.M. Tiryaki, O. Cetin, and O. Dikenelli, "Goal-oriented agent testing revisited," in *Proc. 9th Int. Workshop Agent-Oriented Software Engineering*, 2008, pp. 85–96.

[16] C.D. Nguyen, A. Perini, and P. Tonella, "Goal-oriented testing for MASs," *Int. J. Agent-Oriented Software Engineering*, vol. 4, no. 1, pp. 79–109, 2010.

[17] D.N. Lam and K.S. Barber, "Debugging agent behaviour in an implemented agent system," in *PROMAS 2004*, R.H. Bordini, M.M. Dastani, J. Dix, and A. El Fallah Seghrouchni, Eds., Springer, Heidelberg, vol. 3346, pp. 104–125, 2005.

[18] C.D. Nguyen, S. Miles, A. Perini, P. Tonella, M. Harman, and M. Luck, "Evolutionary testing of autonomous software agents," in *Proc. 8th Int. Conf. Autonomous Agents and Multiagent Systems (AAMAS 2009)*, IFAAMAS, 2009, pp. 521–528.

[19] C.D. Nguyen, A. Perini, and P. Tonella, "Ontology-based test generation for multi-agent systems," in *Proc. Int. Conf. Autonomous Agents and Multiagent Systems*, 2008.

[20] M. Woodward, "Mutation testing: An evolving technique," in *Colloquium Software Testing for Critical Systems*, 1990.

[21] N.E.H. Dehimi, Z. Tolba, and N. Djabelkhir, "Testing inclusive, exclusive, and parallel interactions in multi-agents system: A new model-based approach," *Int. J. Saf. Secur. Eng.*, vol. 14, no. 4, pp. 1125–1138, 2024.

[22] D. Guassmi, N.E.H. Dehimi, and M. Derdour, "A state of art review on testing open multi-agent systems," in *Novel and Intelligent Digital Systems Conferences*, Athens, Greece, 2023, pp. 262–266, doi: 10.1007/978-3-031-44097-7_28.

[23] S. Boukeloul, N.E.H. Dehimi, and M. Derdour, "A state-of-the-art review of the mutation analysis technique for testing multi-agent systems," in *Novel and Intelligent Digital Systems Conferences*, Athens, Greece, 2023, pp. 230–235, doi: 10.1007/978-3-031-44146-2_23.

[24] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[25] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Trans. Software Eng.*, vol. 38, pp. 1276–1304, 2012.

[26] M. Meiliana, S. Karim, H.L.H.S. Warnars, F.L. Gaol, E. Abdurachman, and B. Soewito, "Software metrics for fault prediction using machine learning approaches: A literature review with PROMISE repository dataset," in *Proc. 2017 IEEE Int. Conf. Cybernetics and Computational Intelligence (CyberneticsCOM 2017)*, 2018.

[27] N. Li, M. Shepperd, and Y. Guo, "A systematic review of unsupervised learning techniques for software defect prediction," *Inf. Softw. Technol.*, vol. 122, p. 106270, 2020, doi: 10.1016/j.infsof.2020.106270.

[28] R. Pan, M. Bagherzadeh, T. A. Ghaleb, and Others, "Test case selection and prioritization using machine learning: A systematic literature review," *Empirical Softw. Eng.*, vol. 27, p. 29, 2022, doi: 10.1007/s10664-022-10025-5.

[29] M. Khatibsyarbini *et al.*, "Trend application of machine learning in test case prioritization: A review on techniques," *IEEE Access*, vol. 9, pp. 166262–166282, 2021, doi: 10.1109/ACCESS.2021.3135508.

[30] I.H. Witten, E. Frank, and M.A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, Elsevier eBooks, 2011. [Online]. Available: https://doi.org/10.1016/c2009-0-19715-5.

[31] V.H. Durelli, R.S. Durelli, S.S. Borges, A.T. Endo, M.M. Eler, D.R. Dias, and M.P. Guimaraes, "Machine learning applied to software testing: A systematic mapping study," *IEEE Trans. Rel.*, vol. 68, pp. 1189–1212, 2019.

[32] N. Jha and R. Popli, "Artificial intelligence for software testing: Perspectives and practices," in *Proc. Fourth Int. Conf. Computational Intelligence and Communication Technologies (CCICT)*, 2021, pp. 377–382, doi: 10.1109/CCICT53244.2021.00075.

[33] C. Ioannides and K.I. Eder, "Coverage-directed test generation automated by machine learning – A review," *ACM Trans. Design Autom. Electron. Syst.*, vol. 17, p. 7, 2012, doi: 10.1145/2071356.2071363.

[34] J.M. Balera and V.A. de Santiago Junior, "A systematic mapping addressing hyper-heuristics within search-based software testing," *Inf. Softw. Technol.*, vol. 114, pp. 176–189, 2019.

[35] Z. Zhou, M. Sunkara, Y. Lei, and A. Ramesh, "Machine learning for software testing: A survey," *arXiv:1906.10742*, 2019. [Online]. Available: https://arxiv.org/abs/1906.10742.

[36] M.M. Alam, S. Ali, A. Khan, M. Hamayun, and K.Z. Khan, "Machine learning for improving API testing," *arXiv:2207.13143*, 2018. [Online]. Available: https://arxiv.org/abs/2207.13143.

[37] D. Guassmi, N.E.H. Dehimi, M. Derdour, and A. Kouzou, "Using machine learning techniques for multi-agent systems testing," in *Artificial Intelligence and Its Practical Applications in the Digital Economy (I2COMSAPP 2024), Lecture Notes in Networks and Systems*, 2024, vol. 861, pp. 230–235, doi: 10.1007/978-3-031-71426-9_16

[38] S.U. Rehman and A. Nadeem, "An approach to model-based testing of multiagent systems," *Sci. World J.*, vol. 2015, p. 925206, 2015, doi: 10.1155/2015/925206.

[39] N.E.H. Dehimi, F. Mokhati, and M. Badri, "Testing HMAS-based applications: An ASPECS-based approach," *Eng. Appl. Artif. Intell.*, vol. 46, pp. 25–33, 2015, doi: 10.1016/j.engappai.2015.09.013.

[40] N.A. Bakar and A. Selamat, "Agent systems verification: Systematic literature review and mapping," *Appl. Intell.*, vol. 48, no. 5, pp. 1251–1274, 2018, doi: 10.1007/s10489-017-1112-z.

[41] C. Barnier, O.-E.-K. Aktouf, A. Mercier, and J.P. Jamont, "Toward an embedded multi-agent system methodology and positioning on testing," in *Proc. 2017 IEEE Int. Symp. Software Reliability Engineering Workshops (ISSREW)*, 2017, pp. 239–244.

[42] M. Winikoff, "BDI agent testability revisited," *Autonomous Agents and Multi-Agent Systems*, vol. 31, no. 6, pp. 1094–1132, 2017, doi: 10.1007/s10458-016-9356-2.

[43] E.M.N. Gonçalves, R. A. Machado, B. C. Rodrigues, and D. Adamatti, "CPN4M: Testing multi-agent systems under organizational model Moise+ using colored Petri nets," *Appl. Sci.*, vol. 12, no. 12, p. 5857, 2022, doi: 10.3390/app12125857.

[44] M.S.U. Rehman, A. Nadeem, and M.A. Sindhu, "Towards automated testing of multi-agent systems using Prometheus design models," *Int. Arab J. Inf. Technol.*, vol. 16, pp. 54–65, 2019. [Online]. Available: https://dblp.uni-trier.de/db/journals/iajit/iajit16.html#RehmanNS19.

[45] Z. Huang, R. Alexander, and J. Clark, "Mutation testing for Jason agents," in *Proc. Int. Workshop Eng. Multi-Agent Syst. (EMAS 2014)*, Paris, France, 2014.

[46] N.E.H. Dehimi, A.H. Benkhalef, and Z. Tolba, "A novel mutation analysis-based approach for testing parallel behavioural scenarios in multi-agent systems," *Electronics*, vol. 11, no. 22, p. 3642, 2022, doi: 10.3390/electronics11223642.

[47] N.E.H. Dehimi, S. Boukelloul, and D. Guassmi, "Towards a new dynamic model-based testing approach for multi-agent systems," in *Proc. 2022 4th Int. Conf. Pattern Analysis and Intelligent Systems (PAIS)*, IEEE, 2022, pp. 1–6, doi: 10.1007/978-3-031-44146-2_23.

[48] M. Schuster and K.K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, 1997, doi: 10.1109/78.650093.

[49] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org/.

[50] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[51] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, 2012.

[52] A. Krizhevsky, I. Sutskever, and G.E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[53] H.T. Nguyen, M.W. Berry, and J.D. Kiffe, "Numerical Methods for Partial Differential Equations, 2nd ed., ser. Texts," in *Computational Science and Engineering*. Cham: Springer, 2019, doi: 10.1007/978-3-030-38800-3.

[54] N. El Houda Dehimi and Z. Tolba, "Attention Mechanisms in Deep Learning: Towards Explainable Artificial Intelligence," in *6th International Conference on Pattern Analysis and Intelligent Systems (PAIS)*, El Qued, Algeria, 2024, pp. 1-7, doi: 10.1109/PAIS62114.2024.10541203.