Intrusion Detection System, a Decentralized Approach Using Multi-agent Systems and Machine Learning

Nathan Cesa Nery de Castro - 57011

Thesis presented to the School of Technology and Management in the scope of the Master in Electrotechnical and Computer Engineering, in the scope of the double diploma programme with the Federal University of Technology - Paraná.

Supervisors:

Prof. Dr. Paulo Jorge Pinto Leitão

Prof. Dr. Wesley Angelino de Souza

Bragança

2024 - 2025

Intrusion Detection System, a Decentralized Approach Using Multi-agent Systems and Machine Learning

Nathan Cesa Nery de Castro - 57011

Thesis presented to the School of Technology and Management in the scope of the Master in Electrotechnical and Computer Engineering, in the scope of the double diploma programme with the Federal University of Technology - Paraná.

Supervisors:

Prof. Dr. Paulo Jorge Pinto Leitão

Prof. Dr. Wesley Angelino de Souza

Bragança

2024 - 2025

Acknowledgment

I would like to thank everyone who contributed to the completion of this work and to my academic journey.

Gratitude to the the educational institutions UTFPR (Federal Technological University of Paraná) and IPB (Polytechnic Institute of Bragança) for the opportunity to participate in the dual degree program.

I would also like to thank my supervisors and course colleagues for their support and exchanged experiences.

Finally, my deepest gratitude to all friends, relatives and family. Especially my parents, for all their support and encouragement to complete this journey.

Abstract

The increasing adoption of cloud computing and digital environments has made network infrastructures more complex and, at the same time, more vulnerable to cyberattacks. Traditional centralization of security mechanisms is becoming increasingly challenging, since data and applications are dispersed across multiple environments. Faced with this scenario, new decentralized approaches are emerging to try to solve this problem. This work presents a distributed approach for an intrusion detection system, through a multi-agent system and machine learning, and how this can be an advantageous tool compared to a centralized and individual system.

Through a multi-agent system, it is possible to solve complex problems collaboratively, using the interaction among multiple autonomous entities that work in a coordinated manner. Each agent within the system has specific objectives, the ability to perceive the environment and the ability to make decisions independently or in conjunction with other agents. This approach is especially useful in dynamic and distributed scenarios, such as logistics, process optimization and large-scale simulations, etc. In addition, multi-agent systems favor the adaptability, scalability and robustness of solutions, allowing an efficient response to challenges that require decentralization and cooperation. Combining this with machine learning techniques, it is possible to add new features to an intrusion detection system, making it intelligent and decentralized, in order to increase the scope of detections, reduce data processing time and reaction.

Among the results obtained, comparing a collaborative system with an individual system, accuracy improved by 1.13%, precision by 1.31%, recall (sensitivity) by 0.04% and F1-score by 0.69%. Having an average action time of 6.33 seconds.

Keywords: Cybersecurity, intrusion detection system, machine learning, multi-agent systems.

Resumo

A crescente adoção da computação em nuvem e dos ambientes digitais têm tornado as infraestruturas de rede mais complexas, e ao mesmo tempo mais vulneráveis a ataques cibernéticos. A centralização tradicional dos mecanismos de segurança está se tornando cada vez mais desafiadora, uma vez que os dados e as aplicações estão dispersos em diversos ambientes. Diante desse cenário novas abordagens descentralizadas vêm surgindo para tentarem sanar este problema. Este trabalho apresenta uma abordagem distribuída para um sistema de detecção de intrusão, por meio de um sistema multiagente e aprendizado de máquina, e como esta pode ser uma ferramenta vantajosa em comparação a um sistema centralizado e individual.

Por meio de um sistema multiagente, é possível resolver problemas complexos de forma colaborativa, utilizando a interação entre múltiplas entidades autônomas que trabalham de maneira coordenada. Cada agente dentro do sistema possui objetivos específicos, capacidade de percepção do ambiente e habilidade para tomar decisões independentes ou em conjunto com outros agentes. Essa abordagem é especialmente útil em cenários dinâmicos e distribuídos, como logística, otimização de processos, simulações em larga escala e etc. Além disso, os sistemas multiagentes favorecem a adaptabilidade, a escalabilidade e a robustez das soluções, permitindo uma resposta eficiente a desafios que exigem descentralização e cooperação. Combinando isso com técnicas de machine learning é possível adicionar novas características a um sistemas de detecção de intrusão, tornando-o inteligente e descentralizado, a fim de aumentar o escopo das detecções, reduzir tempo de processamento de dados e reação.

Dentre os resultados obtidos, comparando um sistema colaborativo com um sistema

individual, a acurácia melhorou em 1.13%, a precisão em 1.31%, o recall (sensibilidade) em 0.04% e o F1-score em 0.69%. Tendo um tempo médio de ação de 6.33 segundos.

Palavras-chave: Cybersegurança, sistema de deteção de intrusão, aprendizado de máquina, sistemas multi-agentes.

Contents

List of Figures

Chapter 1

Introduction

This work is set in the context of cybersecurity, focusing on concepts such as “Intrusion Detection System”, “Multi-Agent System” and “Machine Learning”. The combination of these technologies enables the development of more intelligent and adaptable security systems that can contribute to security in various sectors.

1.1 Motivation and Framework

Cybersecurity has become an increasingly crucial topic in our daily lives as through increasing global connectivity and digitalization of services in various spheres of our lives, cyber attacks have become more frequent and sophisticated, as reported by Mijwil et al. [1]. These attacks can be carried out using numerous tactics in order to exploit gaps in computer systems, electronic networks and devices connected to the internet, with the intention of causing damage or gaining access to confidential information, and can affect both ordinary citizens and different types of industries and institutions, being able to interrupt services and business processes [1].

According to Statista [2], the global damage caused by cybercrimes has been growing every year, with values reaching trillions of dollars. Based on the estimate, it is expected that in 2028, the loss will reach 13.82 trillion dollars, an increase of 1507% in ten years.

With this scenario in mind, as organizations face an increasing volume and complexity

of cyber threats, new strategies are being explored, such as distributed approaches to systems monitoring and security [3],[4]. Driven by the expansion of digital environments, adoption of cloud services, increasing proliferation of Internet of Things (IoT) devices and the increasing complexity of networks, thus generating the need for agile and resilient defense mechanisms.

One of the important components for network and system security is an Intrusion Detection System (IDS), which continuously monitors suspicious activities on networks and devices, seeking to detect any suspicious activity or anomalies, as per by Jose et al. [5]. This makes it possible to prevent intrusions from escalating, protecting information assets, network infrastructure, and sensitive data. In addition, an IDS can help security teams identify vulnerabilities in an organization's systems. With intrusions recorded and analyzed, it is possible to detect patterns that point to specific flaws in their systems, enabling corrections and updates. Traditionally, centralized systems, such as IDS, concentrate the monitoring and analysis of suspicious activities in a single point, seeking to identify and mitigate vulnerabilities reactively. However, this model can become a bottleneck in the face of increasingly large and dynamic networks, where a single point of failure can compromise the entire infrastructure. Consequently, new approaches have emerged in search of mitigating these possible points of failure effectively through the implementation of decentralized architectures, which distribute the monitoring and response load among multiple points.

One way to have a decentralized and collaborative system is through multi-agent systems (MAS). MAS are computer systems made up of entities called interdependent agents that collaborate, compete or act autonomously to achieve individual or collective goals, as stated by Dorri et al. [6]. Each agent can have specific skills, for example, being equipped with a machine learning algorithm to detect anomalous activity in the network, being able to perceive its environment, detect possible suspicious activity, make decisions, and carry out actions according to its programming. Cooperation and coordination among agents make MAS an excellent tool for solving distributed problems that are difficult or limited to be dealt with by a centralized system.

1.2 Objectives

The main objective of this work is to develop a decentralized intrusion detection system using MAS and ML technologies through a simulation focused on observing how a decentralized system can be advantageous in relation to an individual one.

For this, several specific objectives were defined:

- Analyze and understand the experimental case study, a distributed approach for an IDS that works collaboratively using MAS and machine learning
- Define the ML implementation and define the algorithms based on the results of metrics such as accuracy, precision, recall, and f-measure.
- Implement ways to continuously improve the system, a scoring mechanism, and algorithm exchange.
- Define how to measure system performance and which metrics will be analyzed to compare individual and collaborative performance.

1.3 Document Structure

This document is organized into 6 chapters, beginning with a contextualization of the work and the main technologies used for this simulation.

Chapter 2 presents the theoretical framework used to develop the work.

Chapter 3 shows the description of the proposed system.

Chapter 4 demonstrates how each part of the system was made, from data processing to the solution with MAS.

Chapter 5 presents the results found and discussions from the training of the models as well as the simulations made based on the variations of confidence levels.

Chapter 6 presents the conclusion and future work for possible improvements to the system.

Chapter 2

Motivation and Framework

This chapter aims to take the reader through a literature review, covering concepts and fundamentals within the scope of Cybersecurity, intelligent and distributed systems, particularly IDS based on MAS technology. First, the concepts of cybersecurity and IDS will be explained. Next are machine learning concepts and algorithms. Finally, the concepts of MAS, followed by a brief explanation of the communication protocols used.

2.1 Cybersecurity and Cloud Computing

Cybersecurity refers to the practice of protecting systems, networks, and data using a set of technologies and processes, as stated by Sarker el al. [7]. Various aspects are linked to cybersecurity in such a way that it ends up being a collection or arrangement of resources, such as infrastructure, personnel, structures and processes with the aim of protecting networks and systems against events that could cause damage, whether in financial, data or reputational terms, preserving the reliability and security of digital operations, as per by Schiliro [8].

According to Boss el al. [9] cloud computing can be defined as a set of services that can be storage, management and data processing on remote servers offering computing resources on demand. Its main objective is to enable the sharing and collaboration of resources on a large scale, facilitating the execution of intensive computing tasks, bringing

interoperability and load balancing in the distribution of tasks, as argued by Beri and Behal [10] .

In today's world, where cloud computing has become essential with the growing dependence on remote resources, the need for cybersecurity practices becomes imperative to prevent and mitigate potential threats and vulnerabilities. In this way, cloud computing and cybersecurity complement each other, with the former operating efficiently and scalably and the latter ensuring that this infrastructure operates safely and securely.

2.2 Intrusion Detection System

In accordance to Jose et al. [5], an Intrusion Detection System (IDS) is a critical cybersecurity component designed to monitor and analyze network traffic or system activities for suspicious or malicious activity network traffic or system activities in order to identify suspicious or malicious activity.

Unlike an Intrusion Prevention System (IPS), which has both detection and prevention capabilities, blocking possible attacks on the system, an IDS system has limited detection functions only, and may also contain alert generation for malicious activities.

Categories of IDS

An IDS system can be categorized based on the type of detection system (Data Collection Technique) and also on how the intrusion is detected (Data analysis Technique), as shown in Figure 2.1

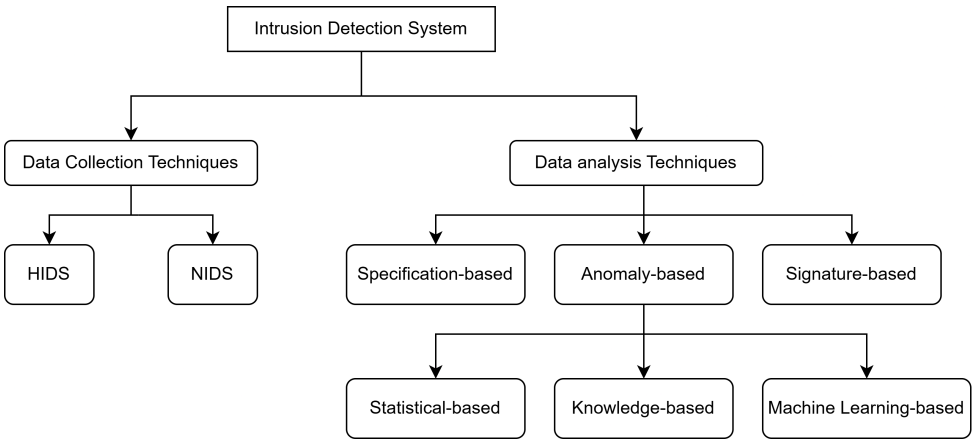

Figure 2.1: Classification of IDS (adapted from Jose et al. [5]).

Data Collection Techniques

An intrusion detection system can be implemented as software or hardware and aims to automate the detection process using monitoring techniques. It aims to identify possible changes in patterns or activities previously defined as malicious and relies on possible data collection methods, as per by De Geus and Nakamura [11]:

- Host Intrusion Detection System (HIDS): They run on the hosts and monitor the system using information from the log files on the machine, allowing access and changes to system files considered crucial to be monitored.
- Network Intrusion Detection System (NIDS): They run on various network devices and monitor traffic within the network segment where they are deployed. Detection takes place by capturing and analyzing the headers and contents of the packets, which are compared with previously known attack signature patterns.
- Hybrids: Systems that monitor system information as well as network packets.

Data analysis Techniques

As stated Adnan et al. [12], an IDS can also be classified according to the type of intrusion identification technique. The main techniques are signature-based and anomaly-based. There are also specification-based techniques and hybrid techniques.

- Signature-based: It uses a stored collection of signatures from previous attacks, such as specific patterns, known malicious instruction sequences, byte sequences in network traffic, and known system vulnerabilities. Each type of threat is associated with a type of malicious signature, such as failed logins, failed application execution attempts, and data packet characteristics [13].

  The main advantage of this system is that it is easy to develop and understand, especially when you know the typical behavior of network traffic and system activity. However, the main limitations of this approach are the need for constant updates and maintenance of the signature collection, as well as the possible inability to identify new and unique attacks. While these systems are effective against attacks with static behavior patterns, they have difficulty dealing with behavioral characteristics that change dynamically.

- Specification-based: This technique uses a set of rules with specifications and restrictions that describe the operation of a given system, and if any type of non-compliance with any specification or rule occurs, the program generates an alert.

- Anomaly-based: It uses a baseline or pattern of normal system activity to identify possible intrusions in progress. Any deviation from this baseline or pattern triggers an alarm.

  This type of technique is designed to be flexible, but one of the main disadvantages of this type of system is the complexity of defining rules. All the protocols analyzed must be clearly defined, implemented, and tested to ensure accuracy, as this type of approach can generate false alarms, thus requiring constant updates of activity patterns. Within this category we have the following subcategories:

  Statistical-based anomaly: It uses the combination of statistical characteristics already captured from the network with a stochastic model-based on the normal behavior of the network. When an attack occurs, the system can identify it by the discrepancy between the two statistical patterns.

Knowledge-based anomaly: It is defined by a set of predefined rules and behaviors to detect deviations in a system's behavior and can also use other strategies, such as fuzzy logic, with a subset of rules activated based on input values, either by a set of heuristics or based on Unified Modeling Language descriptions.

Machine learning-based anomaly: This approach, instead of relying on predefined signatures or fixed rules, uses machine learning algorithms to identify patterns of behavior and detect malicious activity or intrusion.

This work focused on the use of an anomaly-based NIDS using machine learning techniques.

2.3 Machine Learning

Machine Learning (ML) is a sub-area of artificial intelligence, which seeks to develop algorithms and techniques that allow machines to learn something from data, either to make predictions, identify patterns, or make decisions from them, as per by Bini [14]. As stated by Ayodele [15], the Machine Learning taxonomy is divided as follows:

- Supervised learning: where the algorithm generates a function that maps inputs to desired outputs. One standard formulation of the supervised learning task is the classification problem: the learner is required to learn (to approximate the behavior of) a function that maps a vector into one of several classes by looking at several input-output examples of the function.
- Unsupervised learning: which models a set of inputs: labeled examples are not available.
- Semi-supervised learning: which combines both labeled and unlabeled examples to generate an appropriate function or classifier.
- Reinforcement learning: where the algorithm learns a policy of how to act given an observation of the world. Every action has some impact in the environment, and the

environment provides feedback that guides the learning algorithm.

- Transduction: similar to supervised learning, but does not explicitly construct a function: instead, tries to predict new outputs based on training inputs, training outputs, and new inputs.

- Learning to learn: where the algorithm learns its own inductive bias based on previous experience.

This work is limited to supervised learning. The choice of algorithms applied in this study was based on the diversity of approaches and the ability of each method to capture different nuances present in the data. In this way, the combination was chosen to make it possible to compare linear and non-linear techniques, as well as methods based on instances, distance, and margin.

2.3.1 Decision Tree Classifier

This algorithm works with an inductive inference method which uses discrete-valued target functions as an approximation method, so the final function is represented by a decision tree, as outlined by Mitchell [16].

A decision tree model (DT) has a root node connected to several data attribute nodes that determine the path to each subsequent node. Decisions are made by comparing previous data and are represented in the leaf nodes. One of the advantages of this algorithm is that it can automatically ignore irrelevant and redundant features. Figure 2.2 shows how a decision tree works.

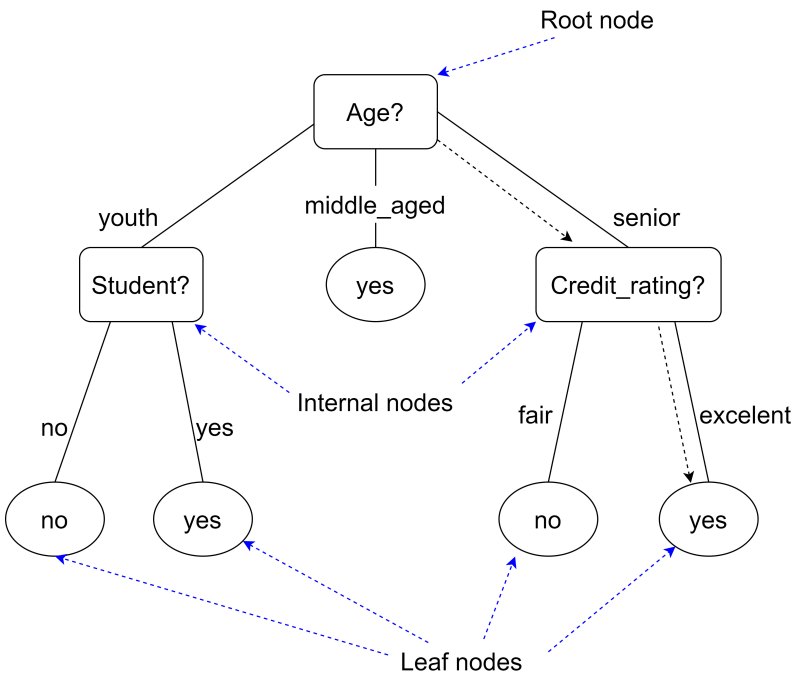

Figure 2.2: Exemple of Decision Tree (adapted from Li [17]).

As seen in the figure above, a decision tree is similar to a flowchart, where the root node is the attribute, the internal nodes represent the tests on an attribute , and the leaves represent the class labels, i.e., the decision. This example in the figure is a decision tree for the case of buying a computer, i.e., it predicts whether a customer is likely to make the purchase.

The process of building the tree is based on the division of attributes so that the algorithm looks for which characteristic of the data is most relevant to separate, i.e., it is based on a heuristic.

As mentioned by Rokach and Maimon [18], there are numerous heuristics with different functions for pattern detection, such as the information gain heuristic, which uses the concept of entropy to choose the variable that maximizes the information gain, or the Gini index, which uses a statistical dispersion index to measure the probability of incorrect data classification. Among the various DT algorithms, the most popular are: ID3, C4.5, and CART [19].

2.3.2 K-Nearest-Neighbor Classifier

The k-Nearest-Neighbors (kNN) seek to classify a sample based on the labels of its nearest neighbors. This approach requires 3 key elements, a set of labeled data, the distance between the samples (labeled data), and the value of K. To classify unlabeled data, the distance between it and the nearest labeled data is calculated, as per by Wu et al. [20].

In a sample where the neighborhood belongs to the same class, that sample has a high probability of belonging to that class. In other words, this type of classification is associated with very close neighbors. The way to adjust this classification is to modify the K parameter, which influences the performance of the models, so that decreasing the K increases the risk of overfitting while increasing the K makes the model simpler and more general, which can reduce its ability to capture complex patterns in the data, weakening its fitting capacity, as described by Liu and Lang [21].

This algorithm is suitable for application to data sets with noise, is effective in dealing with non-linear patterns and has a fast training time. Figure 2.3 illustrates how KNN works.

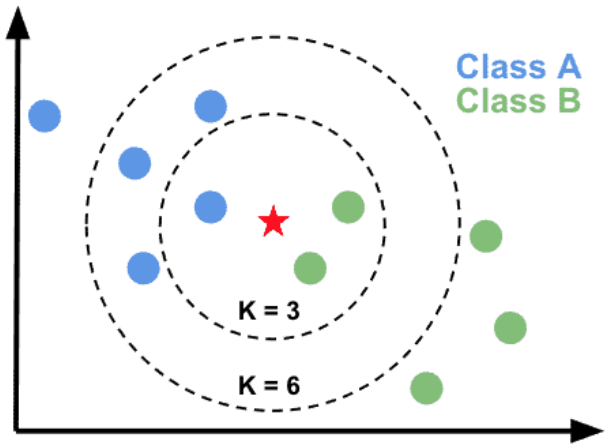

Figure 2.3: Example of KNN (Chouinard [22]).

2.3.3 Logistic Regression

Logistic Regression (LR) is a linear algorithm model that calculates the probability of different classes using the parametric logistic distribution, as shown in Formula (2.1). This model tries to estimate the chance of a given categorical outcome depending on one or more predictors, which can be continuous or categorical.

$$P(Y = 1 \mid X) = \frac{e^{a+bX}}{1 + e^{a+bX}} \tag{2.1}$$

Where:

- P: Probability of an event occurring (Probability of variable Y belonging to class 1, given the vector of characteristics X).
- e: Euler's constant (approximately 2.71828).
- a: Intercept of the regression (the value of P when X is equal to zero).
- b: Regression coefficient, indicates the effect of variation in variable X on probability P.
- X: Explanatory or predictor variable.

In short, the value of P varies between 0 and 1. When the value of a + bX is very negative, P will be close to 0 (low probability). Otherwise, if a + bX is positive and high, the P value will be close to 1 (high probability), as per by Brannick [23]. The sample x is classified into the maximum probability class. An LR model is easy to construct, and model training is efficient. On the other hand, LR does not behave well with non-linear data, which makes its application limited [21].

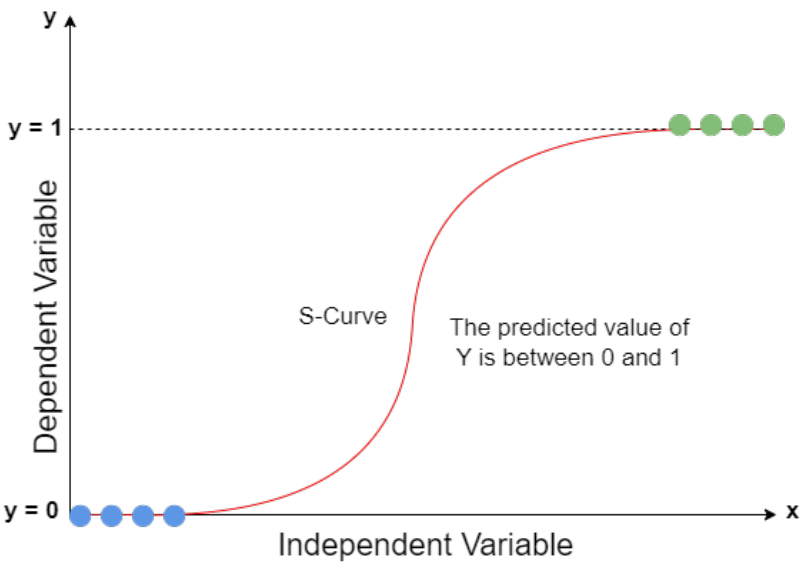

Figure 2.4: Example of LR (adapted from Voxco [24]).

Logistic Regression relies on Maximum Likelihood estimation, which posits that the observed data should be maximally probable, as demonstrated in Figure 2.4. We pass the weighted sum of inputs and apply an activation function capable of mapping values between 0 and 1. This activation function is commonly referred to as the sigmoid function, producing a curve known as the sigmoid curve or S-curve.

2.3.4 Support Vector Machine

The strategy of the Support Vector Machine (SVM) is to find a hyperplane that can best separate the samples into different classes, as outlined by Liu and Lang [21]. In the case of a binary classification, the goal is to draw a line (in the case of two dimensions) or hyperplane (in higher dimensions) that maximizes the distance between the closest points of the two classes, called support vectors, this distance is called a margin. Figure 2.5 shows a simple representation of an SVM.

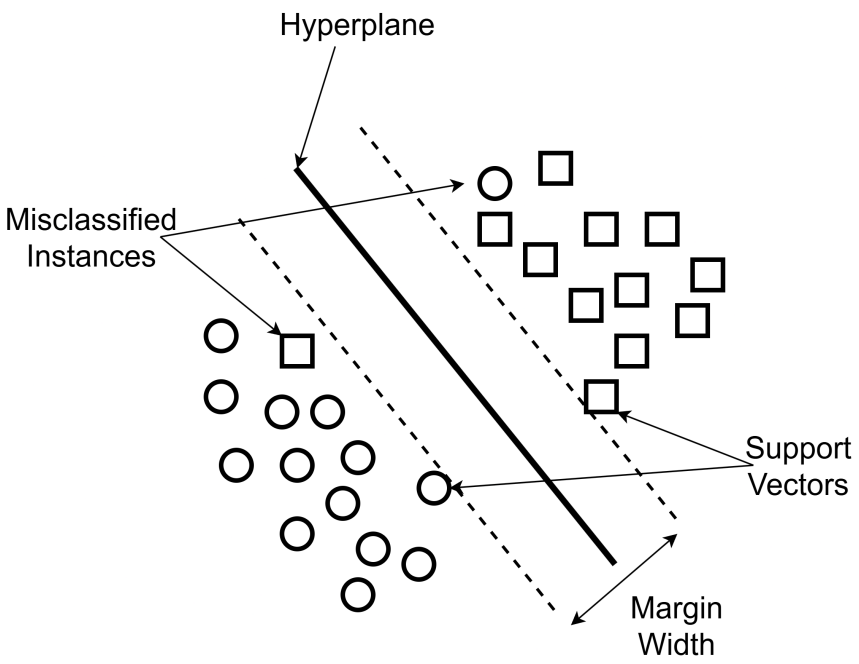

Figure 2.5: Linear SVM example (adapted from Mountrakis et al. [25]).

To classify a new piece of data, the data point is projected onto the same space and assigned to the class on the side of the hyperplane it is on. If the data is not linearly separable, the SVM can use kernel functions to map the data to a higher-dimensional space where linear separation becomes possible, as per by Somvanshi et al. [26]. The SVM has some important features that help to adjust it according to the application: Gamma, Kernel, and C (regularization).

The regularization parameter C adjusts the relationship between the maximum margin and the classification errors found in training. When C is small, the model allows for more classification errors because the margin is large, and when C is large, the opposite occurs: If an attempt is made to classify all the samples correctly, the margin becomes small, which can result in overfitting. The Kernel is a function used to transform the input data into a higher-dimensional feature space, thus making it possible to solve non-linear problems. The Kernels are: Linear, Polynomial, Radial Basis Function (RBF), and Sigmoid, as mentioned by Hastie et al. [27].

At last gamma, which is related to non-linear and controls the range of samples that

will be considered when calculating the decision boundary. By increasing the gamma value, the model will only consider data points very close to the decision boundary, which can lead to overfitting, and by decreasing the gamma, the decision boundary region increases so that more distant data points are also considered, which can lead to underfitting as asserted by Al-Mejibli et al. [28].

2.3.5 Artificial Neural Networks

According to Dastres and Soori [29], ANNs can be characterized as structures made up of processing units, connected artificial neurons, which are capable of performing extensive complex calculations in parallel, for data processing and knowledge representation.

Despite being based on biological concepts, ANNs are simplified representations of complex Neural Networks present in living organisms. Their main objective is to use knowledge about how biological networks process information in order to use this to solve complex problems, and not to exactly replicate their biological functioning. ANNs have some advantages, such as being able to deal with non-linear data, processing multiple pieces of information in parallel, being robust to failures and noise, and dealing with imprecise or vague information, as per by Basheer and Hajmeer [30].

Figure 2.6 depicts the parallels between a biological neural network and an artificial counterpart. The depicted neuron is the perceptron, serving as the fundamental unit of a neural network. Here, x denotes the input, w represents the weight of each connection, and σ signifies an activation function of ε. The activation functions can vary, encompassing simple threshold-based functions, sigmoid, gaussian, and sinusoidal functions.

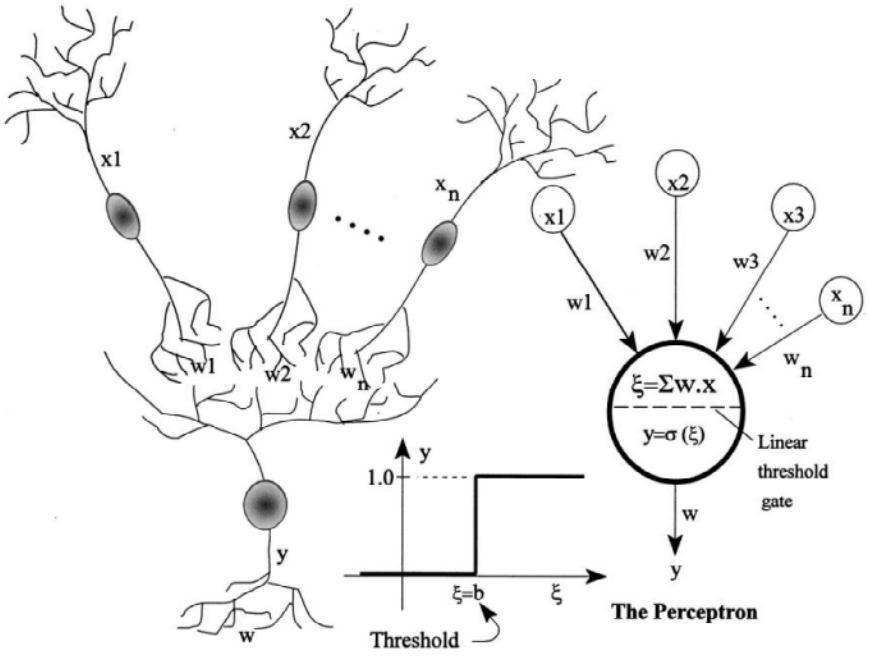

Figure 2.6: Correlation of a biological and an artificial neuron (adapted from Basheer and Hajmeer [30]).

The comparison between an artificial neuron and a biological neuron lies in the representation of connections: the inter-node connections mimic axons and dendrites, the connection weights correspond to synapses, and the threshold mirrors the activity within the soma.

There are various types of neural networks, but the most commonly used are Multilayer Perception (MLP), Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN). In this work, the focus will be on the use of an MLP.

MLP is a type of ANN widely used for classification tasks. It works by mapping input data to a set of outputs through three layers, an input layer, an output layer, and one or more hidden (processing) layers, as stated by Shanbehzade et al. [31]. Each layer is made up of neurons connected to all the neurons in the other layers forming a directed graph, with the exception of the input nodes, where each node is a neuron (processing element) with a non-linear activation function. Figure 2.7 shows the architecture of an MLP.

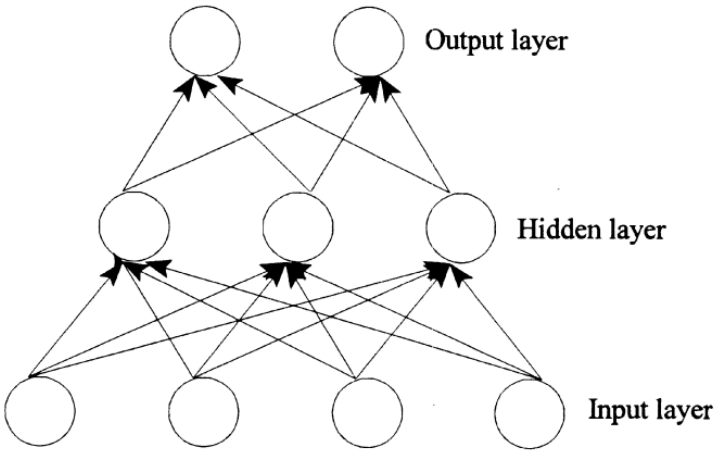

Figure 2.7: Simple MLP architecture (adapted from Basheer and Hajmeer [30]).

The structure of a neural network is defined by hyperparameters such as the number of hidden layers, the dimensionality of each layer, and the function of the chosen activation. By varying the hyperparameters it is possible to test different architectures and evaluate their performance for a given type of application problem in order to find an architecture that optimizes performance metrics, as outlined by AlDahoul et al. [32].

2.3.6 Random Forest

Random Forest (RF) is a supervised algorithm that works non-linearly and can be used for both classification and regression. It is built by creating multiple DT to train the model. The trees work together so that the combination of their decisions generates more accurate results, as per by Thangaraj et al. [33]. The advantage of RF comes from its randomness because when it builds decision trees with random samples and features, the algorithm creates a diversity of models, producing a more robust and generalizable result, reducing the risk of overfitting. Figure 2.8 shows the architecture of an RF.

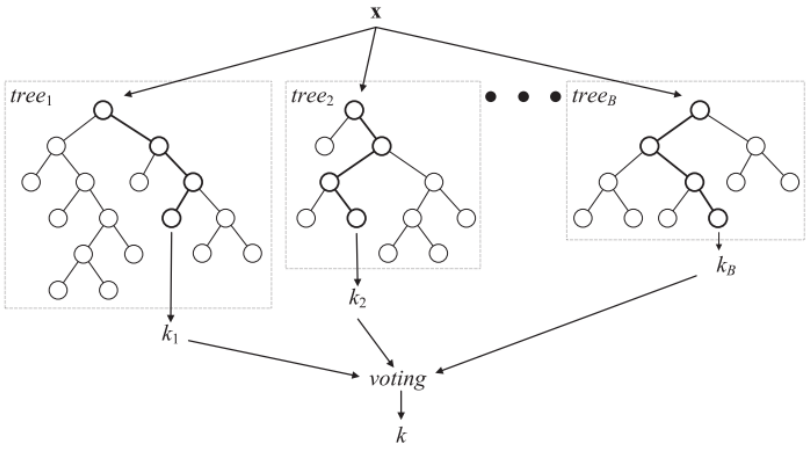

Figure 2.8: A general architecture of a Random Forest (Verikas et al. [34]).

2.3.7 Models Evaluation

Evaluation metrics are fundamental in the process of developing and validating an ML model. Using them, it is possible to quantify and evaluate the performance of a model based on its relation with training and test data. By analyzing these metrics, it is feasible to understand how the model generalizes and behaves with previously unseen data.

There are various metrics for evaluating an ML algorithm. Each metric offers a different perspective on the performance of the generated model, such as accuracy, sensitivity, or the ability to generalize over new data, as stated by Costa et al. [35].

When it comes to a scenario where there is only one dataset, it is necessary to use sampling techniques, defining training, and test subsets to obtain more accurate predictive performance estimates. There are various methods, such as Resubstitution, Bootstrap, Hold-out, k-fold cross-validation, Random Subsampling, and Leave-One-Out Cross-Validation, as mentioned by Reich and Barai [36]. However, this work will use the hold-out method wich randomly divided into disjoint training and testing sets. It is common to select 2/3 of the set for training and the remaining 1/3 for testing, although there are no theoretical foundations for these values. Another option, for example, is to partition the dataset into 4/5 of the set for training and the remaining 1/5 for testing.

Evaluating the performance of a supervised ML algorithm can be done in various

ways. Generally, the evaluation measures in classification problems are defined from a confusion matrix [35]. Also known as the error or contingency matrix, it calculates each class label's number of occurrences for both states (true classified and false classified). In a classification context, there are always binary output values (1, 0) or (true, false) [37]. The Table 2.1 demonstrates a confusion matrix.

Table 2.1: Confusion matrix.

		Predicted Class	
		Positive	**Negative**
Actual Class	**Positive**	TP	FN
	Negative	FP	TN

The concepts of FP, FN, TP, and TN can be described as:

- False positives (FP): examples predicted as positive, which are from the negative class.
- False negatives (FN): examples predicted as negative, whose true class is positive.
- True positives (TP): examples correctly predicted as pertaining to the positive class.
- True negatives (TN): examples correctly predicted as belonging to the negative class.

This makes it possible to calculate some popular performance metrics, such as accuracy, precision, recall, and f1-score.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.2}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

$$F - measure = \frac{(1 + \beta^2) * \text{precision} * \text{recall}}{\beta^2 * \text{precision} + \text{recall}} \tag{2.5}$$

Accuracy is a performance metric reflecting the ratio of total correct predictions over the total records of the dataset used for the evaluation, however, it does not distinguish between the number of correct labels of different classes, different from the precision, recall, and F-measure (F1-score) metrics, which distinguish the correct classification of labels within different classes [38], [39].

Precision is the fraction of all correct predictions for positive class (TP) among all predicted instances answering as positive class. Recall, on the other hand, is a metric that is mainly focused on the ratio of all correct predictions for positive class (TP) among all positive actual classes (all positive classes in the population or dataset). In the context of the classification problem with an imbalanced data set, it is required to use the F-measure metric to tackle the accuracy metric's limitation. The F1-score metric represents the Harmonic Mean of Precision and Recall [39]. In addition, the F1-score achieves an equal weighting of precision and recall at $\beta = 1$. Values of β greater than 1 prioritize precision, while values less than 1 favor recall.

2.4 Multi-Agent Systems

MAS consist of multiple entities called agents that interact in a shared environment aiming to achieve some individual or collective system objective, as per by Theodoropoulos et al. [40]. Stemming from symbolic artificial intelligence, MAS is characterized by the ability of agents to understand their environment and make decisions, either by reacting to events or anticipating future states, based on their individual information or that of other agents, as outlined by Cardoso and Ferrando [41].

In the literature, there are various definitions of an agent since it can be a real or

virtual (abstract) entity. However, these definitions end up differing according to the application to which the agent is subjected. Therefore, an agent is like an autonomous and flexible entity that has the ability to perceive its environment by means of sensors connected to it and to interact with the environment by means of actuators in order to perform a task or achieve a goal, acting as designed.

According to Balaji and Srinivasan [42], there are certain main properties that help illustrate what characteristics an agent should have:

- Situatedness: This refers to the interaction of an agent with the environment through the use of sensors and the resultant actions of the actuators. Environment in which an agent is present is an integral part of its design. All of the inputs are received directly as a consequence of the agents interactions with its environment. The agent's directly act upon the environment through the actuators and do not serve merely as a meta-level advisor.

- Autonomy: This can be defined as the ability of an agent to choose its actions independently without external intervention by other agents in the network (case of multi-agent systems) or human interference. This attribute protects the internal states of the agent from external influence. It isolates the agent from instability caused by external disturbances.

- Inferential capability: The ability of an agent to work on abstract goal specifications, such as deducing an observation by generalizing the information. This could be done by utilizing relevant contents of available information.

- Responsiveness: The ability to perceive the condition of the environment and respond to it in a timely fashion to take account of any changes in the environment. This latter property is of critical importance in real-time applications.

- Pro-activeness: Agent must exhibit a good response to opportunistic behaviour. This is in order to enhance actions that are goal-directed rather than just being

responsive to a specific change in environment. It must have the ability to adapt to any changes in the dynamic environment.

- Social behaviour: Even though the agent's decision must be free from external intervention, it must still be able to interact with the external sources when the need arises to achieve a specific goal. It must also be able to share this knowledge and help other agents (MAS) solve a specific problem. That is, agents must be able to learn from the experience of other communicating entities, which may be humans, other agents in the network, or statistical controllers.

Therefore, in simplified form, MAS is a computational structure consisting of several intelligent, distributed agents. As shown in Figure 2.9, these agents interact in a specific environment in order to achieve their individual goals. To achieve these goals, they need to be equipped with sufficient knowledge and skills. In addition, they must have the autonomy to make independent decisions, allowing them to react to changes in their environment.

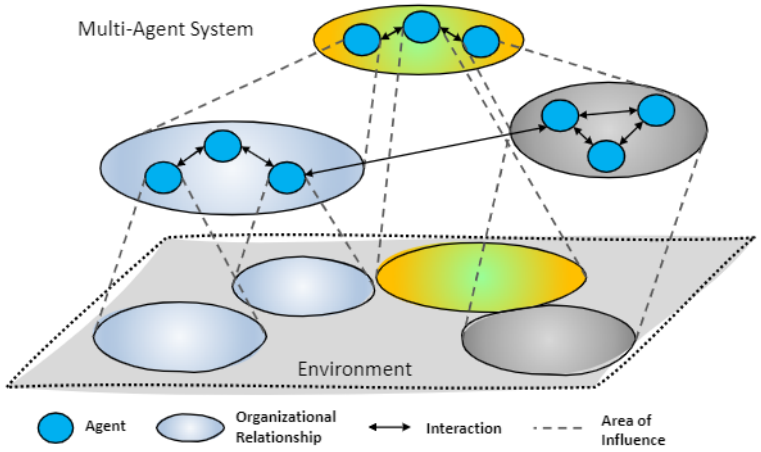

Figure 2.9: Generic scheme of a multi-agent system (De Santis and Reschke[43]).

2.5 Comunication Protocols

This section deals with the communication protocols used in this work. The FIPA-ACL protocol was used for communication among the agents, and the HTTP protocol was used

for each agent to access its ML algorithm more quickly, via the Flask API.

2.5.1 FIPA-ACL

FIPA is a global organization dedicated to promoting the intelligent agents sector, openly developing specifications supporting interoperability among agents and agent-based applications [44]. The FIPA-ACL message is made up of various parameters, as shown in Table 2.2, which can vary from circumstance to circumstance and are essential for effective iteration among agents. In general, the configuration of the message is determined from the parameters that specify the type of communication, the control of the communication, the description of the message, the content, and the agents (participants) involved.

Table 2.2: ACL message parameters [45].

Parameter	Description
performative	Type of the communicative act of the message
sender	Identity of the sender of the message
receiver	Identity of the intended recipients of the message
reply-to	Which agent to direct subsequent messages to within a conversation thread
content	Content of the message
language	Language in which the content parameter is expressed
encoding	Specific encoding of the message content
ontology	Reference to an ontology to give meaning to symbols in the message content
protocol	Interaction protocol used to structure a conversation
conversation-id	Unique identity of a conversation thread
reply-with	An expression to be used by a responding agent to identify the message
in-reply-to	Reference to an earlier action to which the message is a reply
reply-by	A time/date indicating by when a reply should be received

2.5.2 HTTP

Hypertext Transfer Protocol (HTTP) is a stateless communication protocol, client-server (request/response) type used for communication among devices on the Internet, as per

by Fielding et al. [46]. HTTP can be used to request web pages, download files, submit forms, and interact with APIs. Communication occurs through HTTP messages, each message has a method that indicates the action to be performed, a header with additional information (content type) and a body with the request or response data. The most commonly used HTTP methods are listed in table 2.3:

Table 2.3: HTTP methods and descriptions [47].

Method	Description
GET	Retrieves data or a resource from a specified URL. It is used to retrieve information without modifying it.
POST	Submits data or create a new resource on the server. It is used to send data to be processed by the server, often resulting in the creation of a new resource on the server.
PUT	Updates the existing resource with the new data. It replaces the entire resource or creates it if it does not exist.
DELETE	Deletes the specified resource from the server.
PATCH	Partially updates the existing resource with the provided data. For example, updating a specific field, like changing only the email address of a user's profile. PATCH does not create a new resource if it does not exist.
HEAD	Retrieves only the headers of a response. It is used to check the status or headers of a resource without fetching the actual content.
OPTIONS	Fetches the allowed methods and other information of the specified resource.
TRACE	Echoes back the received request to the client, allowing them to inspect the request and see any modifications made by intermediaries. Primarily used for diagnostic purposes.
CONNECT	Converts the request connection to a transparent TCP/IP tunnel, commonly used for establishing secure SSL/TLS connections through proxies.

2.6 Summary

Therefore, cybersecurity is a fundamental pillar in the face of the digital transformation that has been occurring in the world. With the increase in digitalization and the growing dependence on technology, ensuring the security of systems, networks, and information has

become essential for companies, governments, and individuals. Given this reality, network protection has become a challenge since the infrastructure has grown exponentially to support new demands, as well as in the processing and storage of information. As a result, centralized intrusion detection systems may present limitations and bottlenecks in dealing with this new reality, so new decentralized approaches are beginning to emerge to fill these gaps. This work presents an option for a decentralized intrusion detection system using multi-agent systems and machine learning.

Chapter 3

System Description

This chapter presents a general description of the proposed system, focusing on the most important points for building its simulation. A centralized NIDS system, despite offering unified management of network security, presents specific challenges and problems that can compromise its effectiveness, as it concentrates processing, analysis, and storage of data from the entire network in a single point. In view of this, for experimental purposes, the approach used in this NIDS seeks, through MAS and ML technologies, to create a decentralized and collaborative system in which each software agent plays its role in cybersecurity, seeking to improve threat identification through mutual collaboration.

The proposed system aims to distribute the detection of possible threats within the network through the computers available in the cloud so that each computer is supplied with 1 or more agents, each with an ML algorithm capable of detecting a possible threat. Furthermore, if an agent does not have sufficient aptitude to classify a sample correctly, it uses its collaborative skills to communicate with others and thus classify the dubious sample, as shown in Figure 3.1 .

Given this scenario, the system seeks to use the computational power of cloud resources to obtain better data processing time, versatility, and scalability for a NIDS system. Due to the memory limitations of the computer used for this experiment, only 6 agents were chosen. However, as it is an adaptive system, this number can be increased to as many as required.

To build an anomaly-based NIDS, it is necessary to choose a dataset in which there is already an organized and categorized network flow dataset. Therefore, the chosen dataset was the UNSW-NB15 [48]. The data was then pre-processed, checking for null values, removing useless columns, etc. Next, it was necessary to look at how the data is arranged in the dataset in order to check the correlation among the features. In this way, it is possible to reduce the number of columns that are actually important, thereby reducing training time. This was followed by normalization, as well as data balancing, and finally model training. The models chosen were DT, RF, KNN, SVM, MLP, and LR. Figure 3.1 shows the proposed architecture for the system.

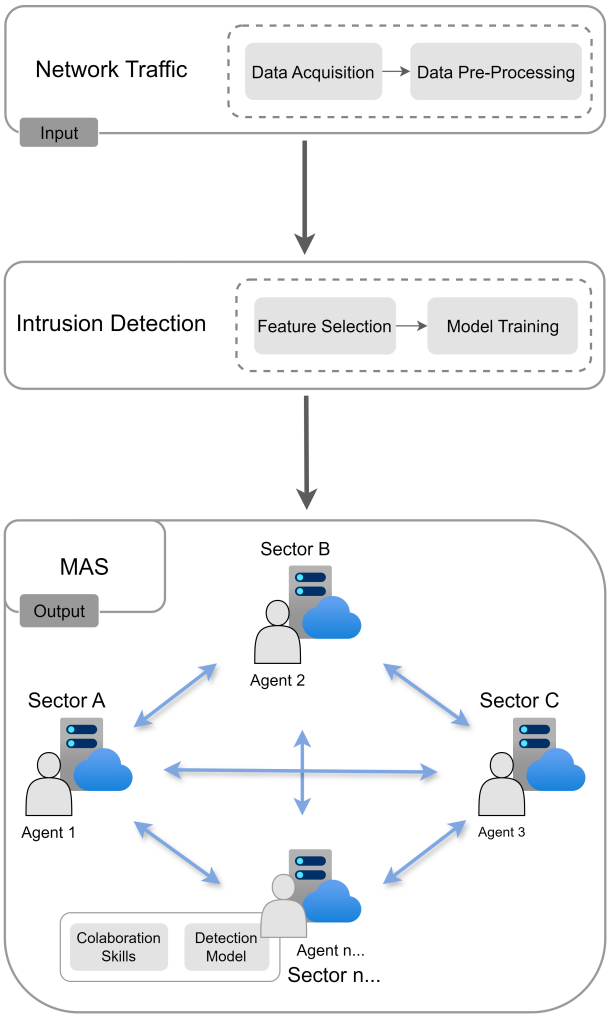

Figure 3.1: Proposed architecture.

For this experiment, the dataset was divided into training, testing, and simulation. For the simulation, the dataset was divided into different parts for each agent, with an equal number of samples, as seen in Figure 3.2. Each agent is responsible for an algorithm, so distributing this within a system, by expanding the analyses with different methods provides greater observation power to identify possible attacks, thus having a more holistic view of classification. In this way, within an organization, it is possible to collaboratively distribute a detection system in a scalable way in order to reduce bottlenecks caused by a large amount of data to be processed compared to a centralized system, and it is also possible to eliminate a single point of failure.

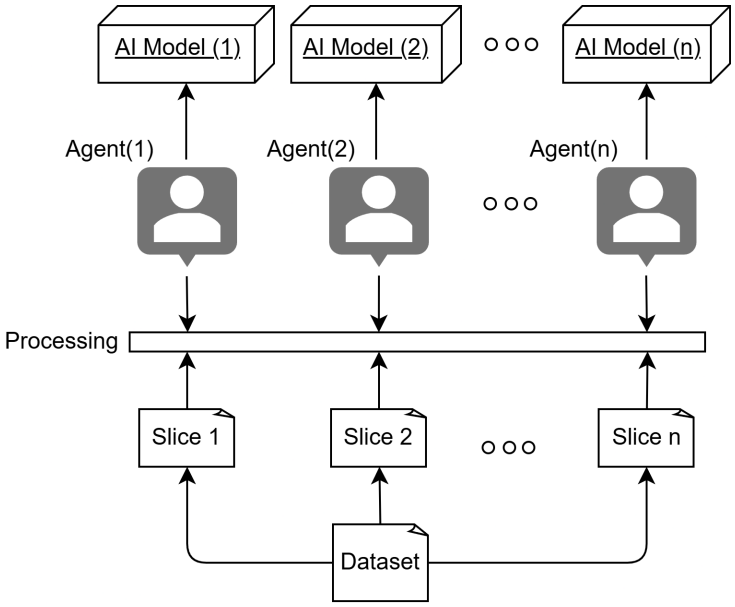

Figure 3.2: System simulation representation.

As illustrated in Figure 3.3, each agent follows a well-defined behavioral pattern. When it is started, the agent receives an identifying name and a specific skill (algorithm), according to transition t2. During data analysis, transition t4, it establishes an HTTP connection to access its individual ML model via the REST API. Should the model fail to classify a sample with sufficient confidence, the agent triggers the others through the FIPA-ACL protocol (t5) to obtain a collaborative classification. The receiving agents process the sample (t12), perform their analyses, and forward the results back to the

requesting agent (t14).

Upon receiving the responses (t7), the agent disregards those with a probability lower than its initial analysis and uses majority voting to define the final classification (t8). To ensure greater robustness, a time limit was established for accepting responses; opinions sent after this period are disregarded. Furthermore, a scoring-based approach was adopted: if an agent performs poorly, its algorithm is replaced by one with a higher score (t15). In short, the system integrates agents with predictive, communication and collaboration capabilities, increasing the efficiency of the NIDS.

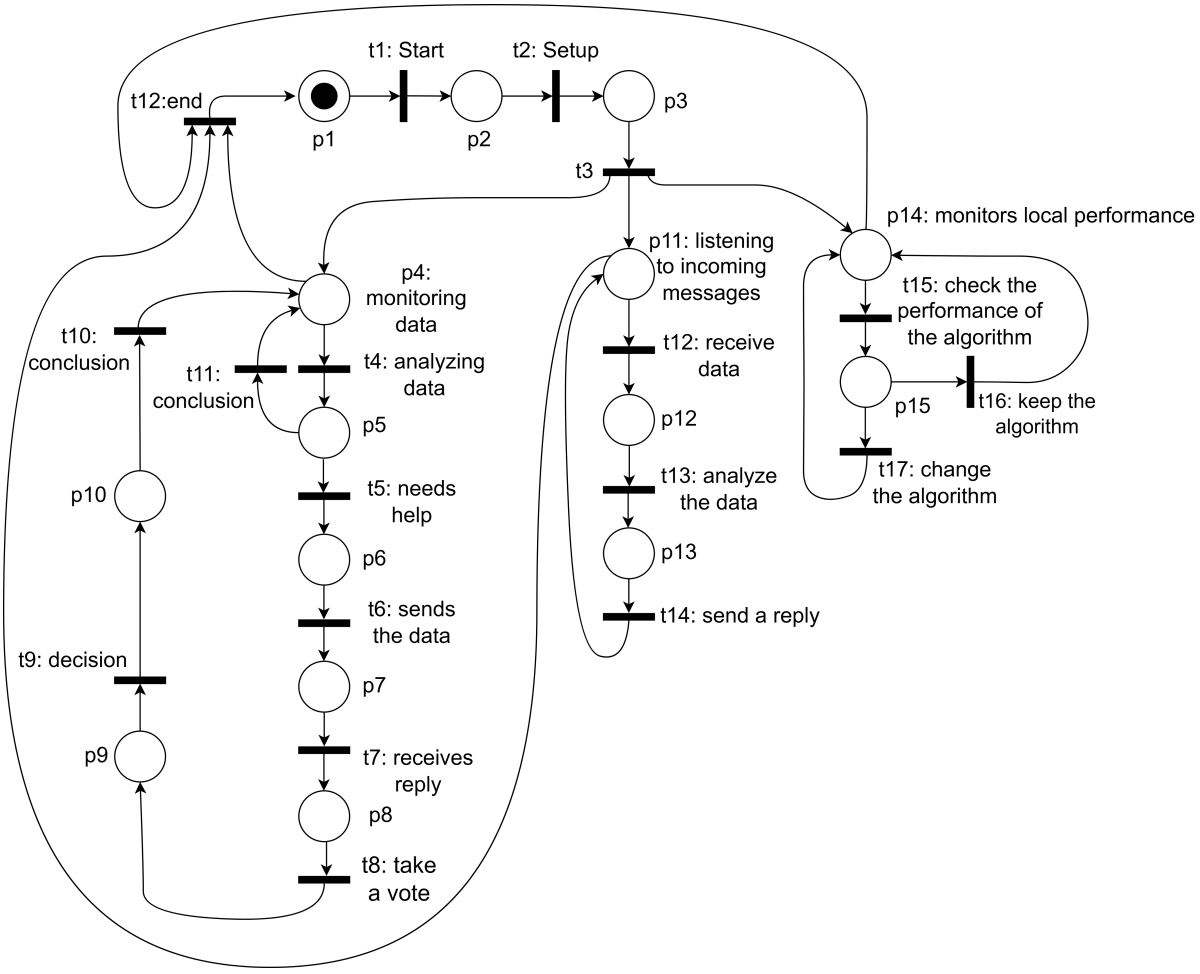

Figure 3.3: Behavior of agent.

Figure 3.4 shows how collaboration among agents works in general. Each computer on the network can be configured with 1 or more agents to monitor that point on the network, thus having 1 or more algorithms to verify the data. When monitoring a point on the network, one computer, even with several agents and algorithms, may have difficulty correctly classifying a sample. This is how agent collaboration kicks in, allowing this sample to be transmitted for all the other agents to check. Next, the sender receives all the responses and filters them to select only those with a higher percentage than its own. Finally, this agent opens a majority vote to classify the doubtful sample.

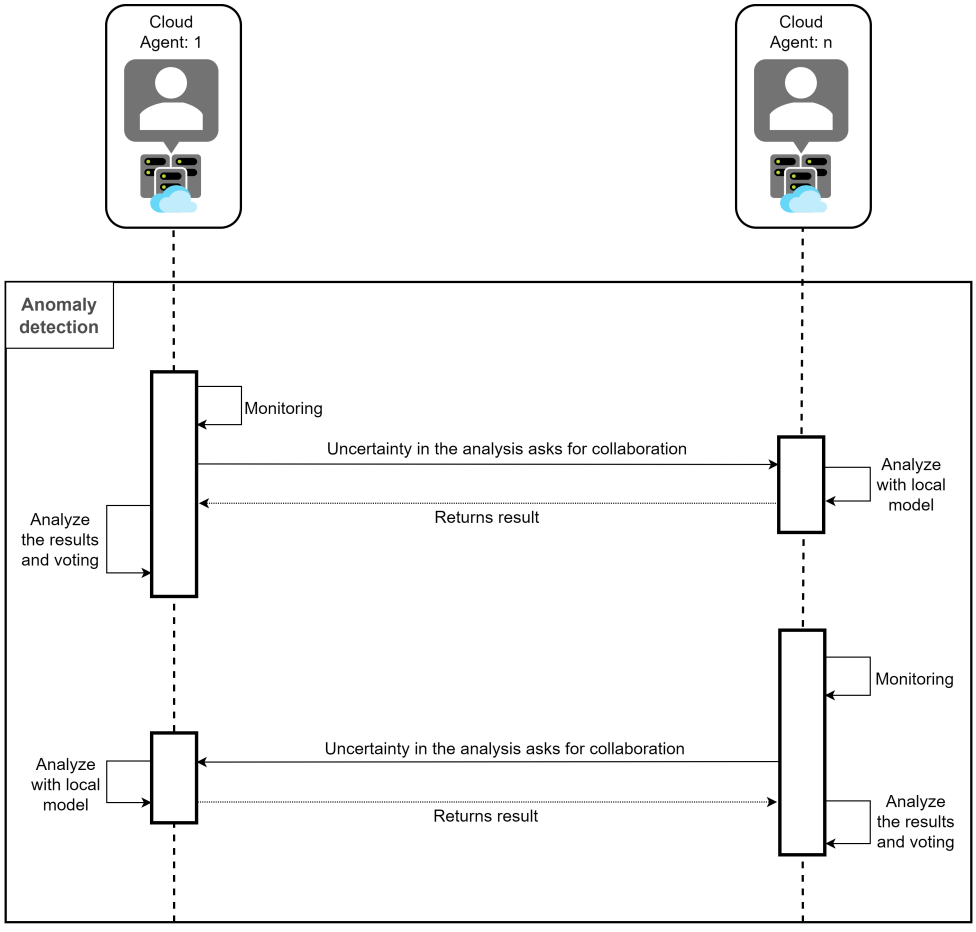

Figure 3.4: Interacting among agents.

Chapter 4

Development of the system

This chapter demonstrates the implementation of the system described previously. Development and testing were performed on a personal computer, consisting of an Inter(R) Core(TM) i5-11260H processor with 2.60GHZ, 8GB of ram and 256GB of storage. As an ML-based IDS system needs to be kept up to date, it is necessary to continually seek to label new unknown attacks, thus quantitatively improving the dataset. In this way, the amount of data only increases, leading to new approaches to operating a NIDS system. One of these is to make a decentralized system, so that information processing is distributed in order to reduce the time to action, as per by Louati et al. [3]. Within this scenario, using a MAS is a possible way for the system to become scalable and fully customizable, according to the approach required.

The system was developed using two main technologies: Java and Python. On the Java side, the MAS was implemented, where each agent has access to an ML algorithm to classify the samples in a data file for simulation. On the Python side, the training process and provision of the ML algorithms were performed. Table 4.1 below summarizes the technologies used.

Table 4.1: Languages and tools used

Language	Tool	Description
Java	Java Agent DEvelopment (JADE)	Develop interactions among agents
Python	Pycaret	Train and save ML algorithms
Python	Scikit-learn	Data preprocessing; load and make available trained algorithms
Python	Flask	REST API (REpresentational State Transfer Application Programming Interface) Server

4.1 Data curation

4.1.1 Dataset Preprocessing

The UNSW-NB15 dataset, made from raw network packets that have already been properly organized and classified, contains 45 features and nine types of attacks: Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms. The files used for this experiment were *'UNSW_NB15_training-set.csv'*, which contains 175,341 samples, and the *'UNSW_NB15_testing-set.csv'* with 82,332 samples. For this work, only binary classification was considered, using the file *'UNSW_NB15_training-set.csv'* for training (80% of samples) and testing (20%). To carry out the simulation, *'UNSW_NB15_testing-set.csv'*. Table 4.2 shows the number of samples for each class.

Table 4.2: Dataset description

	Class 1 (Normal)	**Class 0 (Anomalous)**
Training & Testing Data	119,341	56,000
Simulation Data	45,332	37,000

As the focus of this work is to verify the results of a MAS, it is necessary to pre-process the data in a way that brings the best possible performance to the models, in this sense, we used as a basis some steps as demonstrated by Dubey [49].

Even though the dataset was well organized, it was still necessary to observe and check the data for pre-processing. Firstly, it was checked how the dataset looks, so that the presence of empty values in column *'service'*, and after removing these values, the dataset now has 81173 rows and 45 columns. After removing the null values, columns that would

not be very helpful for our purpose were removed, so the column *'id'* was removed. which is just the numbering of each column in the dataset, and *'attack_cat'*, that only shows the type of attack for each sample, however as this work is restricted to binary classification only, this column was removed.

Then only 43 columns remained, so 3 of them were columns with categorical variables *('proto', 'service' e 'state')*, which need to be transformed into a numerical representation, and for this, the one hot encoding method was chosen. As specified by Brownlee [50], for representation, one hot encoding creates a binary column for each category, where for each entry, a column is defined that corresponds to category 1 and for all other columns, 0. This transforms the text into a sparse binary vector. After applying one hot encoding, the data set gains new columns since this technique needs to create a numerical representation for the categorical variables. With this, the dataset ends up with 59 columns.

Soon after, the data was normalized, using the Min-max scaler technique, which consists of transforming, rescaling the numerical values present, to values between 0 and 1, as stated by Géron [51]. This way, it is ensured that all features contribute equally to a model, especially when they are at different scales. Briefly, it is calculated with the formula:

$$x_{\text{norm}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \tag{4.1}$$

So that x is the actual value in the dataset, the x_{min} is the minimum value within the data set, and the x_{max} is the maximum value.

Considering that several algorithms operate in different ways, it was noticeable to observe how the classes are distributed in the dataset. As it was observed that there were 61685 (76%), more than twice as many attack samples (label 0) compared to normal traffic samples (label 1), 19488 (24%). Therefore, the Synthetic Minority Oversampling Technique (SMOTE) was chosen to balance the dataset since some algorithms may lean towards one side during training. In according to Fernández et al. [52] SMOTE aims to rebalance a set of training data. However, instead of creating copies of minority classes, it replicates synthetic data created by interpolating data from minority classes in a defined

neighbourhood. This procedure therefore concentrates on the feature space rather than the data space, basing itself on the relationships and values that the data presents.

4.1.2 Feature Selection

Often when analyzing a dataset with many features, it is interesting to filter them to choose only the most important ones so that by reducing the number of features, it provides a reduction in the training time of ML models, and to do this, the correlation matrix is applied. The correlation matrix is a row-by-column ordering to create a set of coefficients that demonstrate the correlation of the variables studied. The rows and columns refer to the specific variables studied, so the correlation matrix indicates the linear association between each pair of variables, labeling both columns and rows, as per by Hadd and Rodgers [53]. Figure 4.1 below demonstrates the matrix found.

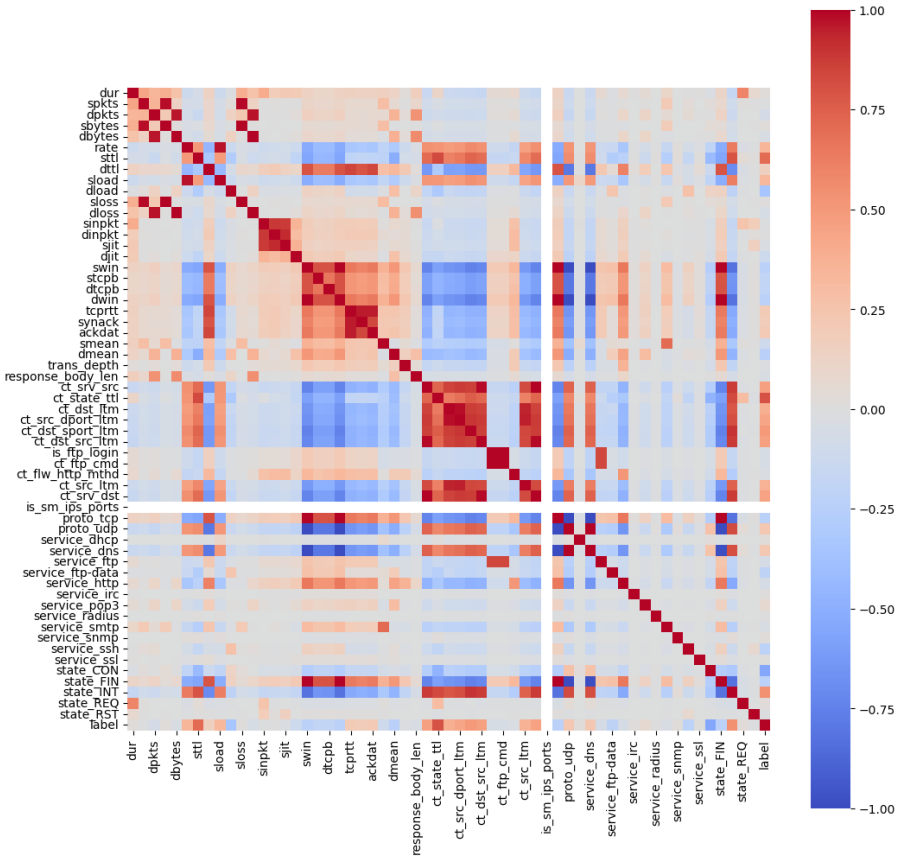

Figure 4.1: Correlation matrix.

As seen in the image above, 1 indicates a perfect positive correlation, -1 is a negative correlation, and 0 is no correlation. For example, the features *'sttl'* and *'ct_state_ttl'*, indicate a good correlation. On the other hand, the features *'dload'* and *'state_CON'* have a negative correlation. Also, it can be observed that the areas in blue have a low correlation, which may indicate that pairs of these variables are independent or have a complex relationship that is not linear. This makes it possible to filter only the features that have a good correlation within the sample space, where a correlation between 0 and 0.3 is considered weak, between 0.3 and 0.6 is considered moderate, and greater than 0.6 is considered strong. For this case study, it was chosen to select features with correlation greater than 0.3. With this, the features which were 59, dropped to 14.

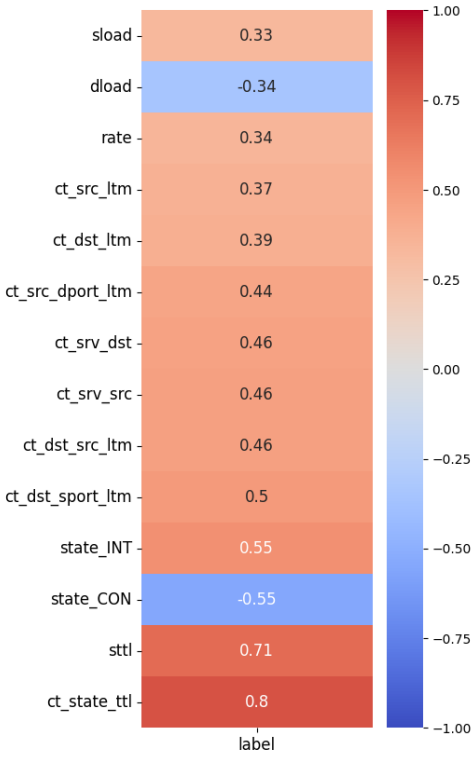

Figure 4.2: Selected features.

As illustrated in figure 4.2, the 14 features chosen for the training were the following:

sload (Source Load - Represents the bitrate per second sent from the source to the destination); dload (Destination Load - Refers to the bitrate per second received by the

destination); rate (Represents the bit rate per second of network traffic); ct_src_ltm (Count of Connections to Same Source Address in 100 Connections); ct_dst_ltm (Count of Connections to Same Destination Address in 100 Connections); ct_src_dport_ltm (Count of Connections to Same Source Address and Destination Port in 100 Connections); ct_srv_-dst (Count of Connections to Same Service and Destination Address in 100 Connections); ct_srv_src (Count of Connections to Same Service and Source Address in 100 Connections); ct_dst_src_ltm (Count of Connections to Same Source and Destination Address in 100 Connections); ct_dst_sport_ltm (Count of Connections to Same Destination Address and Source Port in 100 Connections); state_INT (State: INT - Indicates the state of the intermediate connection); state_CON (State: CON - Indicates the connection status if it was established and completed successfully); sttl (Source to Destination Time to Live - Indicates the TTL (Time to Live) value of the packet from source to destination) and ct_state_ttl (Count of Connections for Each State Based on TTL Ranges);

4.2 MAS Solution

The developed REST API has the sole responsibility of loading the already trained ML model and making the "model.predict_proba" function available, from an HTTP URL. In the case of local execution, each ML algorithm is addressed to a port as follows: RF port 5000, MLP port 5001, LR port 5002, KNN port 5003, DT port 5004 and SVM port 5005. With this, each agent uses the POST method, with the respective URL (ex: http://localhost:5000/predict") sending the data to be analyzed, through an HTTP connection to the REST API. This endpoint returns a JSON response in an array of float values, composed of a vector with two complementary probability values, for example: prob:[20, 80]. In this vector, the first position refers to label 0 (threat) and the second to label 1 (normal activity). As the values are complementary, only the second one is selected for analysis. For this purpose, an uncertainty interval was defined for the classification. For example, if the agent performs the analysis and the result is ≥ 80, there is a high probability that it is a normal activity. If the result is ≤ 20, there is a good probability

of certainty that it is a threat. In other words, if the result found by the agent is in the interval among these values, doubt arises, and this agent will try to communicate with the other agents to collaborate in the classification of this sample. For the sake of simplicity, these uncertainty intervals will be referred to as confidence levels and explored in session 5.

The implementation of agents in Java involves communication with these Python servers and interaction among other agents, according to the following set of behaviors:

- Process individual data: Analyzes a specific range of items from the simulation file. This range is defined when the agent is initialized. The analyzed items can trigger a voting interaction by the agent if the certainty of the answer is below a stipulated limit. This initiates the behavior of sending questions.

- Listen to messages: Receives messages and acts on the received content, updating the agent's current state and activating other behaviors.

- Send questions: Sends the question message to other agents who do not use the same ML model as the agent in doubt. A time limit is set for receiving responses, and then voting is carried out according to the number of agents found.

- Analyze questions: Part of the voting interaction, where the question message is processed by opinion made by an an agent with a doubt results in the sending of a message with the answer.

- Analyze responses: Part of the voting interaction, where the responses to a question are collected. At the end of the collection, a vote is taken in search of an answer, which, if there is a tie, is resolved by calculating the average of the results.

- Log metrics: Records a set of metrics to a log file (metrics.log) for later analysis.

- Analyze Metrics: Individual agent analysis of its metrics. At this point, it is identified whether there is a need to change the ML algorithm observing the API response time and the agent's current score. The change initiates the skill switching behavior.

- Switch Skills: Sends a message to other agents asking for their scores.
- Analyze score: Stores the scores of other agents. At the end, it searches for the best score and selects the corresponding skill (algorithm).

With this in mind, the system can be described in more detail. On being started, the agent triggers the main behavior "createBehaviourInformListener" used to listen to messages, which triggers other behaviors. First, "ReadyMessage", which triggers the behavior "createProcessIteratorBehavior", which makes the agent create an iterator over the number of defined items. Then the function "analyzeData" is called, which takes each item from the iterator and sends it to the REST API, creating an HTTP connection and using the POST method on the pre-defined port for each skill (algorithm).

When the agent receives the result from the REST API, if the value is within the defined uncertainty range, this sample is associated with a random and unique identifier (ID) created by Java from the UUID (Universally Unique Identifier) library. From this, the agent activates the "createBehaviourAskForOpinion" behavior that searches for other agents with different skills (algorithms) in the JADE Direct Facilitator and sends a message of the "OpinionRequestMessage" type to all of them containing the sample with the ID to be analyzed. Upon sending the sample, the agent also sets the time to later calculate the response duration.

Another active agent will receive the "OpinionRequestMessage" message, capture the data and trigger the "createBehaviourAnswerOpinionRequests" behavior, which will make it query the REST API to obtain the result predicted by its individual ML algorithm. It will then call the "OpinionResponseMessage" class and send the result to the sending agent, as well as the time at which it responded.

Once the sending agent receives the message, it activates the "createBehaviourAnalyseOptionResponse" behavior, which calculates the total time between the question and the answer, and checks whether all agents have answered it. If so, the "createBehaviourAnalyzeResult" behavior is activated, which checks the answers received, considering only values greater than its individual result. In this way, voting is carried out, and the classification

for that sample is defined. If there is a tie, the average of all the results is calculated and if the result is greater than 0.5, it is considered an attack. Otherwise, that sample is considered normal activity. There is also the possibility that an agent may take a long time to respond. For this, a fixed time of 5 seconds is configured for receiving messages. However, when a new agent is registered in Direct Facilitator, 1 second is added to this time. Therefore, if an agent responds after this defined time, the sending agent receives the message but does not consider the result, deletes the message ID, and increments the timeout variable by 1 to count the number of times this occurred for each vote made.

In order to measure the system's evolution throughout the simulation, some metrics were defined. The metrics accuracy, precision, recall and F1-score are used to compare individual performance with collaborative performance, also varying the uncertainty intervals. For this purpose, the simulation dataset is already properly classified, so when an agent analyzes a result, this value is compared with the real value of the sample in the dataset. With this, it is possible to calculate the metrics through the number of false negatives (FN), false positives (FP), true positives (TP) and true negatives (TN).

Each agent, when analyzing a sample, writes the following content to a log file (for example, exec-KNN.log): its name, its action (analysis, question, received an answer or received a question) with the sample ID, the number of items processed, whether it was certain when analyzing the sample, the probability found, and the correct classification verified in the dataset. In addition, each time an agent receives a message, it also records it in this log file. Figure 4.1 shows an example.

```
1  [2024-10-24T09:09:53:59.783] [INFO]: agent-003: Skill KNN registered
2  [2024-10-24T09:09:54:22.017] [INFO]: agent-003: RECEIVE(READY)
3  INFO: API Response: [0.0,1.0]
4  [2024-10-24T09:09:54:23.239] [INFO]: agent-003: ANALYZE(1eb2099b-74d4-4614-b090-a31d70796f64) count=1 sure=true prob=1.0 label=true correct=true
5  [2024-10-24T09:09:54:23.252] [INFO]: agent-003: SEND(METRICS_REQUEST): type=all count=1
6  [2024-10-24T09:09:54:23.253] [INFO]: agent-003: RESULT(1eb2099b-74d4-4614-b090-a31d70796f64) : result is attack? true : prob? 1.0 : label? true : correct? true
7  [2024-10-24T09:09:54:23.259] [INFO]: agent-003: RECEIVE(OPINION_REQUEST)
8  INFO: API Response: [0.0,1.0]
9  [2024-10-24T09:09:54:23.289] [INFO]: agent-003: ANSWER(OPINION) : 840f6a2f-ea66-4045-8855-cd0dd72cf60e : agent-002
10 INFO: API Response: [0.0,1.0]
11 [2024-10-24T09:09:54:23.336] [INFO]: agent-003: ANALYZE(f25e7897-3cb1-4e2a-a0aa-b572c1d32f62) count=2 sure=true prob=1.0 label=true correct=true
12 [2024-10-24T09:09:54:23.336] [INFO]: agent-003: RESULT(f25e7897-3cb1-4e2a-a0aa-b572c1d32f62) : result is attack? true : prob? 1.0 : label? true : correct? true
13 [2024-10-24T09:09:54:23.413] [INFO]: agent-003: RECEIVE(OPINION_REQUEST)
14 INFO: API Response: [0.8,0.2]
15 [2024-10-24T09:09:54:23.442] [INFO]: agent-003: ANSWER(OPINION) : dc6fca4f-498e-4738-a2fa-1c0f613ea2b3 : agent-002
16 INFO: API Response: [0.0,1.0]
17 [2024-10-24T09:09:54:23.463] [INFO]: agent-003: ANALYZE(8d7ac35e-b7e6-4ac9-8cce-d4725141fb08) count=3 sure=true prob=1.0 label=true correct=true
18 [2024-10-24T09:09:54:23.463] [INFO]: agent-003: RESULT(8d7ac35e-b7e6-4ac9-8cce-d4725141fb08) : result is attack? true : prob? 1.0 : label? true : correct? true
19 [2024-10-24T09:09:54:23.513] [INFO]: agent-003: RECEIVE(OPINION_REQUEST)
20 INFO: API Response: [0.2,0.8]
```

Figure 4.3: Example of an agent's activity log.

With this in mind, it is clear that some things can happen that disrupt the system as a whole, for example, an agent that is not very accurate may trigger a situation in which it asks too many questions, overloading the other agents, or the REST API may even take too long to deliver the answer. Therefore, a mechanism was developed to try to continuously improve the system if one of these situations occurs. Initially, the round robin scheduling algorithm was used, which switches to the next algorithm on the list, in order to observe how the system would behave. Later, this form of switching was changed, a window for the number of questions was defined. If for every 500 items processed, the agent asks 100 questions or if the REST API takes an average of more than 0.1 seconds to answer, the "createBehaviourAnalyseMetrics" behavior checks these two conditions and if either of them is positive, the "createBehaviourTryChangeSkill" behavior is activated, which checks the skill (algorithm) that has the highest score, which is calculated by the difference between (TP + TN) - (FP + FN) of the results of the samples analyzed individually. Since this result is individual for each agent, the faulty agent sends a message to all the others to check which agent has the best score, and upon receiving the response, it activates the "createBehaviourAnalyzePointsResponse" behavior, which checks if there is a skill (algorithm) with a better score and, if so, performs the change. When changing the skill, the agent resets all of its individual metrics.

Other metrics were also used to observe the behavior of agents, such as: API response

time (API time), total time between asking and receiving the answer (Ask time), and total voting time, when all agents respond, and the agent makes a decision.

So that these metrics can be analyzed later, each agent records its results over time in the same "metrics.log" file, containing: Its name, skill (algorithm), TP, TN, FP, FN, Accuracy, Precision, Recall, F1-score, API time, Ask time, Voting time and Type. The Type metric indicates the type of performance, whether it is individual performance, which does not consider the help of other agents for the classification and calculation of other performance metrics for each skill. Or whether it is collaborative performance considering the responses obtained by other agents. With this, there are 4 options for the Type metric, which are: self (individual performance that does not reset the data when changing skills); self-model (individual performance which resets values made by the previous skill (algorithm)); all (collective performance which does not reset the data when changing skills) and all-model (collective performance which resets values made by the previous skill (algorithm)).

```
Name=agent-004, Skill=DecisionTree, Type=self, F1=0.968, Accuracy=0.950, Precision=0.962, Recall=0.974, Total=100, Asks=0, Timeouts=0, Points=90, TP=76, FP=3, TN=19, FN=2, API time=0.015, Ask time=0.000, Voting time=0.000
Name=agent-004, Skill=DecisionTree, Type=all-model, F1=0.968, Accuracy=0.949, Precision=0.962, Recall=0.974, Total=99, Asks=0, Timeouts=0, Points=89, TP=76, FP=3, TN=18, FN=2, API time=0.000, Ask time=0.000, Voting time=0.000
Name=agent-004, Skill=DecisionTree, Type=self-model, F1=0.968, Accuracy=0.950, Precision=0.962, Recall=0.974, Total=100, Asks=0, Timeouts=0, Points=90, TP=76, FP=3, TN=19, FN=2, API time=0.015, Ask time=0.000, Voting time=0.000
Name=agent-004, Skill=DecisionTree, Type=all, F1=0.968, Accuracy=0.950, Precision=0.962, Recall=0.974, Total=100, Asks=0, Timeouts=0, Points=90, TP=76, FP=3, TN=19, FN=2, API time=0.000, Ask time=0.000, Voting time=0.000
Name=agent-004, Skill=DecisionTree, Type=all, F1=0.969, Accuracy=0.950, Precision=0.962, Recall=0.975, Total=101, Asks=0, Timeouts=0, Points=91, TP=77, FP=3, TN=19, FN=2, API time=0.000, Ask time=0.000, Voting time=0.000
Name=agent-001, Skill=MLP, Type=self, F1=0.987, Accuracy=0.980, Precision=0.974, Recall=1.000, Total=100, Asks=9, Timeouts=0, Points=96, TP=75, FP=2, TN=23, FN=0, API time=0.015, Ask time=1.482, Voting time=2.414
Name=agent-001, Skill=MLP, Type=all-model, F1=0.993, Accuracy=0.989, Precision=0.986, Recall=1.000, Total=95, Asks=0, Timeouts=0, Points=93, TP=73, FP=1, TN=21, FN=0, API time=0.000, Ask time=1.482, Voting time=2.414
Name=agent-001, Skill=MLP, Type=self-model, F1=0.987, Accuracy=0.980, Precision=0.974, Recall=1.000, Total=100, Asks=9, Timeouts=0, Points=96, TP=75, FP=2, TN=23, FN=0, API time=0.015, Ask time=1.482, Voting time=2.414
Name=agent-002, Skill=LogisticReg, Type=self, F1=0.975, Accuracy=0.960, Precision=0.951, Recall=1.000, Total=100, Asks=22, Timeouts=0, Points=92, TP=77, FP=4, TN=19, FN=0, API time=0.021, Ask time=0.851, Voting time=2.737
Name=agent-002, Skill=LogisticReg, Type=all-model, F1=0.992, Accuracy=0.988, Precision=0.984, Recall=1.000, Total=83, Asks=0, Timeouts=0, Points=81, TP=63, FP=1, TN=19, FN=0, API time=0.000, Ask time=0.851, Voting time=2.737
Name=agent-002, Skill=LogisticReg, Type=self-model, F1=0.975, Accuracy=0.960, Precision=0.951, Recall=1.000, Total=100, Asks=22, Timeouts=0, Points=92, TP=77, FP=4, TN=19, FN=0, API time=0.021, Ask time=0.851, Voting time=2.737
Name=agent-005, Skill=SVM, Type=self, F1=0.926, Accuracy=0.890, Precision=0.863, Recall=1.000, Total=100, Asks=31, Timeouts=0, Points=78, TP=69, FP=11, TN=20, FN=0, API time=0.021, Ask time=2.344, Voting time=3.781
Name=agent-005, Skill=SVM, Type=all-model, F1=1.000, Accuracy=1.000, Precision=1.000, Recall=1.000, Total=84, Asks=0, Timeouts=0, Points=84, TP=59, FP=0, TN=25, FN=0, API time=0.000, Ask time=2.344, Voting time=3.781
Name=agent-005, Skill=SVM, Type=self-model, F1=0.926, Accuracy=0.890, Precision=0.863, Recall=1.000, Total=100, Asks=31, Timeouts=0, Points=78, TP=69, FP=11, TN=20, FN=0, API time=0.021, Ask time=2.344, Voting time=3.781
Name=agent-001, Skill=MLP, Type=all, F1=0.994, Accuracy=0.990, Precision=0.987, Recall=1.000, Total=100, Asks=0, Timeouts=0, Points=98, TP=78, FP=1, TN=21, FN=0, API time=0.000, Ask time=1.482, Voting time=2.414
Name=agent-001, Skill=MLP, Type=all, F1=0.994, Accuracy=0.990, Precision=0.988, Recall=1.000, Total=101, Asks=0, Timeouts=0, Points=99, TP=79, FP=1, TN=21, FN=0, API time=0.000, Ask time=1.734, Voting time=2.414
Name=agent-002, Skill=LogisticReg, Type=all, F1=0.993, Accuracy=0.990, Precision=0.987, Recall=1.000, Total=100, Asks=0, Timeouts=0, Points=98, TP=74, FP=1, TN=25, FN=0, API time=0.000, Ask time=2.990, Voting time=3.870
Name=agent-002, Skill=LogisticReg, Type=all, F1=0.993, Accuracy=0.990, Precision=0.987, Recall=1.000, Total=101, Asks=0, Timeouts=0, Points=99, TP=75, FP=1, TN=25, FN=0, API time=0.000, Ask time=2.990, Voting time=3.870
Name=agent-005, Skill=SVM, Type=all, F1=1.000, Accuracy=1.000, Precision=1.000, Recall=1.000, Total=100, Asks=0, Timeouts=0, Points=100, TP=72, FP=0, TN=28, FN=0, API time=0.000, Ask time=2.820, Voting time=3.998
Name=agent-005, Skill=SVM, Type=all, F1=1.000, Accuracy=1.000, Precision=1.000, Recall=1.000, Total=101, Asks=0, Timeouts=0, Points=101, TP=72, FP=0, TN=29, FN=0, API time=0.000, Ask time=2.820, Voting time=3.998
```

Figure 4.4: General results register.

Finally, to analyze the data, the information contained in the "metrics.log" file is passed to a table file in xlsx format to later create graphs and tables.

Chapter 5

Results and Discussions

5.1 Training Models

The process of training an ML model can often be time-consuming, as it involves adjusting the algorithm's hyperparameters until a satisfactory result is achieved. In light of the fact that different models are being employed, a tool was used to facilitate training and hyperparameters tuning. This tool called PyCaret, created by Ali [54] is an open-source low-code tool that automates and facilitates ML workflows, bringing greater speed and versatility to solving certain tasks.

Hyperparameter tuning was done using the Random Grid Search algorithm, where a set of possible hyperparameters is defined for each model. This way, instead of exhaustively testing all combinations (as in a full grid search), it evaluates a specific number of random samples from this hyperparameter space, which is more time efficient. For each model there are also the following hyperparameters found in Table 5.1:

Then the training was carried out, and Table 5.2 shows the results obtained.

Table 5.1: Hyperparameters.

Classifier	**Hyperparameters.**
RF	$n\_estimators = 100$, $criterion = gini$, $max\_depth = None$, $min\_samples\_split = 2$, $min\_samples_leaf = 1$, $max\_features = sqrt$, $n\_jobs = -1$.
DT	$criterion = gini$, $max\_depth = None$, $min\_samples\_split = 2$, $min\_samples\_leaf = 1$.
KNN	$n\_neighbors = 5$, $weights = uniform$, $metric = minkowski$, $p = 2$, $n\_jobs = -1$.
LR	$C = 1.0$, $fit\_intercept = True$, $max\_iter = 1000$, $penalty =' l2'$, $solver = lbfgs$.
SVM	$C = 1.0$, $break\_ties = False$, $cache\_size = 200$, $class\_weight = None$, $coef0 = 0.0$, $decision\_function\_shape = ovr$, $degree = 3$, $gamma = auto$, $kernel = linear$, $max\_iter = -1$, $probability = False$, $random\_state = None$, $shrinking = True$, $tol = 0.001$, $verbose = False$.
MLP	$activation = relu$, $alpha = 0.0001$, $hidden\_layer\_sizes = (100,)$, $learning\_rate = constant$, $max\_iter = 500$, $solver = adam$.

Table 5.2: Accuracy, precision, recall, F1-score and training time.

Classifier	**Accuracy**	**Recall**	**Precision**	**F1**	**TT (Sec)**
RF	0.9908	0.9922	0.9894	0.9908	1.3400
DT	0.9859	0.9853	0.9864	0.9858	0.3900
KNN	0.9827	0.9835	0.9819	0.9827	0.3100
LR	0.9587	0.9926	0.9296	0.9601	0.2500
SVM	0.9556	0.9987	0.9194	0.9574	0.1000
MLP	0.9768	0.9852	0.9689	0.9770	66.4950

Table 5.2 shows that, in general, all algorithms performed well for classification. In particular RF presented the best accuracy and F1-score, which indicates that it can be effective in classifying data and can also balance precision and recall (sensitivity) well in the analysis of samples, being able to correctly identify positive cases and avoid false positives. Its training time is average, considering the complexity of this algorithm.

DT demonstrated solid performance, even though it did not excel in any specific metric, indicating a good balance between successfully identifying positive cases and avoiding false positives. Its relatively fast training time is an additional advantage.

KNN had the lowest recall of all models, which means it has the highest probability of missing positive cases. However, the rest of the metrics are quite consistent. Its training time ends up being a positive point, making it a viable option for smaller data sets or when a fast response time is required.

LR performed well. Its strong point is recall, indicating that the model is effective in detecting positive cases. On the other hand, its low precision suggests that the model generates a higher number of false positives. Although it does not have the best F1-score, the model offers a good balance between performance and training speed, making it a good option in cases where minimizing false negatives is crucial.

The SVM algorithm has the highest recall, indicating that it is quite effective in detecting the positive class. However, its precision is relatively lower, suggesting that it can generate considerable false positives. Its fastest training time makes it an efficient choice but may not be ideal for scenarios where precision is more important.

Finally, the MLP demonstrated good accuracy but lags behind ensemble models such as Random Forest. Its high Recall indicates a good ability to identify the positive class, and its accuracy is balanced, with few false negatives. On the other hand, its long training time, demanding a higher computational cost, suggests that it may be a limitation in real-time environments.

5.2 Simulation Results

To measure the results, it was decided to compare the system based on the metrics accuracy, precision, recall and F1-score between when the agents act individually, that is, without considering the help of others, and when they act collaboratively. Another aspect to be compared is the confidence levels, varying between the options: 70%, 80%, and 90% certainty.

The confidence levels indicate that if the result of the analysis obtained by an agent is lower than one of these values, the agent will be in doubt and will seek help from other colleagues, i.e. they will only classify correctly if the value is above this percentage. This comparison among levels helps to better understand the system's behavior. By varying the confidence levels, it is possible to see if the system's dynamics are affected as the number of questions asked begins to increase.

Below, each figure shows the time graph for each agent and algorithm, comparing their individual performance with the collaborative performance. Using fewer agents would mean fewer results to analyze, thus generating little information for graphs and model comparisons. On the other hand, the ideal would be to use more agents with more varieties of algorithms, but due to hardware limitations, the ideal number was 6 agents.

5.2.1 Confidence at 70%

Looking at Figures 5.1 and 5.2, it can be seen that only agents 000, 002, and 005 managed to improve their accuracy through collaboration with the other agents, when comparing both the last performance values.

Accuracy

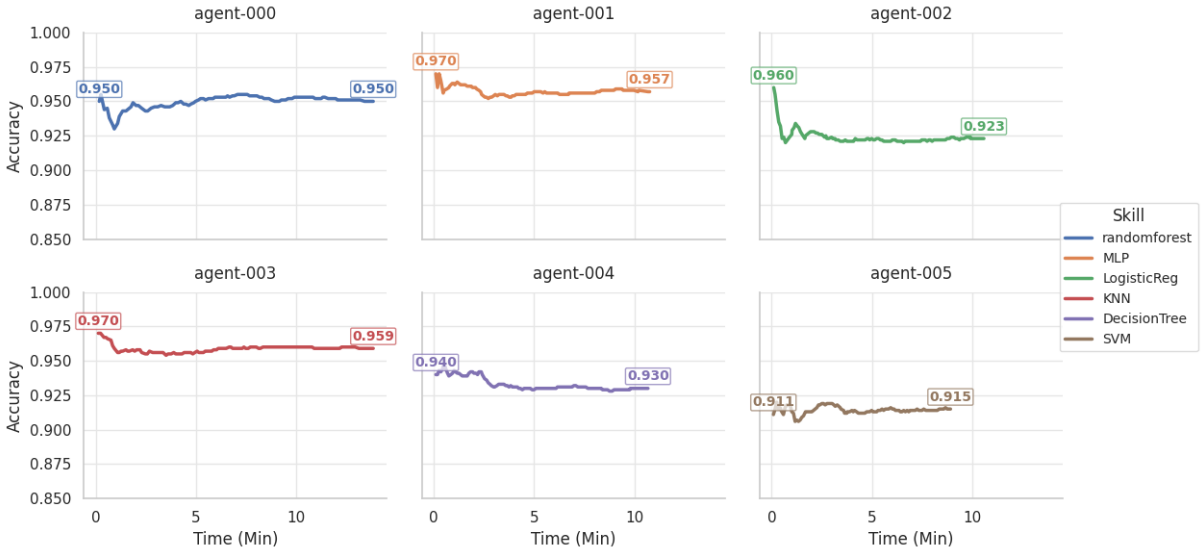

Figure 5.1: Accuracy - Individual Performance - Confidence at 70%.

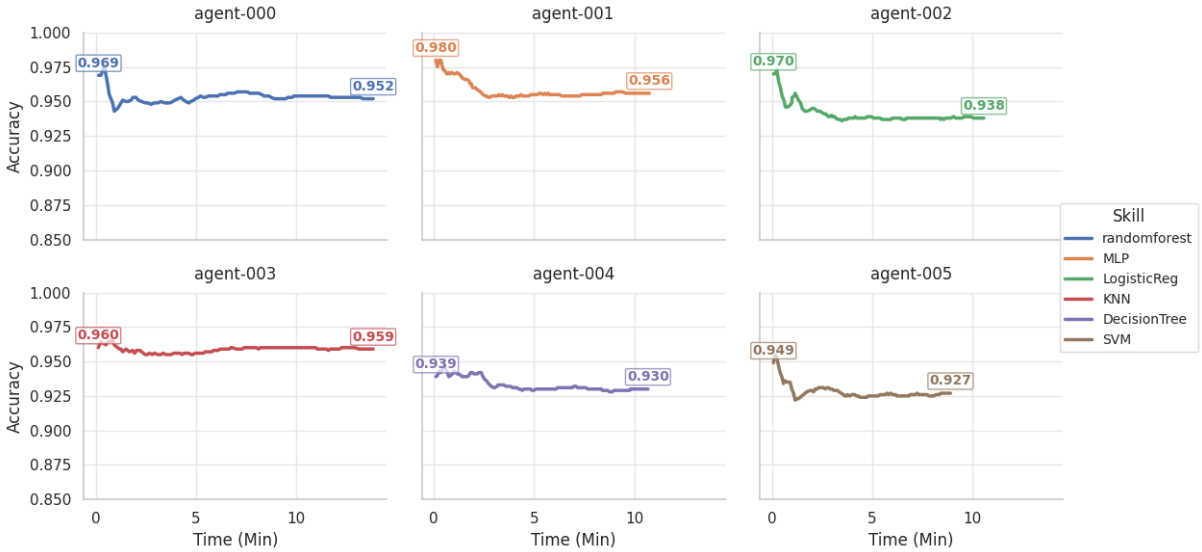

Figure 5.2: Accuracy - Collaborative Performance - Confidence at 70%.

Precision

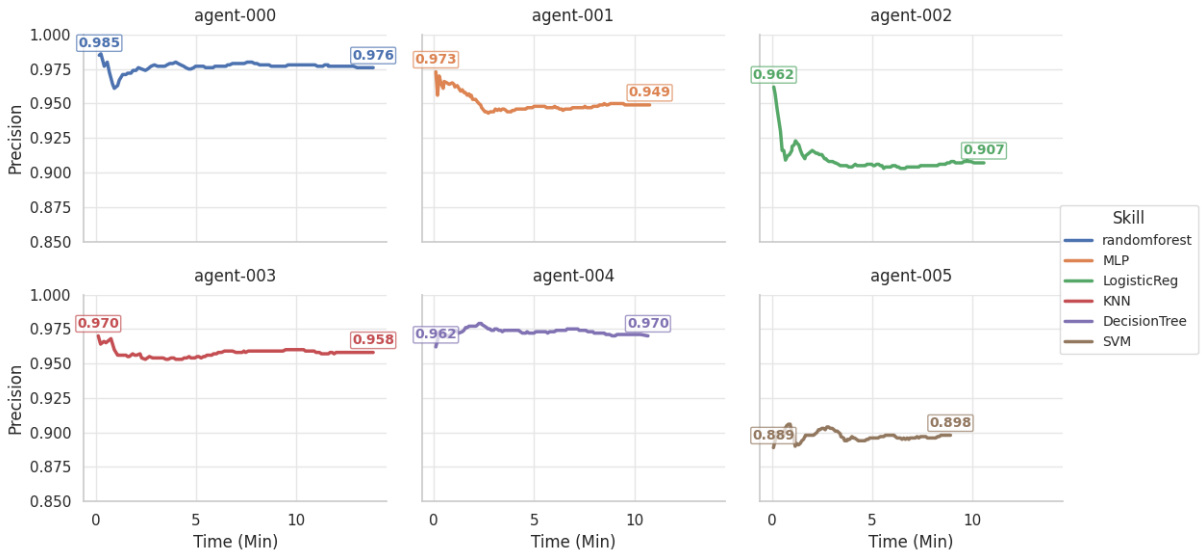

Figure 5.3: Precision - Individual Performance - Confidence at 70%.

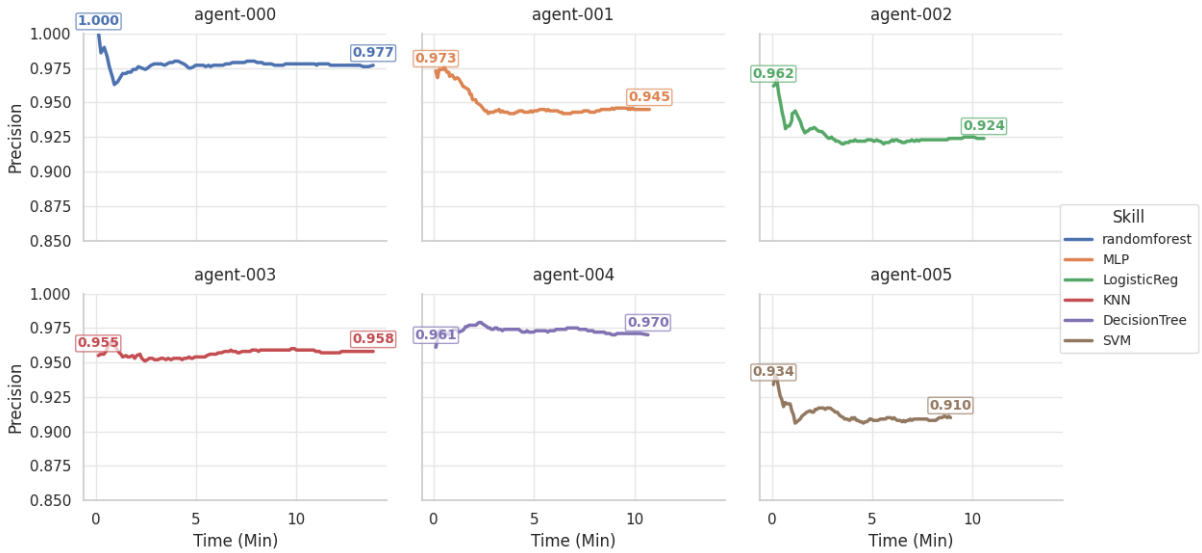

Figure 5.4: Precision - Collaborative Performance - Confidence at 70%.

From Figures 5.3 and 5.4, it is clear that only agents 000, 002, and 005 managed to improve their final precision performance through collaboration, comparing the final individual value with the final collaborative value. The others maintained their results, except for agent-001, which slightly reduced its result.

Recall

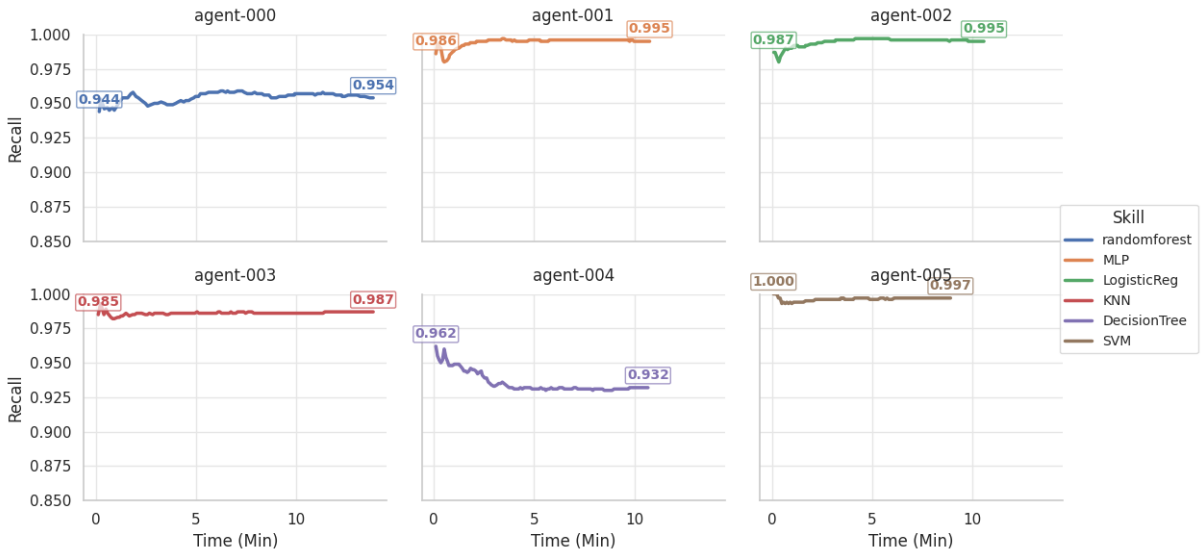

Figure 5.5: Recall - Individual Performance - Confidence at 70%.

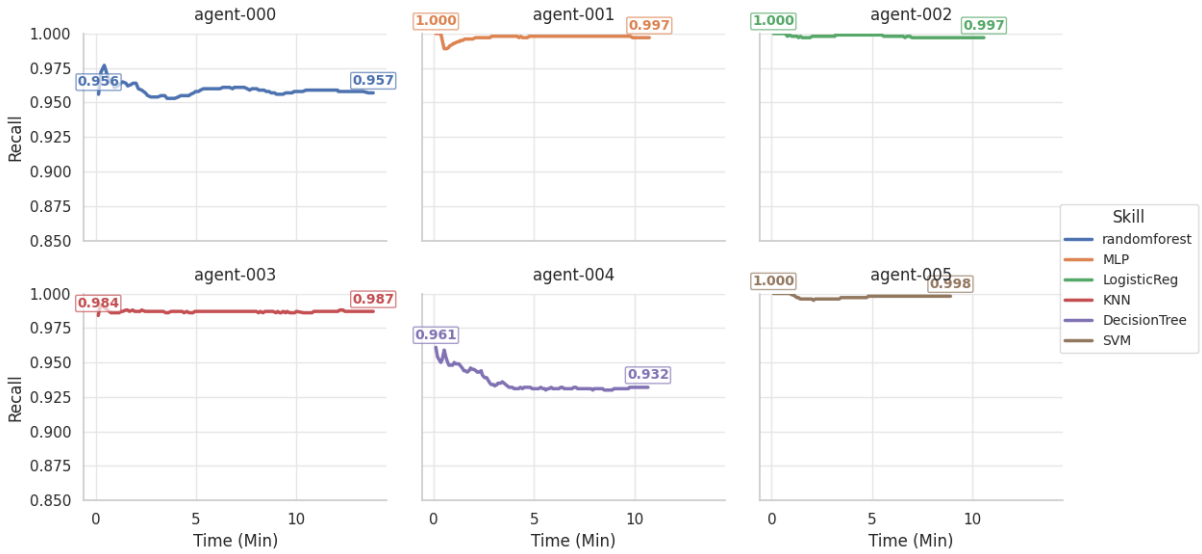

Figure 5.6: Recall - Collaborative Performance - Confidence at 70%.

With Figures 5.5 and 5.6, it is possible to observe that all agents obtained a better final collaborative result than the individual one, except agent 004.

F1-score

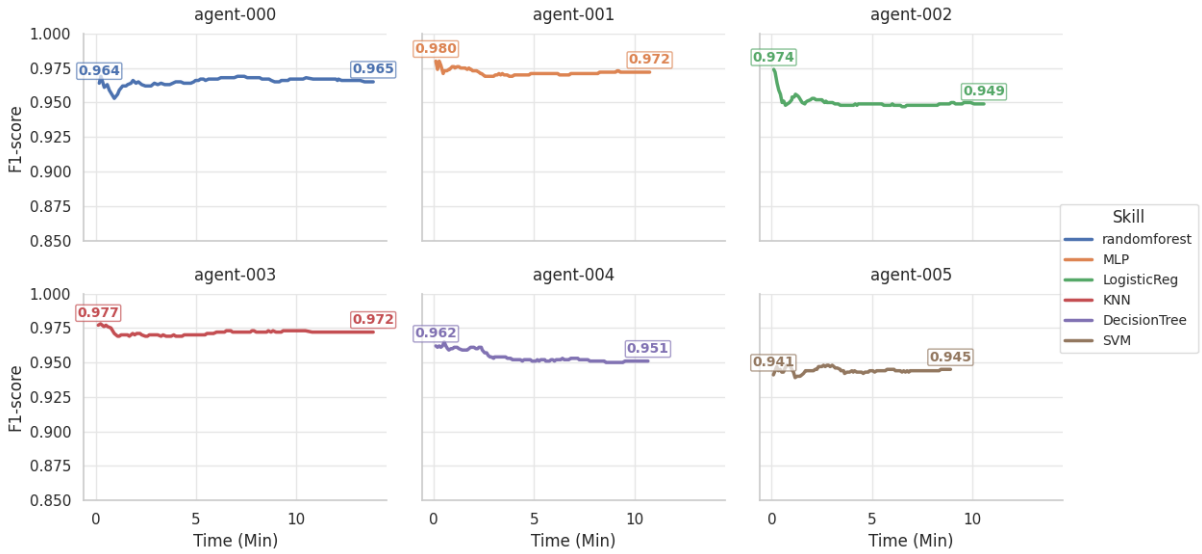

Figure 5.7: F1 score - Individual Performance - Confidence at 70%.

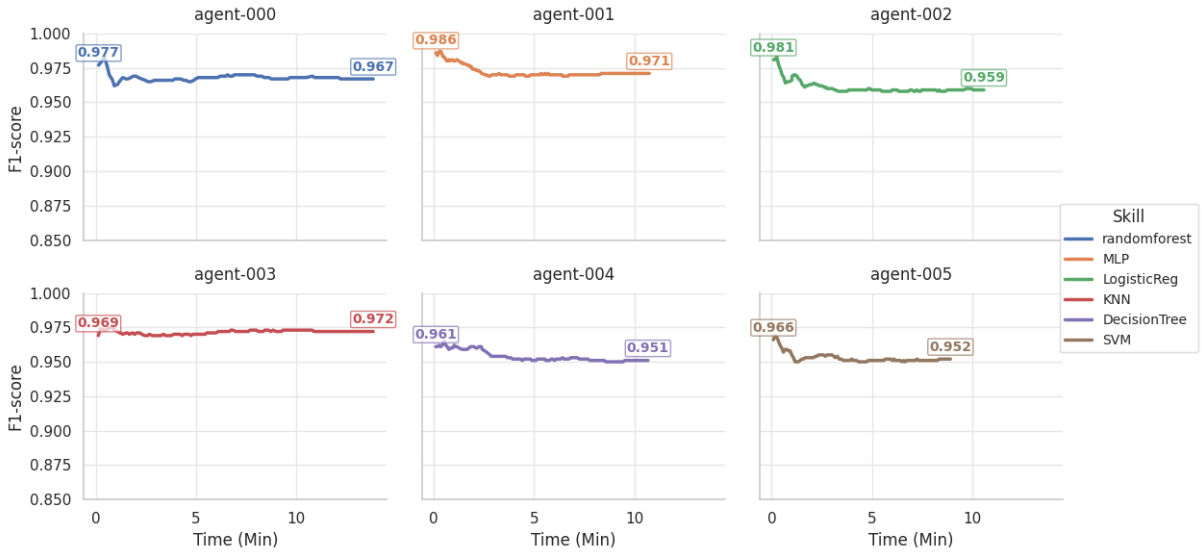

Figure 5.8: F1 score - Collaborative Performance - Confidence at 70%.

Comparing the graphs in Figures 5.7 and 5.8, it can be seen that agents 000, 002, and 005 managed to improve their final results. On the other hand, agent 001 lost some F1-score in collaboration. Agents 003 and 004 maintained their performances.

5.2.2 Confidence at 80%

Accuracy

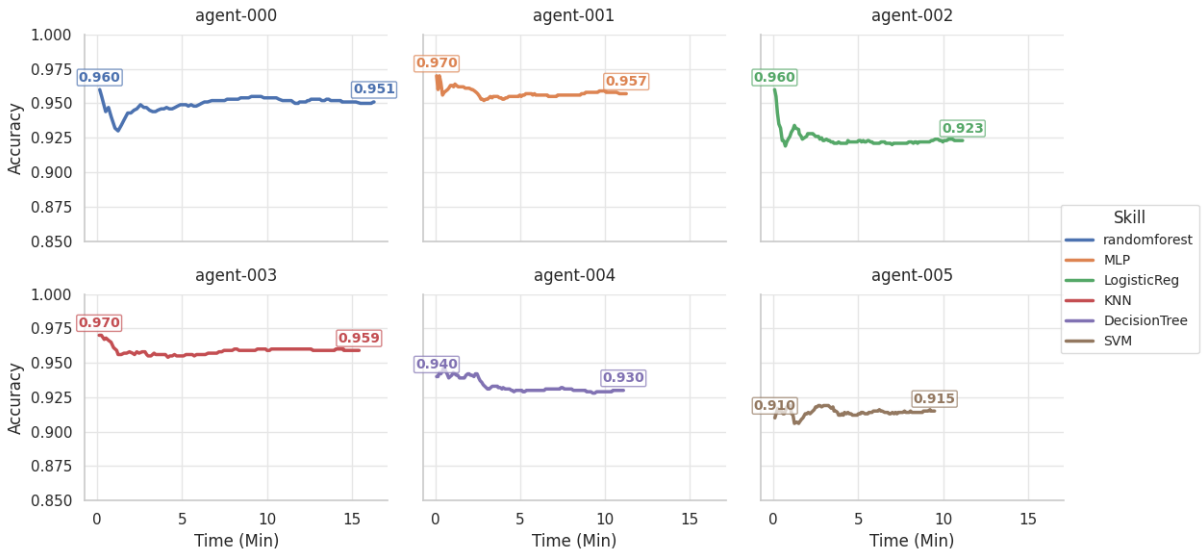

Figure 5.9: Accuracy - Individual Performance - Confidence at 80%.

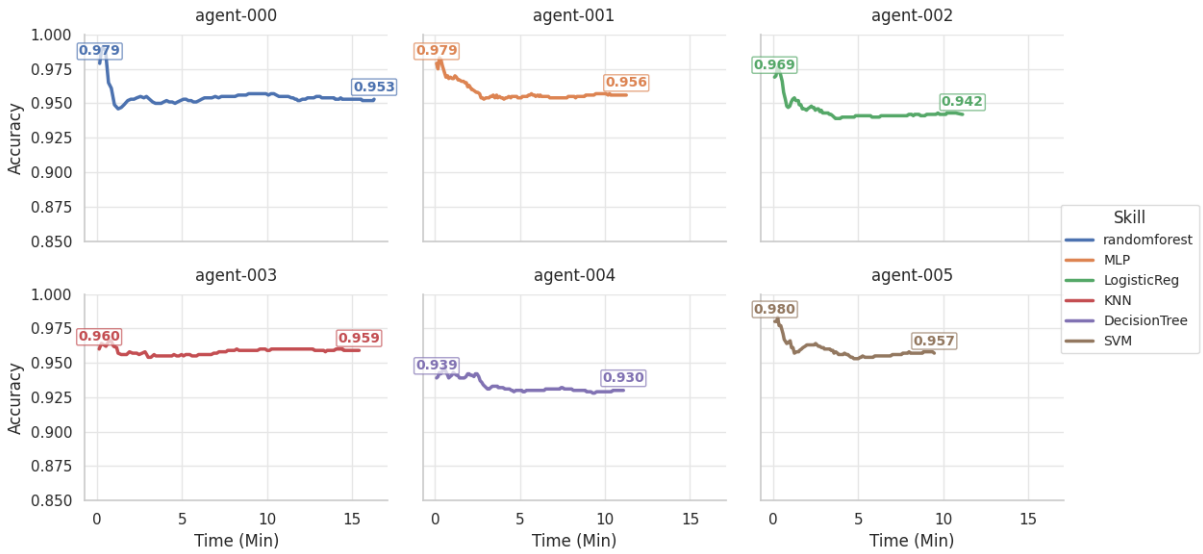

Figure 5.10: Accuracy - Collaborative Performance - Confidence at 80%.

With Figures 5.9 and 5.10 it is possible to observe that agents 000, 002 and 005 managed to improve their final value performance with the collaboration, unlike the others who lost

a little performance or remained with the same value.

Precision

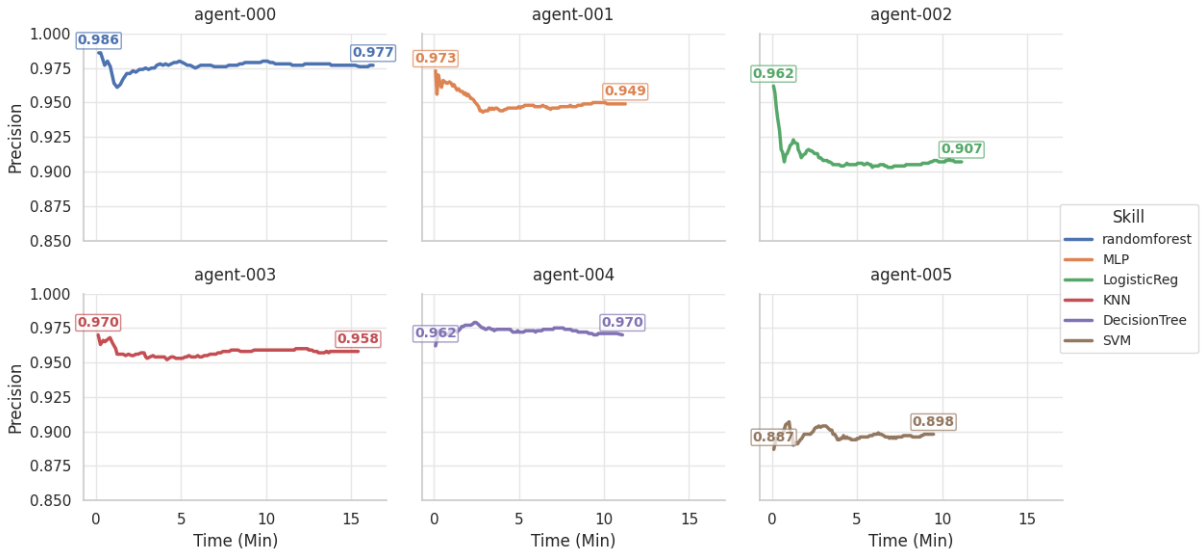

Figure 5.11: Precision - Individual Performance - Confidence at 80%.

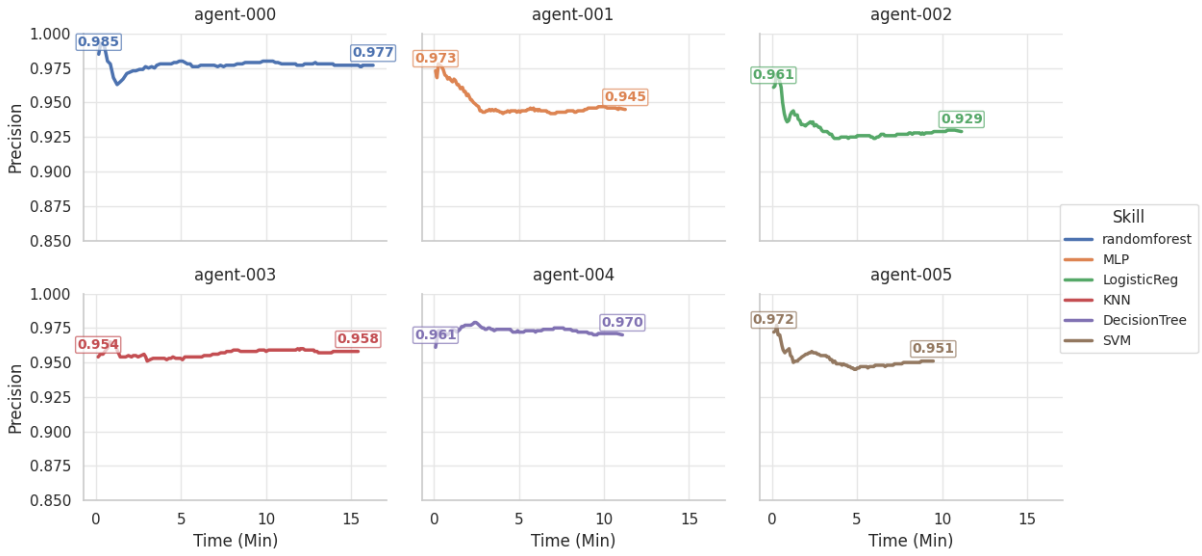

Figure 5.12: Precision - Collaborative Performance - Confidence at 80%.

From Figures 5.11 e 5.12, it can be seen that only agents 002 and 005 were able to benefit from the collaboration, unlike agent 001, who lost some of his precision when compared to

the final individual performance value. The others maintained the same results.

Recall

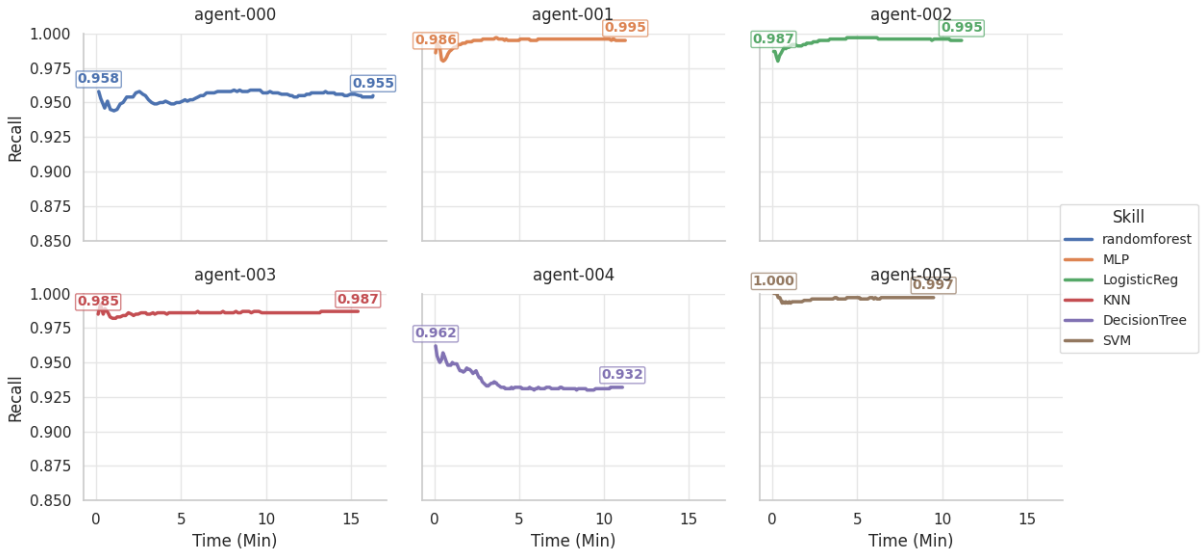

Figure 5.13: Precision - Individual Performance - Confidence at 80%.

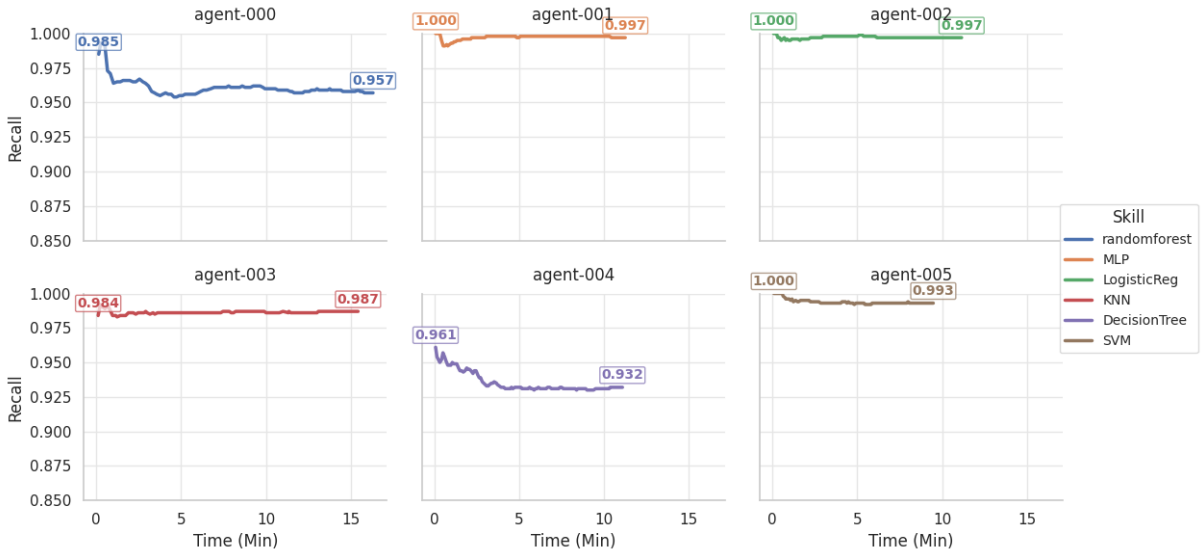

Figure 5.14: Recall - Collaborative Performance - Confidence at 80%.

Comparing the graphs in Figures 5.13 and 5.14, it is clear that when collaborating with others, only agents 000, 001, and 002 improved their final recall when compared to the

final individual result. On the other hand, agent 005 got a little worse, while agents 000 and 004 maintained their results.

F1-score

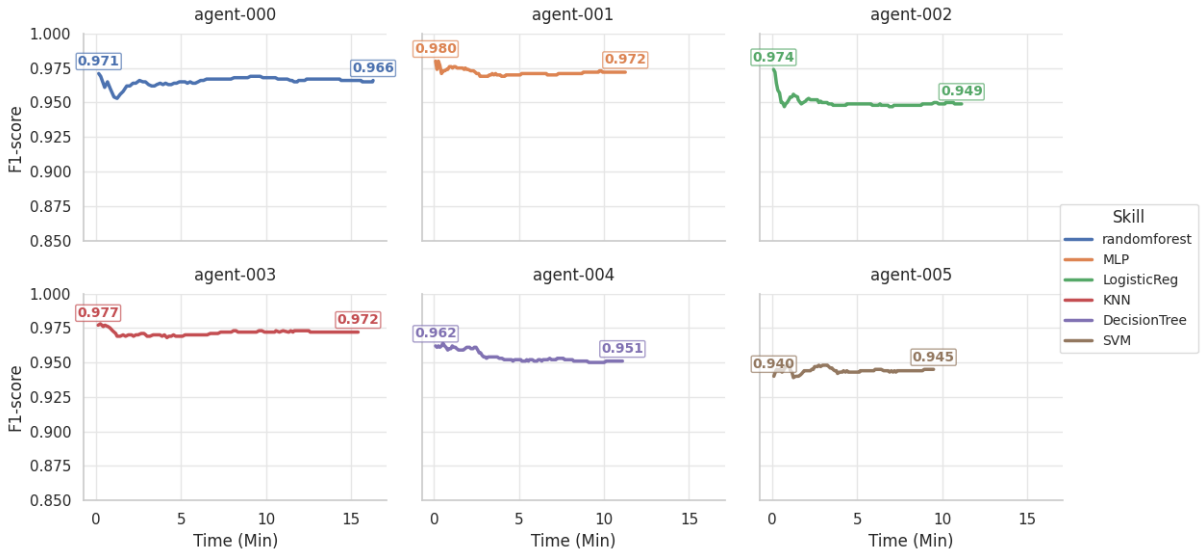

Figure 5.15: F1 score - Individual Performance - Confidence at 80%.

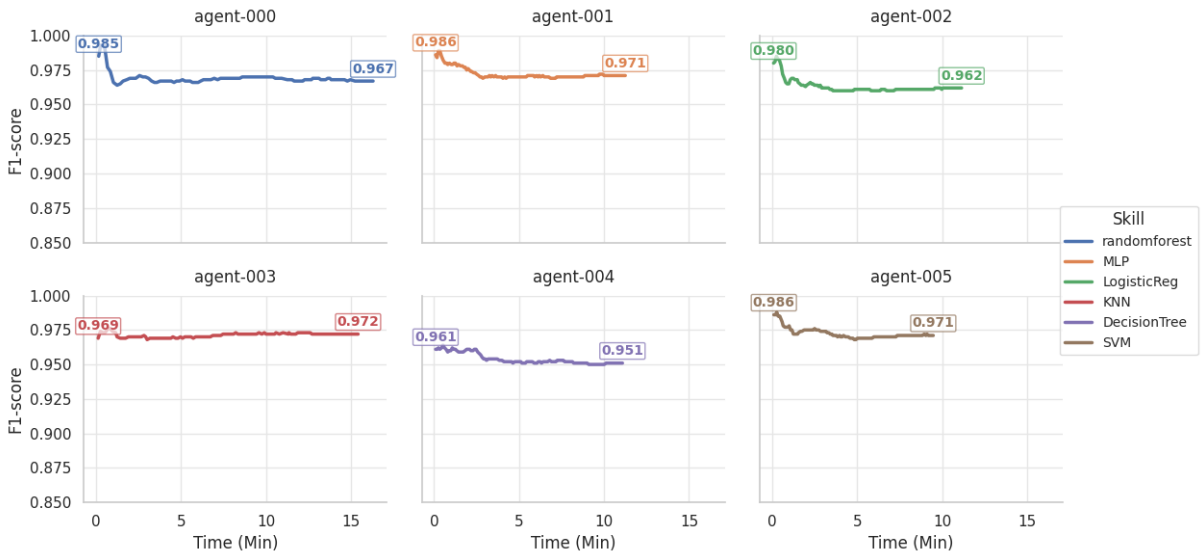

Figure 5.16: F1 score - Collaborative Performance - Confidence at 80%.

With the graphs in Figures 5.15 and 5.16, it can be seen that agents 000, 002, and 005 obtained improvements when they collaborated. On the other hand, agent 001 worsened slightly. Agents 003 and 004 maintained the same result.

5.2.3 Confidence at 90%

Accuracy

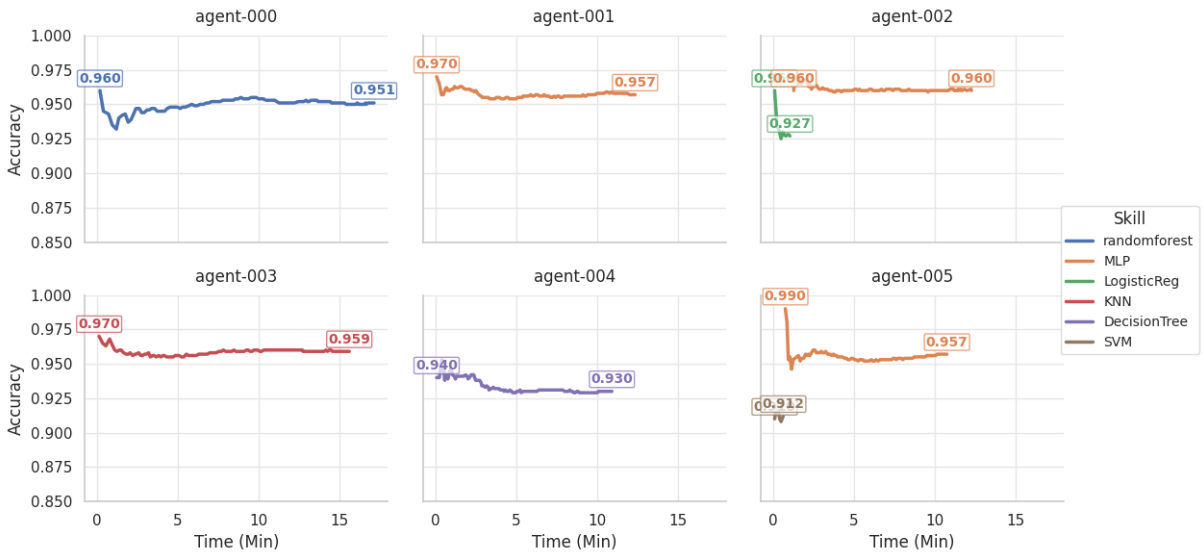

Figure 5.17: Accuracy - Individual Performance - Confidence at 90%.

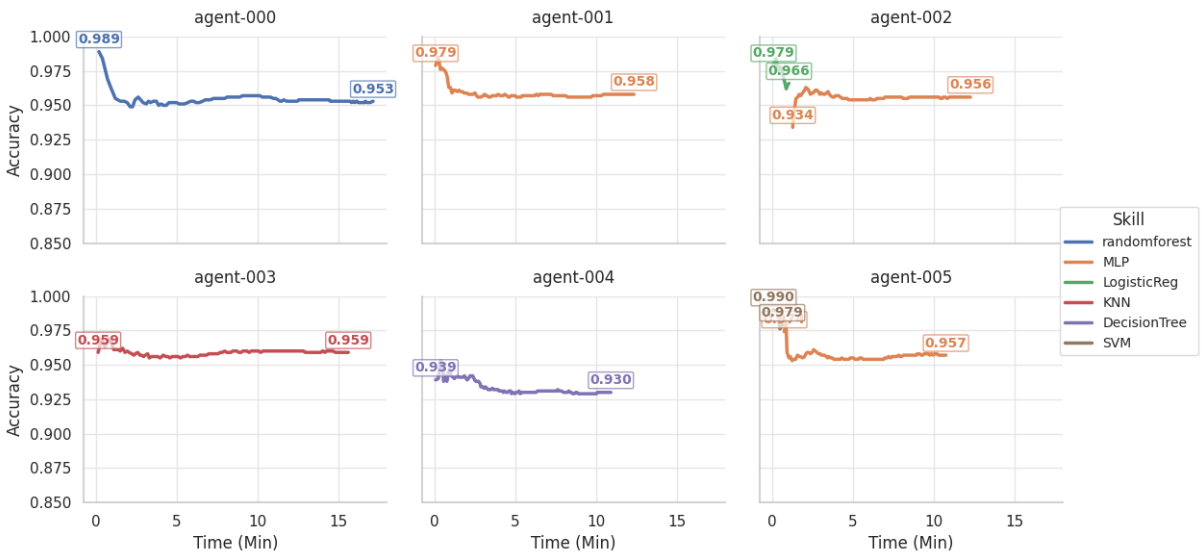

Figure 5.18: Accuracy - Collaborative Performance - Confidence at 90%.

From the graphs in figures 5.17 and 5.18, it is possible to observe that agents 003, 004, and 005 maintained their performance in collaborative mode. On the other hand, agent 002's performance decreased. Agents 000 and 001 managed to improve their final results with collaboration.

Precision

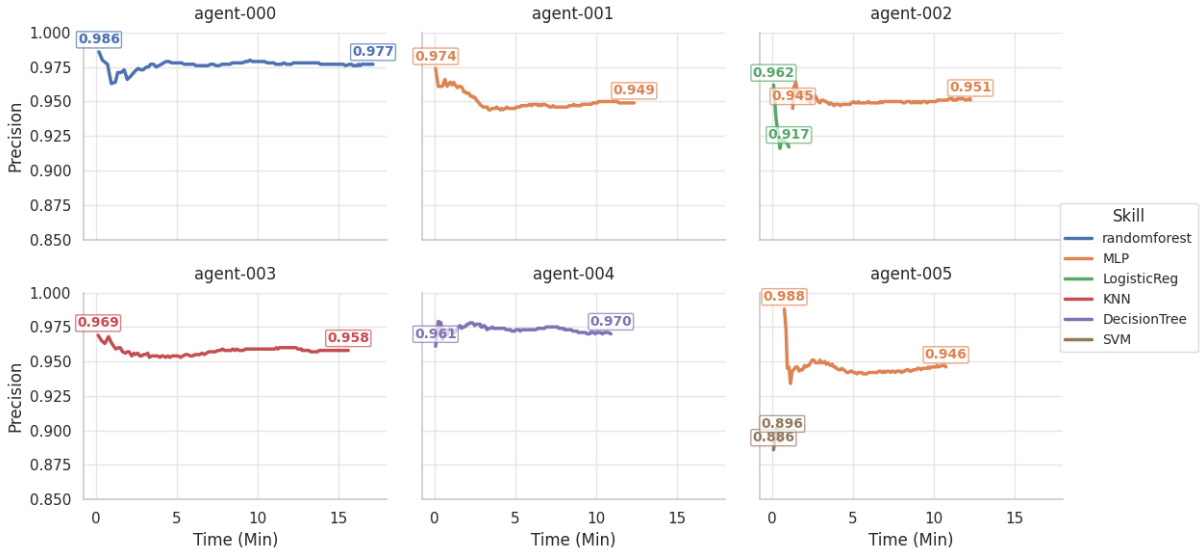

Figure 5.19: Precision - Individual Performance - Confidence at 90%.

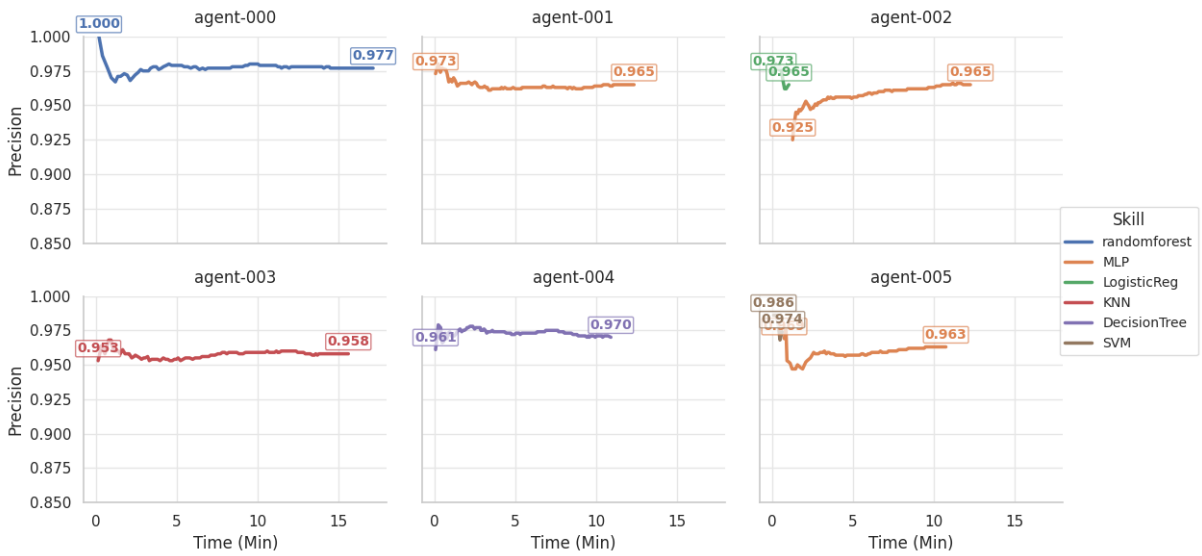

Figure 5.20: Precision - Collaborative Performance - Confidence at 90%.

Observing Figures 5.19 and 5.20, it is clear that with collaboration, agents 001, 002, and 005 improved their precision as a result of collaboration, however, agents 000, 003, and 004 remained with the same final value.

Recall

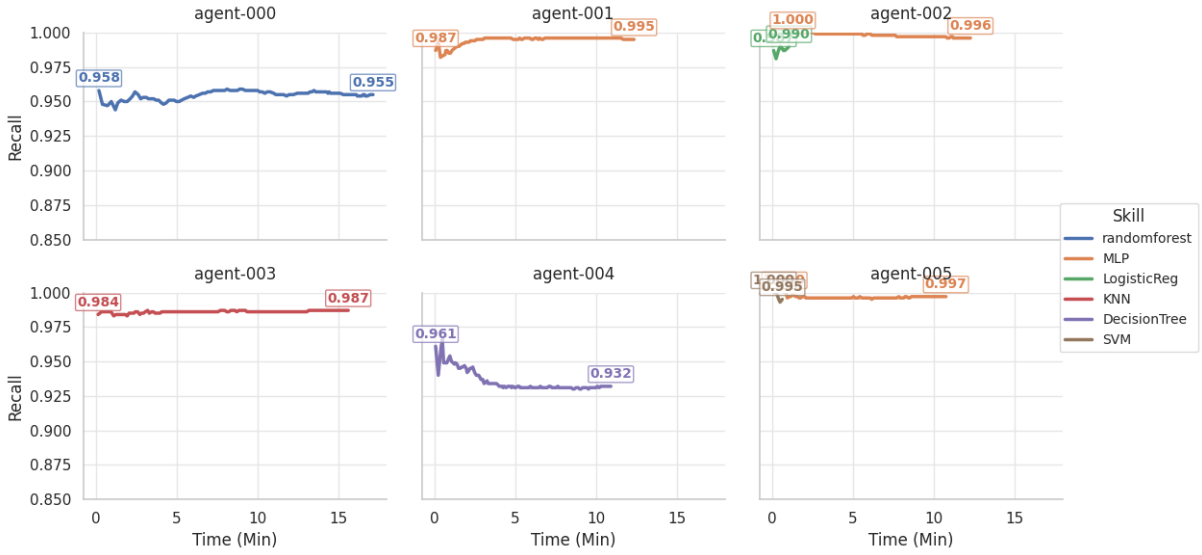

Figure 5.21: Recall - Individual Performance - Confidence at 90%.

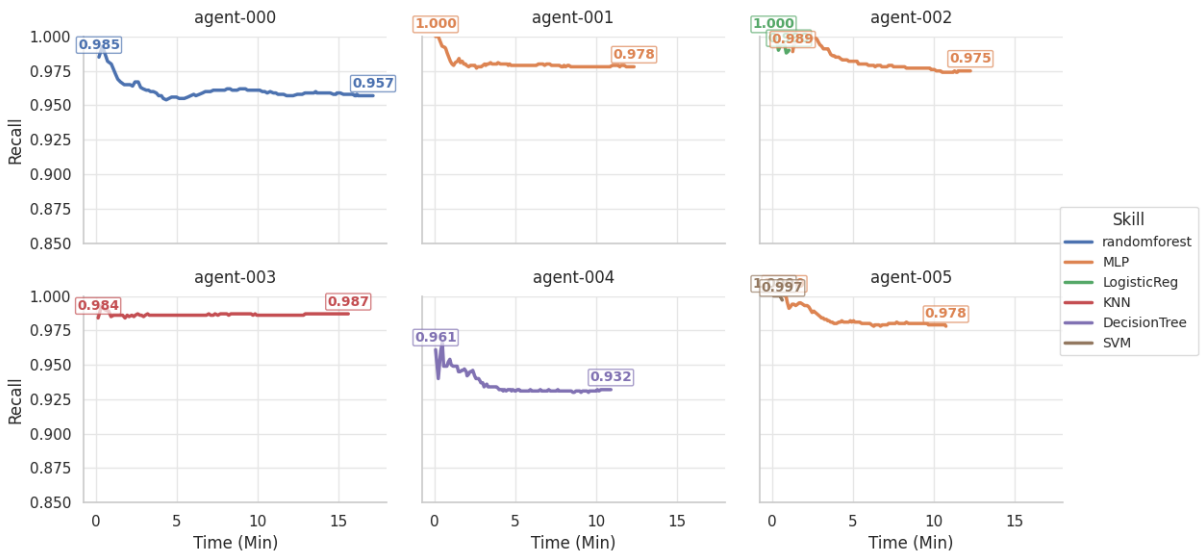

Figure 5.22: Recall - Collaborative Performance - Confidence at 90%.

From the graphs in Figures 5.21 and 5.22, it can be seen that only agent 000 improved its final result with the collaboration. However, agents 003 and 004 maintained their results. The other agents slightly decreased their recall.

F1-score

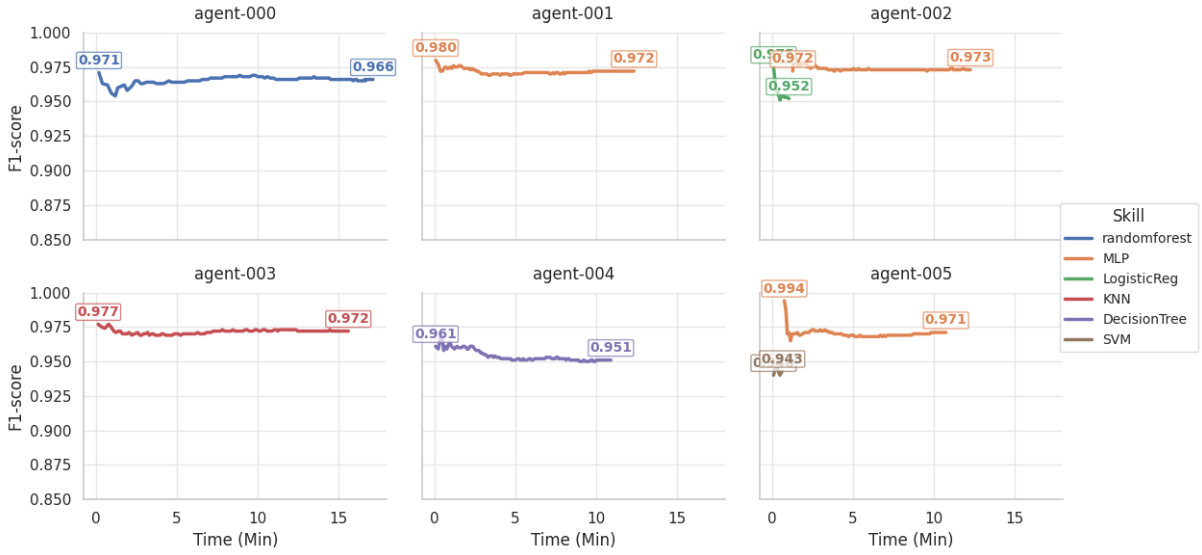

Figure 5.23: F1 score - Individual Performance - Confidence at 90%.

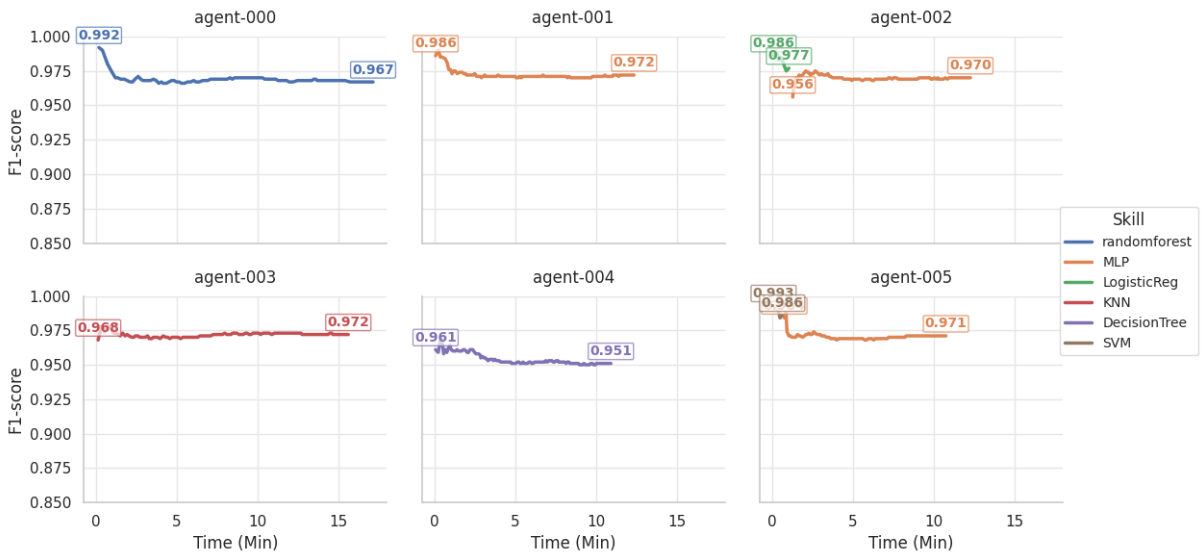

Figure 5.24: F1 score - Collaborative Performance - Confidence at 90%.

Through the graphs in the Figures 5.23 and 5.24, it is possible to observe that only agent 000 improved its final results with the collaboration. On the other hand, agent 002 decreased its final result. However, the others maintained their results.

In summary, at a confidence level of 70%, agents 000 (RF), 002 (LR) and 005 (SVM) showed a consistent improvement in all the metrics evaluated, demonstrating robust performance. The increase in accuracy implies a higher proportion of correct classifications, indicating that collaboration has made the agent more effective at discriminating between different classes. In addition, the improved accuracy shows that, of the instances classified as positive, a greater proportion actually correspond to the class of interest, thus reducing the incidence of false positives. In addition, the high recall demonstrates the model's improved ability to identify all positive instances, which is crucial for minimizing false negatives. The improvement of the F1-score, which integrates precision and recall by means of its harmonic mean, shows that a more robust balance has been achieved between these essential metrics.

At the 80% confidence level, agent 001 (MLP) showed a reduction in accuracy and precision, indicating a worsening in the model's ability to correctly classify instances. Although recall improved, the drop in F1-score shows that the balance between precision

and recall was not maintained, suggesting a high incidence of false positives and/or false negatives. Agent 002 (LR) showed a consistent improvement in all the metrics evaluated, indicating robust performance. The increase in accuracy implies a higher proportion of correct classifications, indicating that collaboration has made the agent more effective at discriminating between the different classes. In addition, the improved accuracy shows that, among the instances classified as positive, a greater proportion actually corresponds to the class of interest, thus reducing the incidence of false positives. In addition, the high recall demonstrates the model's improved ability to identify all positive instances, which is crucial for minimizing false negatives. The increase in the F1 value, which measures the balance between precision and recall, demonstrates a more consistent balance between these two fundamental metrics. Agent 005 (SVM) showed mixed performance: there was an increase in accuracy and precision, which indicates that the number of correct predictions increased, and false positives were reduced. However, recall worsened, indicating that the model failed to identify some positive instances, which may have led to an increase in false negatives. Interestingly, the F1 score improved, suggesting that the gain in precision was robust enough to compensate for the drop in the ability to identify positives.

At confidence level of 90%, Agent 001(MLP), on the one hand, improved accuracy and precision, indicating an increase in the number of correct classifications and a reduction in false positives. On the other hand, the drop in recall suggests that there was a decrease in the model's ability to correctly identify all positive instances, implying an increase in false negatives. Interestingly, the F1-score remained unchanged, which can be interpreted as a compensatory balance between the gains in precision and the loss in recall. Agent 002 (LR) despite the improvement in precision, the algorithm showed a worsening in accuracy, recall, and F1-score, indicating that although the reduction in false positives is good, the ability to correctly identify positive instances has decreased, compromising the overall performance of the model and suggesting the need for adjustments to achieve a more effective balance between the metrics. Although the accuracy of agent 005 (SVM) remained unchanged, the improvement in precision indicates a reduction in false positives, while the worsening in recall shows a reduction in the correct identification of positive instances.

Maintaining the F1-score suggests that the increase in precision evenly compensated for the drop in recall, highlighting the challenge of improving the model's sensitivity without compromising its accuracy.

Agent 000 (RF), at confidence levels 80 and 90, showed notable improvements in its performance: there was an increase in accuracy, showing a greater number of correct predictions, and recall improved, indicating an increased ability to correctly identify positive instances. Although precision remained constant, the balance between precision and recall resulted in a higher F1 score, which reinforces the model's effectiveness in detecting positive cases without increasing the false positive rate. These findings suggest that the modifications implemented made a significant contribution to the system's robustness, opening up prospects for future research that could further improve performance, especially in terms of maintaining the quality of the predictions.

For all confidence levels, the agents 003 (KNN) and 004 (DT) showed consistent performance, keeping all the metrics evaluated unchanged. This behavior suggests stability in classification, with accuracy, precision, recall, and F1-score indices remaining constant throughout the tests. This constancy can be interpreted as an indication of the model's robustness, although there is also a possible limitation of this model in terms of adaptability to variations in data and collaboration.

In general, different confidence levels generated different final performances, as demonstrated on Table 5.3.

Table 5.3: Improvements at Different Confidence Levels

Confidence Level	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Total Improvements**
70%	3	3	4	3	13
80%	3	2	3	3	11
90%	2	3	1	1	7

Finally, it can be seen that on confidence at 70%, 3 agents improved accuracy, 3 improved precision, 4 improved recall, and 3 improved F1-score. On Confidence level at 80%, 3 agents improved accuracy, 2 improved precision, 3 improved recall, and 3 improved

F1-score. At last on confidence of 90%, 2 agents improved accuracy, 3 improved precision, 1 improved recall, and 1 improved F1-score.

5.3 Additional metrics comparison

For comparison purposes, it is also interesting to analyze metrics such as the total number of items processed by each agent, the total number of points scored, the total number of questions asked, the number of algorithm changes, and the number of timeouts, i.e., the number of times the answer did not arrive within the predefined time during a vote. Other interesting points to be analyzed are: the average API response time, the average time between asking the question and receiving the answer, and voting time, which is the total time it took the agent to receive all the answers and make a decision.

Table 5.4: Confidence at 70%

Name (Skill)	Total Items	Points	Asks	Changes	Timeouts
agent-000 (RF)	10400.00	9354.0	657.00	0.0	623.0
agent-001 (MLP)	12000.00	10980.0	458.00	0.0	439.0
agent-002 (LR)	12000.00	10140.0	292.00	0.0	275.0
agent-003 (KNN)	11920.00	10942.0	277.00	0.0	265.0
agent-004 (DT)	12000.00	10320.0	0.00	0.0	0.0
agent-005 (SVM)	10358.00	8602.0	188.00	0.0	171.0
Total (Mean)	11446.33	10056.33	312.00	0.0	295.5

From Table 5.4, it is possible to observe that there was no change of algorithm by any agent. For this uncertainty interval configuration, the total number of items processed was very similar for most. The agent that scored the most points was 001 with the MLP algorithm, thus indicating greater accuracy. Agent 000 (RF) was the one that asked the most questions throughout the simulation, indicating a certain distrust, of course, considering the number of points it scored.

Table 5.5: Time metrics per agent - Confidence at 70%

Name (Skill)	API Time (Sec)				Response Waiting Time (Sec)				Voting Time (Sec)			
	Min	Max	Mean	Std	Min	Max	Mean	Std	Min	Max	Mean	Std
agent-000 (RF)	0.016	0.098	0.041	0.017	1.246	8.703	8.572	0.863	1.872	8.282	8.165	0.799
agent-001 (MLP)	0.001	0.040	0.012	0.006	0.045	4.244	0.577	0.764	1.740	7.628	7.452	0.934
agent-002 (LR)	0.001	0.033	0.011	0.006	0.051	3.104	0.460	0.492	0.758	8.007	7.815	1.051
agent-003 (KNN)	0.003	0.083	0.029	0.015	1.751	8.537	8.084	1.408	1.837	6.211	6.118	0.594
agent-004 (DT)	0.002	0.031	0.012	0.005	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
agent-005 (SVM)	0.001	0.028	0.012	0.005	0.066	4.728	0.603	0.928	1.518	7.248	7.062	0.966
Total (Mean)	0.004	0.052	0.020	0.009	0.527	4.886	3.633	0.743	1.2875	6.229	6.102	0.724

Table 5.5 shows the minimum and maximum values, the arithmetic mean of the times (Mean), and the standard deviation (Std). It is worth noting that since the average response time of the API did not exceed the stipulated value (0.1) for algorithm change, there were no changes in this way. The average time for the interval between sending a question and receiving the answer was high for agents 000 (RF) and 003 (KNN) compared to the total average, indicating that the other agents were a little slow to meet the demand. Finally, the voting time had an average of 6 seconds, where agents 000 (RF), 001 (MLP) and 002 (LR) exceeded the average by more than a second. It is worth noting that agent 003 (KNN) was the one that demonstrated the lowest standard deviation for voting time, indicating that it is the most consistent agent with the lowest dispersion in the observed values.

Table 5.6: Confidence at 80%

Name (Skill)	Total Items	Points	Asks	Changes	Timeouts
agent-000 (RF)	11100	10008	1068	0.0	1034
agent-001 (MLP)	12000	10980	688	0	664
agent-002 (LR)	12000	10140	1004	0	977
agent-003 (KNN)	12000	11012	398	0.0	390
agent-004 (DT)	12000	10320	0	0	0
agent-005 (SVM)	10358	8602	716	0	685
Total (Mean)	11576.33	10177.0	645.67	0.0	625.0

Table 5.6 shows that there was no change in the algorithm. In general, the algorithms scored well, specifically agent 003 (KNN), which indicates good accuracy and efficiency, taking into account the number of questions asked by it. The agents that asked the most questions were agents 000 (RF) and 002 (LR), which indicates that they faced more situations of uncertainty. Finally, the agent with the highest number of timeouts was agent 000 (RF), which means that the other agents took longer to respond, which may be due to the individual speed of each algorithm or even overload in information processing.

Table 5.7: Time metrics per agent - Confidence at 80%

Name (Skill)	**API Time (Sec)**				**Response Waiting Time (Sec)**				**Voting Time (Sec)**			
	Min	**Max**	**Mean**	**Std**	**Min**	**Max**	**Mean**	**Std**	**Min**	**Max**	**Mean**	**Std**
agent-000 (RF)	0.013	0.195	0.050	0.028	1.831	8.733	8.651	0.686	2.414	8.219	8.127	0.689
agent-001 (MLP)	0.007	0.040	0.018	0.006	0.074	5.562	0.570	0.661	1.896	8.295	8.121	0.972
agent-002 (LR)	0.006	0.038	0.018	0.006	0.049	3.315	0.581	0.532	0.000	8.401	8.220	1.105
agent-003 (KNN)	0.009	0.129	0.041	0.025	2.120	6.972	6.866	0.677	2.659	5.978	5.930	0.380
agent-004 (DT)	0.007	0.034	0.017	0.006	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
agent-005 (SVM)	0.007	0.048	0.017	0.006	0.076	3.384	0.624	0.646	2.083	7.633	7.477	0.888
Total (Mean)	0.008	0.081	0.027	0.013	0.692	4.327	2.549	0.534	1.842	6.088	6.312	0.672

Observing Table 5.7, it is clear that there was no change in the algorithm. Agents 000 (RF) and 003 (KNN) presented a response waiting time much higher than the average, indicating that the other agents were possibly slower in processing the information or that some type of overload may have occurred on their behalf. Finally, the agents had an average voting time of 6,312 seconds, with only agent 003 (KNN) being below this average.

Table 5.8: Confidence at 90%

Name (Skill)	Total Items	Points	Asks	Changes	Timeouts
agent-000 (RF)	11400.00	10274.0	1450.00	0.0	1405.0
agent-001 (MLP)	12000.00	10980.0	1030.00	0.0	1001.0
agent-002 (LR)	1000.00	854.0	264.00	0.0	151.0
agent-002 (MLP)	10901.00	10039.0	938.00	1.0	971.0
agent-003 (KNN)	12000.00	11012.0	788.00	0.0	766.0
agent-004 (DT)	12000.00	10320.0	0.00	0.0	0.0
agent-005 (MLP)	9830.00	8982.0	824.00	1.0	850.0
agent-005 (SVM)	501.00	413.0	115.00	0.0	19.0
Total (Mean)	10779.00	9297.0	676.13	0.25	645.25

Table 5.8 shows that there were two algorithm changes: agent 002 changed from the LR algorithm to the MLP, while agent 005 changed from the SVM to the MLP. Agents 000 (RF) and 001 (MLP) were the ones who asked the most questions. On the other hand, agent 003 (KNN) scored the highest, indicating greater accuracy, while also observing the total number of questions asked. Finally, the agents who had the highest number of timeouts were agents 000 (RF) and 001 (MLP). It is also possible to observe that a problem occurred with agents 002 and 005, both of whom switched to MLP, and the total number of timeouts was greater than the number of questions asked. A possible explanation is that when switching algorithm,s the system could have been overloaded, and these agents still had opinions to receive from other agents, and when they received these responses, the stipulated time to receive responses had already passed, so they counted more timeouts than questions.

Table 5.9: Time metrics per agent - Confidence at 90%

Name (Skill)	API Time (Sec)				Response Waiting Time (Sec)				Voting Time (Sec)			
	Min	**Max**	**Mean**	**Std**	**Min**	**Max**	**Mean**	**Std**	**Min**	**Max**	**Mean**	**Std**
agent-000 (RF)	0.019	0.147	0.054	0.025	2.237	8.369	8.297	0.605	2.993	8.116	8.059	0.497
agent-001 (MLP)	0.003	0.069	0.025	0.012	0.021	7.192	0.738	1.717	1.618	7.626	7.504	0.758
agent-002 (LR)	0.007	0.025	0.015	0.006	0.083	7.002	3.975	2.668	2.264	8.145	6.758	2.368
agent-002 (MLP)	0.002	0.055	0.026	0.012	0.023	8.795	1.384	2.964	0.000	0.000	0.000	0.000
agent-003 (KNN)	0.010	0.091	0.039	0.018	1.003	8.234	8.078	0.993	2.459	6.762	6.708	0.441
agent-004 (DT)	0.003	0.053	0.016	0.006	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
agent-005 (MLP)	0.006	0.065	0.027	0.013	0.015	7.444	1.055	2.334	0.000	0.000	0.000	0.000
agent-005 (SVM)	0.007	0.028	0.018	0.008	0.446	3.204	1.952	1.035	1.993	7.693	5.478	2.400
Total (Mean)	0.007	0.067	0.028	0.013	0.478	6.280	3.185	1.540	1.416	4.793	4.313	0.808

Table 5.9 presents the temporal metrics, and from them, it is possible to see that agents 000 (RF) and 003 (KNN) had a time interval between sending the question and receiving the answer that was well above the average, thus indicating that the other agents were possibly overloaded to answer them. However, the standard deviations for most agents in terms of sending the question and receiving the answer were considerably high when compared to their individual average, thus indicating significant variability in this time metric. Finally, the voting times for decision-making indicate that all agents were above the average, taking up to 8 seconds to receive all the answers and make the decision. Consequently, it is also possible to observe that agents 005 (SMV) and 002 (LR) had a high standard deviation for the final voting time, indicating that the average voting time had a significant fluctuation in values over time. It is clear that agents 002 and 005 switched to the MLP algorithm, but the voting times appear to be zero, which can be explained by a possible task overload on the agents. An overload on one or more agents prevented the communication that existed during the votes from being carried out completely since the processes are concurrent and each algorithm has a speed of action. Whence, these agents were unable to record the voting times.

- Confidence at 70%: There was no algorithm change by any agent and the average number of questions and timeouts was very low.

- Confidence at 80%: There was no algorithm change by any agent. The number of questions and timeouts doubled compared to the previous interval.
- Confidence at 90%: There were 2 algorithm changes by the agents. The number of questions and timeouts did not increase as much compared to the previous interval.

An important consideration that was only observed after the tests was that the DT algorithm only returns full answers, as if it were 100% certain, leaving no room for doubt. Consequently, it did not ask any questions and did not record any timeouts. Another point is that at the end of the simulation, with a confidence level of 90%, the agents that changed algorithms to MLP were unable to record the total voting time, as explained above, due to some type of failure in the execution of the simulation. Therefore, it is thought that the response and voting times would be slightly higher compared confidence level of 80% if the simulation had occurred under perfect conditions.

Ultimately, Table 5.10 demonstrates the percentage of evolution between the final results obtained collaboratively in relation to the individual results.

Table 5.10: Performance metrics for various confidence levels

Confidence Level	Accuracy	Precision	Recall	F1-score
70%	0.51%	0.48%	0.13%	0.32%
80%	1.13%	1.31%	0.04%	0.69%
90%	1.43%	2.36%	-0.66%	0.87%

Therefore, from the graphs and tables shown previously in the confidence level of 70%, accuracy improved by an average of 0.51%, precision by 0.48%, recall by 0.13%, and F1-score by 0.32%. The average number of questions asked was 312, and the number of timeouts was 296. The average total time spent by each agent for the entire process of asking, receiving the answer and making a decision was 6.12 seconds.

In the confidence level of 80%, accuracy improved by an average of 1.13%, precision by 1.31%, recall by 0.04% and F1-score by 0.69%. The average number of questions asked was 646, and the number of timeouts was 625. The average total voting time was 6.33

seconds. Finally in a confidence level of 90%, accuracy improved by an average of 1.43%, precision by 2.36% and F1-score by 0.87%. However the recall decreased by -0.66%. The average number of questions was 676, and the number of timeouts was 645. The average total voting time was 4.34 seconds.

Accordingly, it is clear that the collaboration of the agents managed to improve the overall performance of the system in all the intervals tested, except for recall in the case of confidence at 90%. The confidence level at 80% was the most balanced when compared to the others in relation to all the metrics and also considering the problems that occurred in the confidence level at 90%.

Chapter 6

Conclusions

The increasing complexity of cyber threats, combined with the evolution of network technologies and infrastructures, has driven the search for more robust and adaptable security solutions. Traditionally, IDS are centralized, concentrating data processing and analysis at a single point in the network. However, as IT infrastructures grow in complexity and interconnectivity, centralization begins to show limitations, mainly in terms of scalability, latency, and resilience.

In this context, the present work demonstrates an approach to a decentralized IDS using MAS and ML technologies. After the theoretical study on IDS, it was decided to follow the line of application of an NIDS based on anomalies. For this, the UNSW-NB15 dataset was chosen because it is popular and presents a good amount of samples with diverse attacks, containing benign and malignant network traffic already organized and classified. After that, it was necessary to treat and pre-process the data to train the models, seeking the best performance for all metrics: accuracy, precision, recall, and F1-score, which were all above 0.9 for the different models chosen. After adding intelligence to the agents through ML algorithms, a continuous improvement mechanism complement was added, which changes the agent's algorithm if it presents undesirable behavior, and then tests were carried out to analyze the results.

To measure and analyze the final performance, the following metrics were used: comparison between uncertainty ranges, accuracy, precision, recall, and F1-score metrics, as

well as number of points (correct answers), number of questions asked, number of algorithm changes , and number of timeouts that occurred. Other temporal metrics were added to observe the agents' reaction over time, namely: average API response time, average response waiting time, and average voting time (total time between receiving all responses and making a decision).

Eventually the confidence level at 80% was the most balanced in relation to most metrics, such as average time waiting for a response of 2.55 seconds, accuracy improving by an average of 1.13%, precision by 1.31%, recall by 0.04% and F1-score by 0.69%. In the context of cybersecurity, these are considerable improvements, since a system must operate with a very narrow margin of error, where each percentage point has an important impact on risk mitigation and operational efficiency. Consequently, these improvements indicate that collaboration between the agents has made the system more robust by reducing false positives (better accuracy) and, at the same time, maintaining a good detection capacity.

Therefore, it is evident that the agents' collaboration enhanced the system's overall performance across all tested confidence levels, with the exception of recall at the 90% confidence level. The 80% confidence level demonstrated the most balanced results when compared to the others across all metrics, taking into account the issues encountered at the 90% confidence level, as discussed at the end of the previous session.

For future work, use a larger dataset with more samples for training and simulation, which will consequently generate a simulation with a longer time. This way, it is possible to have a broader view of the operation and better observe the system's dynamics. Add mechanisms to protect messages exchanged among agents, one of which could be a blockchain, for example, which uses encryption, decentralization, and consensus, creating a secure system that is very difficult to violate since once data is recorded in the blockchain, it becomes immutable, and a single user is not able to change the transaction records, since everything is recorded in a transparent and auditable way. Another interesting improvement would be to try to eliminate the use of the REST API and use ML models with Java, thus reducing the time of action and eliminating a possible item for a security breach. Another additional point would be to use other types of algorithms that use other

approaches, such as unsupervised learning. Finally, tests should be carried out in a real environment with several computers to verify the behavior of the system.

Bibliography

[1] M. Mijwil, O. Unogwu, Y. Filali, I. Bala, and H. Al-Shahwani, "Exploring the top five evolving threats in cybersecurity: An in-depth overview," Mar. 2023. DOI: 10.58496/MJCS/2023/010.

[2] A. Fleck. "Expected cost of cybercrime until 2027." Online; accessed 11-Oct-2024. (2023), [Online]. Available: https://www.statista.com/chart/28878/expected-cost-of-cybercrime-until-2027/.

[3] F. Louati, F. B. Ktata, and I. Amous, "Big-ids: A decentralized multi-agent reinforcement learning approach for distributed intrusion detection in big data networks," *Cluster Computing*, vol. 27, no. 5, pp. 6823–6841, 2024. DOI: 10.1007/s10586-024-04306-9. [Online]. Available: https://doi.org/10.1007/s10586-024-04306-9.

[4] G. D. Putra, V. Dedeoglu, S. S. Kanhere, and R. Jurdak, "Poster abstract: Towards scalable and trustworthy decentralized collaborative intrusion detection system for iot," in *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*, 2020, pp. 256–257. DOI: 10.1109/IoTDI49375.2020.00035.

[5] S. Jose, D. Malathi, and B. R. et al., "A survey on anomaly based host intrusion detection system," *J. Phys.: Conf. Ser.*, vol. 1000, p. 012 049, 2018. DOI: 10.1088/1742-6596/1000/1/012049.

[6] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," *IEEE Access*, vol. 6, pp. 28 573–28 593, 2018. DOI: 10.1109/ACCESS.2018.2831228.

[7] I. H. Sarker, A. S. M. Kayes, S. Badsha, *et al.*, "Cybersecurity data science: An overview from machine learning perspective," *Journal of Big Data*, vol. 7, no. 41, 2020. DOI: 10.1186/s40537-020-00318-5. [Online]. Available: https://doi.org/10.1186/s40537-020-00318-5.

[8] F. Schiliro, "Towards a contemporary definition of cybersecurity," *arXiv preprint arXiv:2302.02274*, 2023.

[9] G. Boss, P. Malladi, D. Quan, L. Legregni, and H. Hall, "Cloud computing," *IBM white paper*, vol. 321, pp. 224–231, 2007.

[10] R. Beri and V. Behal, "Cloud computing: A survey on cloud computing," *International journal of computer applications*, vol. 111, no. 16, 2015.

[11] P. De Geus and E. Nakamura, *Segurança de Redes em Ambientes Cooperativos*. Aug. 2007, ISBN: 978-85-7522-136-5.

[12] A. Adnan, A. Muhammed, A. a. abdul ghani, A. Abdullah, and H. Fahrul, "An intrusion detection system for the internet of things based on machine learning: Review and challenges," *Symmetry*, vol. 13, p. 1011, Jun. 2021. DOI: 10.3390/sym13061011.

[13] A. Khraisat, I. Gondal, P. Vamplew, *et al.*, "Survey of intrusion detection systems: Techniques, datasets and challenges," *Cybersecurity*, vol. 2, p. 20, 2019. DOI: 10.1186/s42400-019-0038-7. [Online]. Available: https://doi.org/10.1186/s42400-019-0038-7.

[14] S. A. Bini, "Artificial intelligence, machine learning, deep learning, and cognitive computing: What do these terms mean and how will they impact health care?" *The Journal of Arthroplasty*, vol. 33, no. 8, pp. 2358–2361, 2018, ISSN: 0883-5403. DOI: 10.1016/j.arth.2018.02.067. [Online]. Available: https://doi.org/10.1016/j.arth.2018.02.067.

[15] T. O. Ayodele, "Types of machine learning algorithms," in *New Advances in Machine Learning*, Y. Zhang, Ed., Rijeka: IntechOpen, 2010, ch. 3. DOI: 10.5772/9385. [Online]. Available: https://doi.org/10.5772/9385.

[16] T. Mitchell, *Machine Learning* (McGraw-Hill International Editions). McGraw-Hill, 1997, pp. 52–53, ISBN: 9780071154673. [Online]. Available: https://books.google.pt/books?id=EoYBngEACAAJ.

[17] G. Li. "Do a data science project in 10 days." [Online; accessed 27-Apr-2024]. (), [Online]. Available: https://bookdown.org/gmli64/do_a_data_science_project_in_10_days/prediction-with-decision-trees.html.

[18] L. Rokach and O. Maimon, "Decision trees," in *Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach, Eds. Boston, MA: Springer US, 2005, pp. 165–192, ISBN: 978-0-387-25465-4. DOI: 10.1007/0-387-25465-X_9. [Online]. Available: https://doi.org/10.1007/0-387-25465-X_9.

[19] B. Gupta, A. Rawat, A. Jain, A. Arora, and N. Dhami, "Analysis of various decision tree algorithms for classification in data mining," *International Journal of Computer Applications*, vol. 163, no. 8, pp. 15–19, 2017.

[20] X. Wu, V. Kumar, J. R. Quinlan, *et al.*, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, no. 1, pp. 1–37, 2008. DOI: 10.1007/s10115-007-0114-2. [Online]. Available: https://doi.org/10.1007/s10115-007-0114-2.

[21] H. Liu and B. Lang, "Machine learning and deep learning methods for intrusion detection systems: A survey," *Applied Sciences*, vol. 9, no. 20, 2019, ISSN: 2076-3417. DOI: 10.3390/app9204396. [Online]. Available: https://www.mdpi.com/2076-3417/9/20/4396.

[22] J.-C. Chouinard. "How to use k-nearest neighbors (knn) with python (scikit-learn example)." [Online; accessed 27-Apr-2024]. (2023), [Online]. Available: https://www.jcchouinard.com/k-nearest-neighbors/.

[23] M. Brannick. "Logistic regression." Online; accessed 27-Apr-2024. (2024), [Online]. Available: http://faculty.cas.usf.edu/mbrannick/regression/Logistic.html.

[24] Voxco. "Linear regression vs logistic regression: Difference and working." (), [Online]. Available: https://www.voxco.com/blog/linear-regression-vs-logistic-regression-difference-and-working/ (visited on 04/27/2024).

[25] G. Mountrakis, J. Im, and C. Ogole, "Support vector machines in remote sensing: A review," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 66, no. 3, pp. 247–259, 2011, ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2010.11.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271610001140.

[26] M. Somvanshi, P. Chavan, S. Tambade, and S. V. Shinde, "A review of machine learning techniques using decision tree and support vector machine," in *2016 International Conference on Computing Communication Control and automation (ICCUBEA)*, 2016, pp. 1–7. DOI: 10.1109/ICCUBEA.2016.7860040.

[27] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd. New York: Springer, 2009, pp. 417–449.

[28] I. S. Al-Mejibli, J. K. Alwan, and D. H. Abd, "The effect of gamma value on support vector machine performance with different kernels," *International Journal of Electrical and Computer Engineering*, vol. 10, no. 5, pp. 5497–5506, 2020.

[29] R. Dastres and M. Soori, "Artificial neural network systems," *International Journal of Imaging and Robotics (IJIR)*, vol. 21, no. 2, pp. 13–25, 2021. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03349542.

[30] I. Basheer and M. Hajmeer, "Artificial neural networks: Fundamentals, computing, design, and application," *Journal of Microbiological Methods*, vol. 43, no. 1, pp. 3–31, 2000, Neural Computting in Micrbiology, ISSN: 0167-7012. DOI: https://

doi.org/10.1016/S0167-7012(00)00201-3. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167701200002013.

[31] M. Shanbehzadeh, H. Kazemi-Arpanahi, A. Orooji, S. Mobarak, and S. Jelvay, "Performance evaluation of selected machine learning algorithms for covid-19 prediction using routine clinical data: With versus without ct scan features," *Journal of Education and Health Promotion*, vol. 10, p. 285, Aug. 2021. DOI: 10.4103/jehp.jehp_1424_20.

[32] N. AlDahoul, A. N. Ahmed, M. F. Allawi, *et al.*, "A comparison of machine learning models for suspended sediment load classification," *Engineering Applications of Computational Fluid Mechanics*, vol. 16, no. 1, pp. 1211–1232, 2022. DOI: 10.1080/19942060.2022.2073565.

[33] T. Thangaraj, S. S, C. Chelliah, T. Chung, and A. Khan, "Performance analysis of machine learning algorithms in intrusion detection system: A review," *Procedia Computer Science*, vol. 171, pp. 1251–1260, Jan. 2020. DOI: 10.1016/j.procs.2020.04.133.

[34] A. Verikas, A. Gelzinis, and M. Bacauskiene, "Mining data with random forests: A survey and results of new tests," *Pattern Recognition*, vol. 44, no. 2, pp. 330–349, 2011, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2010.08.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320310003973.

[35] E. Costa, A. Lorena, A. Carvalho, and A. Freitas, "A review of performance evaluation measures for hierarchical classifiers," *AAAI Workshop - Technical Report*, Jan. 2007.

[36] Y. Reich and S. Barai, "Evaluating machine learning models for engineering problems," *Artificial Intelligence in Engineering*, vol. 13, no. 3, pp. 257–272, 1999, ISSN: 0954-1810. DOI: https://doi.org/10.1016/S0954-1810(98)00021-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0954181098000211.

[37] T. Saranya, S. Sridevi, C. Deisy, T. D. Chung, and M. Khan, "Performance analysis of machine learning algorithms in intrusion detection system: A review," *Procedia Computer Science*, vol. 171, pp. 1251–1260, 2020, Third International Conference on Computing and Network Communications (CoCoNet'19), ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2020.04.133. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050920311121.

[38] M. Sokolova, N. Japkowicz, and S. Szpakowicz, "Beyond accuracy, f-score and roc: A family of discriminant measures for performance evaluation," in *AI 2006: Advances in Artificial Intelligence*, A. Sattar and B.-h. Kang, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1015–1021, ISBN: 978-3-540-49788-2.

[39] M. A. El Mrabet, K. El Makkaoui, and A. Faize, "Supervised machine learning: A survey," in *2021 4th International Conference on Advanced Communication Technologies and Networking (CommNet)*, 2021, pp. 1–10. DOI: 10.1109/CommNet52204.2021.9641998.

[40] G. Theodoropoulos, R. Minson, R. Ewald, and M. Lees, "Multi-agent systems: Simulation and applications," *by AM Uhrmacher and D. Weyns. 1st. Boca Raton*, 2009.

[41] R. C. Cardoso and A. Ferrando, "A review of agent-based programming for multi-agent systems," *Computers*, vol. 10, no. 2, 2021, ISSN: 2073-431X. DOI: 10.3390/computers10020016. [Online]. Available: https://www.mdpi.com/2073-431X/10/2/16.

[42] P. G. Balaji and D. Srinivasan, "An introduction to multi-agent systems," in *Innovations in Multi-Agent Systems and Applications - 1*, 2010, pp. 1–27. DOI: 10.1007/978-3-642-14435-6_1.

[43] E. De Santis, G. Granato, and A. Rizzi, "Facing graph classification problems by a multi-agent information granulation approach," Nov. 2023, pp. 185–204, ISBN: 978-3-031-46220-7. DOI: 10.1007/978-3-031-46221-4_9.

[44] F. for Intelligent Physical Agents, *Fipa acl message structure specification*, [Online; accessed 06-May-2024], 2002. [Online]. Available: http://www.fipa.org/specs/fipa00061/SC00061G.html%5C#%5C_Toc26669700.

[45] F. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*. John Wiley & Sons Ltd, 2007, ISBN: 978-0-470-05747-6.

[46] R. T. Fielding and J. Reschke, *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*, RFC 7230, Jun. 2014. DOI: 10.17487/RFC7230. [Online]. Available: https://www.rfc-editor.org/info/rfc7230.

[47] T. Karts, *Http requests and methods*, https://www.testkarts.com/blog/post/http-requests-and-methods, Accessed: 2024-08-16.

[48] N. Moustafa and J. Slay, "Unsw-nb15: A comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," pp. 1–6, 2015. DOI: 10.1109/MilCIS.2015.7348942.

[49] A. Dubey, *Cyberattacks detection in iot-based smart city network traffic*, https://pub.towardsai.net/cyberattacks-detection-in-iot-based-smart-city-network-traffic-c874588c5f6c, Towards AI, 2021.

[50] J. Brownlee, *One-hot encoding for categorical data*, https://machinelearningmastery.com/one-hot-encoding-for-categorical-data/, Accessed: 2024-10-16, 2019. [Online]. Available: https://machinelearningmastery.com/one-hot-encoding-for-categorical-data/.

[51] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*, 2nd. O'Reilly Media, 2019.

[52] A. Fernández, S. Garcia, F. Herrera, and N. V. Chawla, "Smote for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary," *Journal of artificial intelligence research*, vol. 61, pp. 863–905, 2018.

[53] A. Hadd and J. Rodgers, "Introduction," in *Understanding Correlation Matrices*. SAGE Publications, Inc., 2021, pp. 1–16. DOI: 10.4135/9781071878613. [Online]. Available: https://doi.org/10.4135/9781071878613.

[54] M. Ali, *Pycaret: An open source, low-code machine learning library in python*, PyCaret version 1.0.0, 2020. [Online]. Available: https://www.pycaret.org.

Appendix A

External Links

A manual containing instructions for running the simulation, along with all the necessary files developed in this work, can be accessed through this Link