POLITECNICO DI TORINO

Master of Science in Ingegneria Informatica
(Computer Engineering)

Master Degree Thesis

Edge-Cloud Platform for Cybersecurity Data Analysis leveraging K3s and Federated Learning

Supervisors
prof. Marco Mellia
prof. Idilio Drago

Candidate
Andrea Sordello

Academic year 2023-2024

Acknowledgements

Desidero esprimere la mia profonda gratitudine a tutte le persone che hanno contribuito alla realizzazione di questa tesi. In primo luogo, ringrazio i miei genitori, Giancarlo e Tiziana, per avermi dato l'opportunità di seguire i miei sogni e per il loro incessante sostegno durante tutto il percorso. Questa tesi è dedicata a voi.

Esprimo inoltre la mia sincera gratitudine ai miei relatori, Marco Mellia e Idilio Drago, per la guida e il supporto costante nello sviluppo di questo lavoro. Ringrazio Rodolfo Vieira Valentim per i preziosi consigli e contributi che hanno arricchito questa ricerca. Un grazie va anche al gruppo di ricerca SmartData@Polito, per il supporto, la collaborazione e l'ambiente stimolante che hanno reso possibile la realizzazione di questo progetto.

Summary

Cyberattacks are growing in complexity due to the adoption of more sophisticated techniques to exploit vulnerabilities. Their identification has become a complex challenge, and novel detection approaches become necessary.

A popular approach is based on the monitoring of network's traffic, which is crucial to observe and distinguish new cyberattacks patterns. In this approach, distributed platforms are generally adopted, since they can significantly enhance attacks detection through the federation of multiple measurement points. These nodes can be strategically placed within different networks to improve coverage and effectiveness. Moreover, they can actively attract advanced attacks when they act as darknets, i.e., ranges of IP addresses without any host active, and honeypots i.e., decoy and vulnerable hosts that log attackers's action, rather than merely traffic sniffer.

In parallel, Artificial Intelligence (AI) and Machine Learning (ML) offer significant opportunities to develop innovative techniques for automated detection. Intuitively, using self-supervised approaches, it is possible to train Neural Network and use the internal representation they create as features for specific tasks. Examples are i-DarkVec and DANTE, both are Artificial Neural Network used to analyze time evolving network's traffic, and to identify potential cyber-threats. When considering multiple monitoring platform, the issue of how to create a common model arise. In fact, sharing data between different measurement points is not simple due to amount of data and privacy implications. This does not allow to leverage the benefits of a broader and collaborative analysis.

This limitation can be address using the Federated Learning (FL) approach. In fact, FL offers the possibility to build a common AI/ML model without sharing data with participants. This brings several advatages, e.g., the amount of data transmitted and risks of privacy violations are reduced, and model accuracy is increased thanks to the larger amount of data that can be used during training. This approach is implemented by training a model in each node with its own collected data. Then, each node shares its own model's parameters. The shared models are aggregated in one single model which it is sent back to the nodes for the next round of training. After some rounds, the aggregated model have an accuracy comparable to a model trained using the entire dataset with the standard ML approach.

Although FL could provide broader insights, its use need to be adapted to time evolving models. The network's traffic evolves over time, e.g., new IPs addresses are discovered. Therefore, models need to update their model's structure to adapt over-time to classify new traffic with new labels. This approach is quite different from typical ML scenarios,

where the labels are known at the beginning and the model is created accordingly and kept fixed.

The contribution of this thesis is threfold: At first, we focus on the improvement of node's management and applications's deployment for a platform developed in a previous work. Then, we design an application to train dynamical models using the federated approach. Finally, we conduct some experiment with the application with different AI/ML models. This is done in preparation for the deployment of the application within the platform, capable of training the i-DarkVec's model using Federated Learning.

Considering the creation of a distributed and flexible platform to collect and process data, we build our platform on a previous proposal to manage several measurement points. In that work, the solution used was K3s, a lightweight Kubernetes distribution, in order to create a cluster with local machines only. In this thesis, we improve how nodes configuration thanks to the use of Ansible playbooks. These support both local and remote machines configuration.

On the nodes, various applications can be deployed on demand. According to the application, the node can function as darknet, honeypot, or simply packet sniffer. The previous work's use K3s's manifest files for the deployment. This approach do not scale well with a large number of nodes, it offers limited customization settings, and it does not support template creation. For this reason, we propose an optimization of the deployment based on Helm charts. Thanks to these optimizations, we were able to create the distributed platform with third-party organization nodes involved e.g., universities. Moreover, we are able to easily control the applications running on the nodes thanks to the use of Ansible and Helm, which allow us to activate or remove applications seamlessly.

Considering the second contribution, we develop, step by step, an application to train the dynamical models using the federated approach. The first step is to choose a FL framework. We conduct an initial survey, in which we explore the available frameworks. At the end, we choose Flower as framework, since it offers the possibility to be easily customized to support the model's architectures evolving over time, it is lightweight, and it is a good-fit to develop our application for resource-constrained edge nodes.

Despite its flexibility, Flower assumes the model's architecture fixed. Given our need to let the model's architecture evolve over time, we modify Flower to include the support for dynamic architecture feature, by adding new data structures and new methods. Our solution introduce additional server-client interactions to make the clients propose updates on model's architecture. Examples of proposals are to add layers, to remove layers or to tweak connections. These proposals are processed by the server to update the model's architecture. Once the aggregation is performed, the updated model is sent to clients for the training phase as usual. Note that we implement the library to be generic as possible and not context-specific.

To demonstrate the flexibility of this newly introduced capability, we validate the enhanced Flower's implementation with two experiments, i.e. Convolutional Neural Network (CNN) for image classification, and i-Darkvec for malicious traffic identification.

We consider the first experiment as initial benchmark to check the correctness of the new model's architecture update process. We use a CNN as model, and MNIST images dataset in which each image can have 1 of 10 possible labels rappresenting digits. In this experiment, each client splits its dataset to generate 10 subsets containing the same digit. At each FL's iteration, a client adds, to its own dataset, one of the subsets obtained at the beginning. Then, the client inform the server about the labels contained in the actual dataset, and the server updates the CNN classification layer accordingly to the labels received. At this point, the client receives the models, and it trains the updated model using its own dataset. We investigate and compare different scenarios's performance related to the choice of subset's labels from clients. As example, two clients could propose the same label over the iterations, or they could choose their label randomly.

We consider the second experiment to check the correctness of i-DarkVec model. We compare the obtained model with a model trained with the traditional ML approach. We obtain similar performance metrics. Moreover, we perform a detailed analysis to investigate the resource consumption e.g., CPU and RAM usage , and of client-server interaction duration. Overall, the results show the benefits of the proposed approach.

The improvements of the initial distributed platform enabled third-party organizations to seamlessly join their machines. With the use of Ansible Playbooks and Helm Charts, node and application management has become straightforward. The application we developed has shown in the two experiment we conducted excellent result both in term of effectiveness and good quality of the metric's performance obtained.

Finally, we discuss several potential directions for future work. The distributed platform should be tested capturing real network traffic from the distributed nodes. Moreover, a pre-processing phase needs to be implemented to transform the captured data into a suitable format for real-time training of the AI/ML model. Increasing the types of honeypots used will improve the quality of traffic analysis. The Federated Learning application could be optimized by reducing the data transferred during client-server interactions, which currently has a significant impact, particularly when the model grows.

Contents

Chapter 1

Introduction

1.1 Context

Cyberattacks are growing in complexity due to the adoption of more sophisticated techniques to exploit vulnerabilities. Their identification has become a complex challenge, and novel detection approaches have become necessary.

A popular approach is based on the monitoring of network's traffic, which is crucial to observe and distinguish new cyberattacks patterns. In this approach, distributed platforms are generally adopted, since they can significantly enhance attacks detection through the federation of multiple measurement points. These nodes can be strategically placed within different networks to improve coverage and effectiveness. Moreover, they can actively attract advanced attacks when are act as darknets and honeypots, rather than merely traffic sniffer.

In parallel, Machine Learning (ML) and Artificial Intelligence (AI) offer significant opportunities to develop innovative techniques the automated detection.

1.2 Challenges

Distributed platforms are used to collect network's traffic data in order to identify cyberattaks. This poses several problems when measurement points are situated across different organizations. In fact, network's traffic contains sensitive information such as IP addresses and unencrypted payloads, which cannot be typically shared between parties due to privacy regulations. Moreover, the volume of the captured data is typically enormous, and data transfer is challenging. As result of these issues, the collected data is generally retained in the organization that gathered them. Hence, the use of Artificial Intelligence (AI) and Machine Learning (ML) technologies is limited to the local dataset of each measurement points, without the possibility to leverage the benefits of a broader and collaborative analysis.

The limitation related to local datasets can be address using a Federated Learning (FL) approach. The idea is to train a model in each node with its own collected data, and to share the parameters of the model to each other. The shared models are then aggregated to build one single model, with an accuracy comparable to a model trained on the entire dataset. This approach ensures that the local dataset remains private, and it enables a comprehensive and collective analysis across all nodes.

Although Federated Learning could provide broader insights, its application needs further developments. The network's traffic evolves over time e.g. new IPs addresses are discovered. Therefore, there is the need of models capable to update their structure to adapt over-time to classify new traffic with new labels. This approach is quite different from typical Machine Learning scenarios, where labels are known at the beginning and the static model is created accordingly. Federated Learning existing implementation are typically based on static models, therefore this additional challenge shall be faced.

1.3 Research Questions

In this section, we outline the key questions to be addressed throughout this thesis.

- **How are these measurements points managed in a distributed computing scenario with third-party organizations involved?**

  The definition of the distributed scenario and nodes management is addressed in the related work presented in 1.4, but this solution is limited to local machines at Politecnico di Torino. We take a further step to significantly improve how nodes configurations are handled, allowing third-party nodes to join our cluster seamlessly. Moreover, we propose a scalable method to deploy applications across the nodes. A more detailed discussion of this topic is provided in Chapter 2.

- **What are frameworks and libraries already implement Federated Learning?**

  We conducted a survey to identify frameworks and libraries that implements federated learning approaches. Our analysis explores their capabilities and their rationale behind. Thanks to this analysis, we chose the framework used to implement our solution. These frameworks are presented in Chapter 4.

- **How can we adapt Federated Learning to accommodate the evolving structure of our trained model?**

  Initially, we identified the information necessary to update the model structure on the fly. Then, we explored how to incorporate this requirement into the federated learning process. Finally, we proposed and apply the solution using a specific framework. Further details can be found in Chapter 5.

- **How did we assess the effectiveness of our proposed solutions?**

  We assessed the effectiveness of our proposed solutions through experiments using a 1) Convolutional Neural Network (CNN) model for image classification, and 2) the i-DarkVec model, presented in (10), for network traffic analysis. A detailed discussion of these experiments is presented in Chapter 6.

1.4 Previous Work

The foundations of this thesis are built upon the prior work presented in (8; 10). The first one offers a solution to manage distributed physical machines, meanwhile the second one presents the Machine Learning model i-DarkVec to identify threats. An introduction is given in the following sections.

1.4.1 Management of Distributed Physical Machines

The work presented in (8) offers a solution to manage distributed physical machines, in particular how to instruct them to perform specific actions. The solution proposed is to use a K3s[1] cluster. This choice allows to operate at cluster-level, since it abstracts the complexities of the physical layer. Furthermore, it simplifies applications management through the use of pods[2] which can be easily deployed or removed from a node. Communication between nodes within the cluster is secured using WireGuard VPN. The Figure 1.1 shows the different abstraction used to simplify the management of distributed nodes.

The setup of the physical machines is handled through Bash scripts to facilitate configuration management, such as to either join an existing K3s cluster or establishing a new one by properly install all the necessary dependencies on the physical machines.

The captured network traffic is stored locally on each node, and the use of any machine learning solution to process data is not implemented yet.

1.4.2 Threats Identification with i-DarkVec

The work presented in (10) introduces a Machine Learning model, known as i-DarkVec, to analyze time-evolving traffic and to identify potential cyber-threats. i-DarkVec employs a clustering technique to group unknown IP addresses, either associating them with known malicious actors or uncovering new sources conducting similar attack patterns. An example of result related to attacks identification is shown in Figure 1.2. The learning approach allows to dynamically adapt the model structure, in order to handle large-scale traffic effectively. A more detailed explanation of i-Darkvec's model will presented in 3.2.1.

[1]K3s is a lightweight Kubernetes distribution, specifically designed for IoT and Edge computing.

[2]A pod is the basic unit running a containerized application.

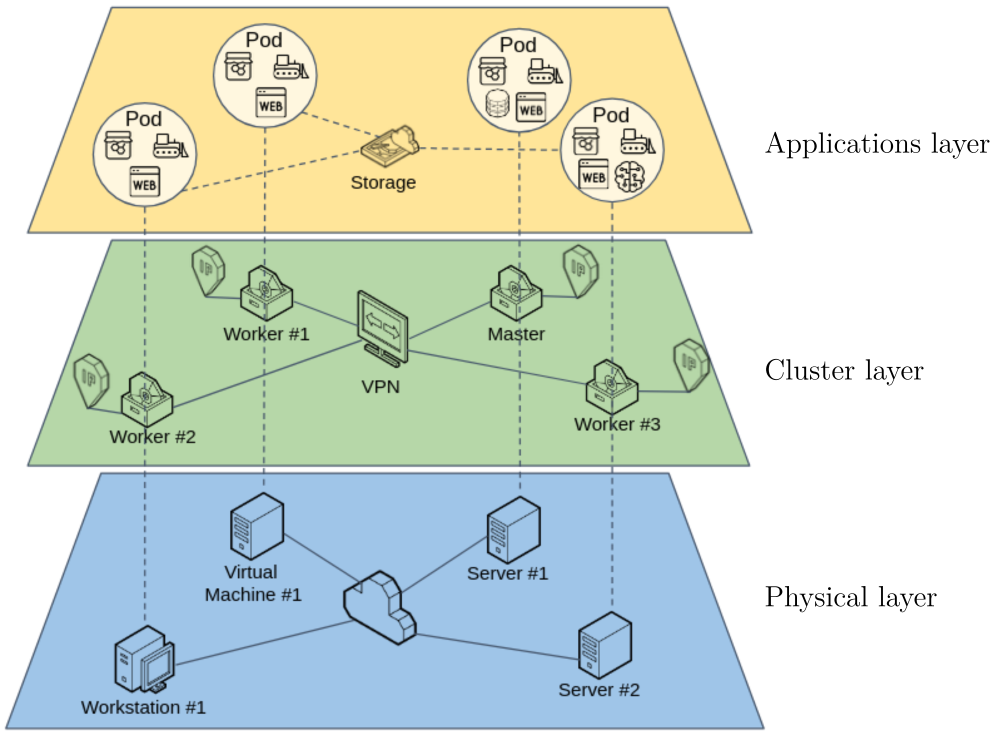

Figure 1.1: The figure describes the plaftform's infrastructure presented in (8). The network is rappresented on three different horizontal layers. The vertical lines rappresents instead the relations between physical machines, cluster agents, and applications.

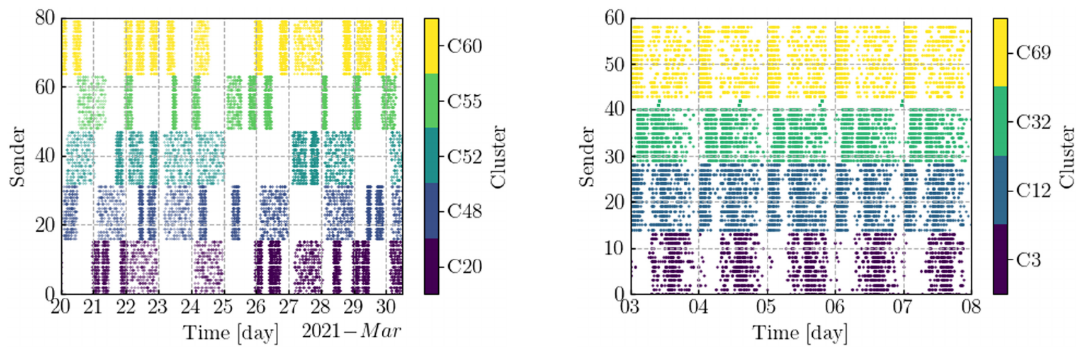

Figure 1.2: The figure shows an example of clusters activity discovered by i-DarkVec. Each cluster is identified by the identifier Cx, where $x \in \mathbb{N}$ is a number used to indentify the cluster itself. Credits: (10).

1.5 Contributions

In this section, we list the main contributions of this thesis.

- **Optimized Node Management**. We improved the management process of nodes within the platform and the deployment of applications, by leveraging Ansible and Helm technologies .
- **Survey of Federated Learning Frameworks**. We conducted a comprehensive survey on the state of the art of Federated Learning frameworks.
- **Extension of the used Federated Learning Framework**. We proposed an enhancement of Flower, which is the chosen Federated Learning framework to support potential updates to the model structure.
- **Training of a Convolutional Neural Network (CNN) using Federated Learning supporting evolving class**. The application we developed is built on the enhanced framework that supports updating the model's architecture during training.
- **Training of i-DarkVec using Federated Learning**. We created a scenario with i-DarkVec for network's traffic classification. We integrated it using the enhanced Flower framework.
- **Performance Analysis**. We conducted performance analyses to assess the efficiency of the implemented models.

1.6 Index

In this section, we summarize the topics covered in each chapter of this thesis.

In chapter 2, we present how we manage the platform with third-party organizations. Chapter 3 explores the concepts of Machine Learning and Federated Learning. In Chapter 4, we conduct a comprehensive survey of existing frameworks, evaluating their strengths and weaknesses in relation to our objective. Chapter 5 focuses on the integration of model's structure update in Federated Learning. Chapter 6 presents results related to the federated training of a Convolutional Neural Networks (CNN) for image classification, and i-DarkVec for network's traffic classification. Finally, Chapter 7 summarize the thesis's work, and suggests idea for future research.

Chapter 2

Distributed Platform Design and Deployment

2.1 Introduction

In this chapter, we improve the current configurations script and deployment process of applications within the platform proposed in (8).

These improvements are necessary because we want to build the platform using machines belonging to other third-part organizations, rather than machines only at Politecnico di Torino.

For this reason, we present a straightforward installation process to be independently executed by each organization , without our intervention. Furthermore, since we may not be able to connect to these machines remotely, the proposed solution need to be a user-friendly for the third-party owners, which need to configure their machines to join in the platform. The solution we propose is to use Ansible playbooks to handle the installation process because they are a good-fit to configure at the same time multiple local-or-remote machines.

We propose an alternative solution to optimize the deployment of the application within the platform. The current management approach, using K3s's manifest files, does not scale well. Each change requires locating and manually editing the correct manifest file, which becomes unsustainable with a large number of nodes and applications. Furthermore, K3s's manifests offer limited customization options and do not support template creation. For this reason we decide to adopt Helm charts for the deployment of applications.

2.2 Technologies used

2.2.1 Ansible

Ansible is an open-source automation tool to simplify tasks such as configuration management, application deployment, and orchestration. It allows users to automate complex processes efficiently across multiple systems, from configuring machines to running scripts on them. It operates with a simple, agent-less architecture. It uses playbooks in the the YAML format to define tasks. A key component is its inventory i.e., an external YAML which contains variables and details about the managed system. Further details can be found in the documentation (1).

2.2.2 Helm

Helm is an open-source package manager for K3s, which simplifies the deployment and management of applications in clusters. It uses preconfigured templates, known as charts, to define K3s resources and to streamline the deployment process. Users can version, update, and roll back applications easily. For this reason, it is an essential tool to automate the deployment of complex K3s applications and to manage them consistently across environments. From an architectural point of view, Helm charts are composed by two parts. The first one lists the collection of template files used to define K3s's resources to be deployed. The second one contains the values with which the template's files are populated. Moreover, when the deployment is created, Helm offers the possibility to assign additional values on-the-fly. Further details can be found in the documentation (14).

2.3 Definition of a Configuration File

We propose the creation of a single configuration file to consistently manage node and application deployment. This centralized configuration simplifies the management of nodes and modifications of the applications running on them. This approach becomes particularly beneficial as the number of nodes increases.

The file's format is YAML to be capable of being recognized as inventory by Ansible. The file's structure is divided in two main sections as shown in Figure 2.1. The first section is used by Ansible to manage the nodes. It contains the connections settings of each node of the network, and the parameters of the WireGuard VPN which has been used. The second section is used by Ansible to instruct Helm how to deploy the applications. It contains the settings for applications deployed for each node. These applications include the Docker registry, darknets, honeypots, and the enhanced Flower framework used for our Federated Learning training.

.

2.4 Ansible Playbook Configuration

As discussed in the section 1.4, current configuration process relies on several Bash scripts to be manually executed on each node physical machine to configure the system. This process is inefficient and prone to errors as the number of nodes grows and when updates are required. Ansible playbooks were adopted to improve the configurability and the management of the platform. This permits to handle several scenarios such as

- the initial creation of the platform,
- the addition and the removal of a node from the platfrom,

- the enablement and disablement of an applications on specific nodes thank to the cooperation with Helm.

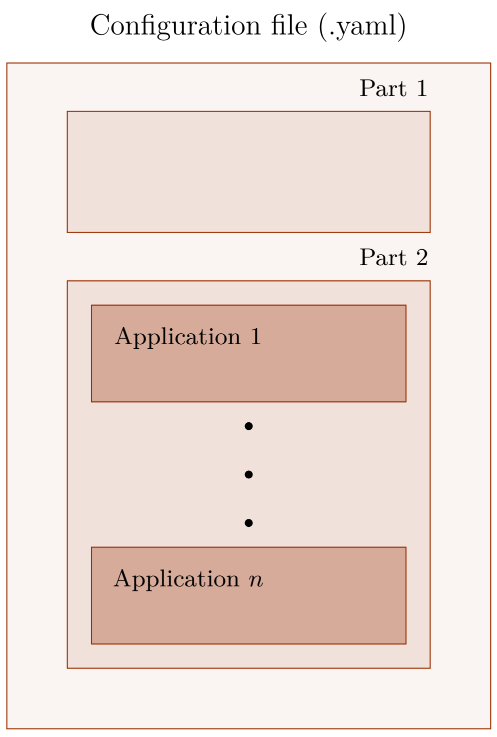

Figure 2.1: The figure shows the structure of the configuration file.

2.5 Helm Deployment

Regarding the application deployment, the previous work presented in (8), used an approach based on the K3s manifest files, which contain all the needed information hard-coded. This approach has lack of flexibility and customization. For this reason, a new and powerfull tool, known as Helm, is adopted. Helm allows to create parameterized deployments, referred as Helm Charts, which can be easily integrated with the configuration file presented in the section 2.3.

In our proposed solution, the deployment process is still managed via Ansible playbooks. A initial playbook determines, from the configuration file, which application has to be installed. After that, it invokes additional playbooks to install the applications that extract still from the configuration file the values to be used by Helm for the deployment. The process is shown in Figure 2.2.

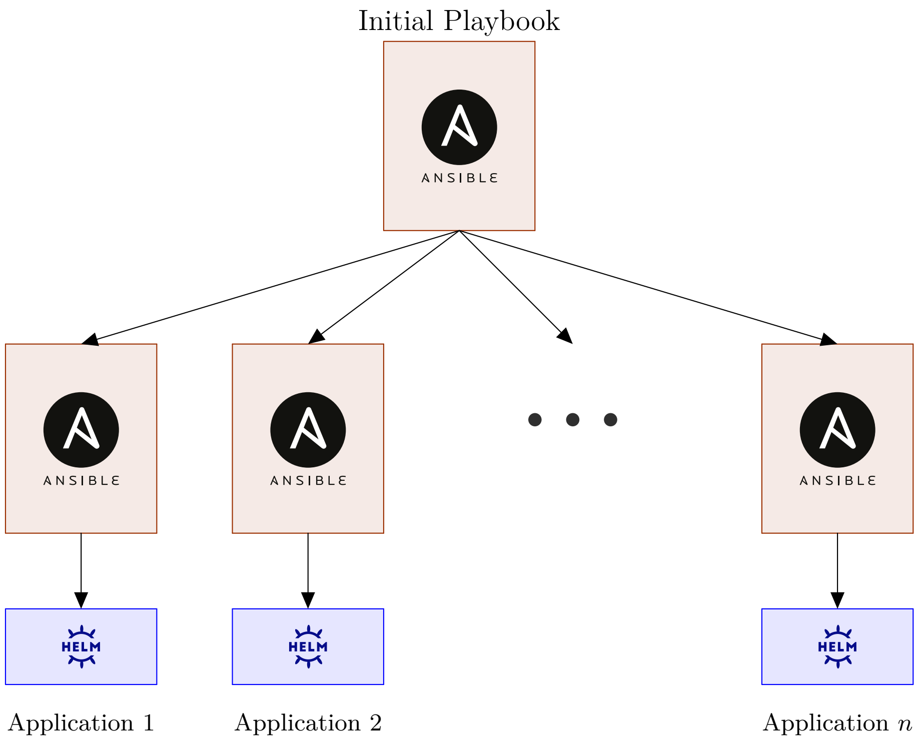

Figure 2.2: The figure illustrates the application deployment process. An initial playbook triggers the application's playbooks, which these invokes the corresponded Helm Chart to manage the deployment.

2.6 Current Platform Nodes

The use of Ansible playbooks allows us to seamlessly integrate machines from external organizations into our platform. As shown in Figure 2.3, the current platform consists in multiple nodes connected to the central server at Politecnico di Torino.

The server acts as the K3s master and WireGuard VPN server. It hosts the Docker registry, which maintains and distributes the containerized applications. On-demand applications can be activated as needed, such as the Federated Learning server, which is used to perform Federated Learning tasks.

Each measurement's point is connected to the central server via VPN, and it is configured to be a K3s's agent. Within these nodes are activated several applications according to the needs specified in the configuration's file presented in Section 2.3. Once the file is configured, it is required to execute the Ansible Playbook related to the applications management. The applications running on the nodes are automatically updated.

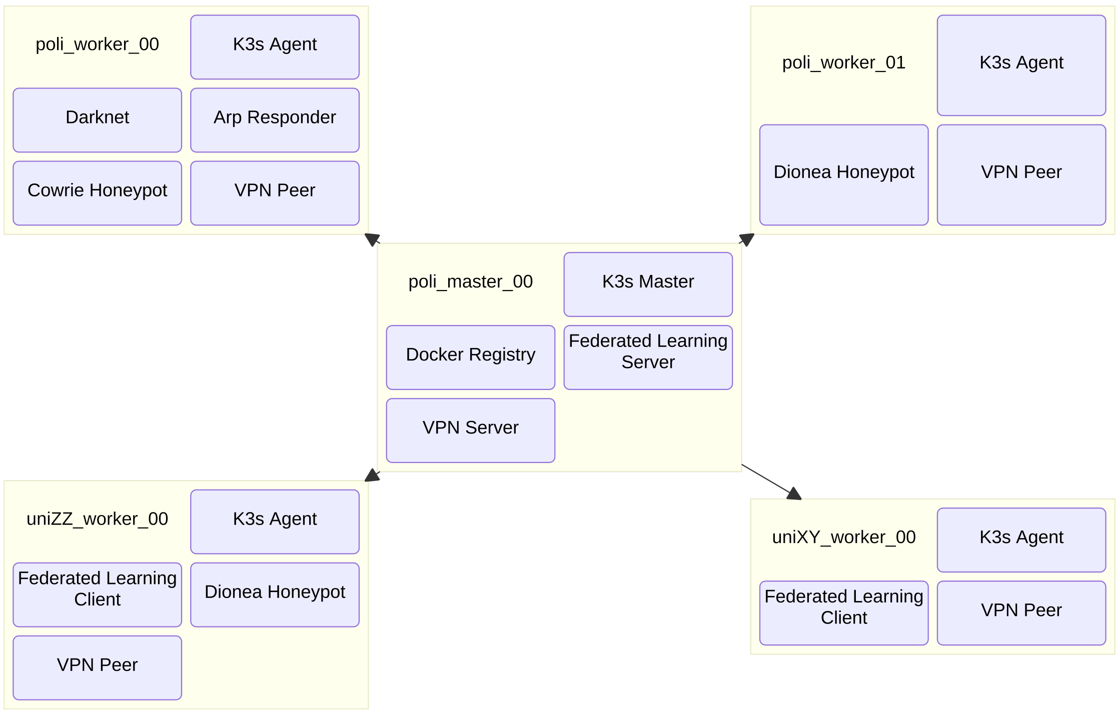

Figure 2.3: The figure shows five different nodes within the platform, the *poli_master_00* is the server, while the others are clients placed in different networks respect to the server's network. Note that in this figure we have only two Federated Learning clients connected to the server on the *poli_master_00* machine.

2.7 Conclusion

In this chapter, we presented significant enhancements to the configuration and deployment processes of the platform presented in (8).

We streamlined the installation and management of applications across both local and third-party nodes thanks to the transition from Bash script to Ansible playbooks. This simplifies the on-boarding process for third-part organizations, and it reduces the likelihood of errors related to configuration. In particular, we adopted an approach with a single configuration file to simplify management as the number of nodes grows.

The adoption of Helm charts allowed us to create parameterized deployments, enhancing flexibility and customization when deploying applications within the K3s environment.

These improvements allowed us to integrate third-party machines into the platform. This made possible to and build a large-scale plaftorm able to collect and to process malicious traffic.

The Ansible playbooks and other files related to the platform are contained in the Github repository at `https://github.com/SmartData-Polito/hiac`.

Chapter 3

Introduction to Artificial Intelligence

Artificial Intelligence (AI) refers to the use of technologies to build machines and computers that have the ability to mimic cognitive functions associated with human intelligence, such as being able to see, understand, and respond to spoken or written language, analyze data, make recommendations, and more (3).

The most popular subfield of Artificial Intelligence (AI) is Machine Learning (ML) will be introduced in section 3.1. In particular, we will focus on Artifical Neural Network (ANN) models to which the section 3.2 is dedicated. In our applications, a great attention is given to i-DarkVec, which is a particular ANN architecture used for malicious traffic identification. For this reason, it is presented in section 3.2.1. Finally, the federated approach for training is presented in section 3.3. This is usefull since the models presented in our results are trained in a distributed way, typical of this approach.

3.1 Machine Learning

Machine learning (ML) is a subset of AI that consist in algorithms that learn new information from data without being explicit programmed. Rather than relying on hard-coded rules, machine learning algorithms leverage patterns, trends, and relationships in data to make predictions, decisions, or classifications (3). ML has also found widespread applications in industries such as healthcare, finance, transportation, entertainment and cybersecurity.

The development of a ML model is based on a well-established key phases after data has been collected: i) data processing ii) model selection iii) model training and iii) model testing.

3.1.1 Data Preprocessing

The initial step is data preprocessing, which involves transforming raw data into a suitable format for the model. Common preprocessing tasks include handling missing values, scaling features to a consistent range, and encoding categorical variables into numerical data. During this stage, the dataset is splitted into training and testing sets. The training set is used to train the model, while the testing set checks how well the model performs on new, unseen data. A common split ratio is 80% for training and 20% for testing.

3.1.2 Model Selection

Once the dataset is prepared, the appropriate model is selected based on the task and the nature of the data as different models perform better on different types of data and problems. For instance, regression models may be suitable for predicting continuous outcomes, while classification models are used for categorical outputs.

3.1.3 Model Training

Next, the model is trained by feeding the training data and adjusting its parameters to improve the model's performance.

Eventually, in the training phase can be insert an additional phase known as model validation. This phase is used i) to optimize model's parameters which cannot be trained during the training phase known as hyperparameters, ii) to avoid overfitting and underfitting. Note this requires an additional split of the training set data in two set: a new training set with which the model will be effectively trained, and a validation set to be used for the model validation.

3.1.4 Model Testing

At the end, the model is tested using the testing set to estimate how well it will perform on new, unseen data. Various metrics can be employed to assess performance. The most used metrics are presented below for a classification problem with two possible outcomes.

- *Accuracy* is the proportion of correctly predicted instances out of the total number of instances.
- *Precision* is the proportion of true positives among all predicted positives.
- *Recall* is the proportion of true positives among all actual positives.
- *F1-Score* is the harmonic mean of precision and recall.

These metrics can be extented to classification problems with more than 2 classes.

3.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a particular class of machine learning models. These consists in connected nodes, called neurons, where each node performs the mathematical operation

$$y = f\left(\sum_{i=1}^{n} \omega_i x_i + b\right),$$

where y is the neuron output, x_i is the $i-$th input, $\omega_i \in \mathbb{R}$ is known as $i-$th weight, $b \in \mathbb{R}$ is known as bias, and the non-linear function f is known as activation function. Most used ANN models operates are feed-forward ANN i.e., the output of each node is connected to the input other nodes without creating loops and internal states. The typical architecture is organized using several layers of interconnected neurons. This includes an input layer which is feeded with input data, an output layer which provides the results of the internal operations, and eventually a group of intermediate layers known as hidden layers. An example is depicted in Figure 3.1.

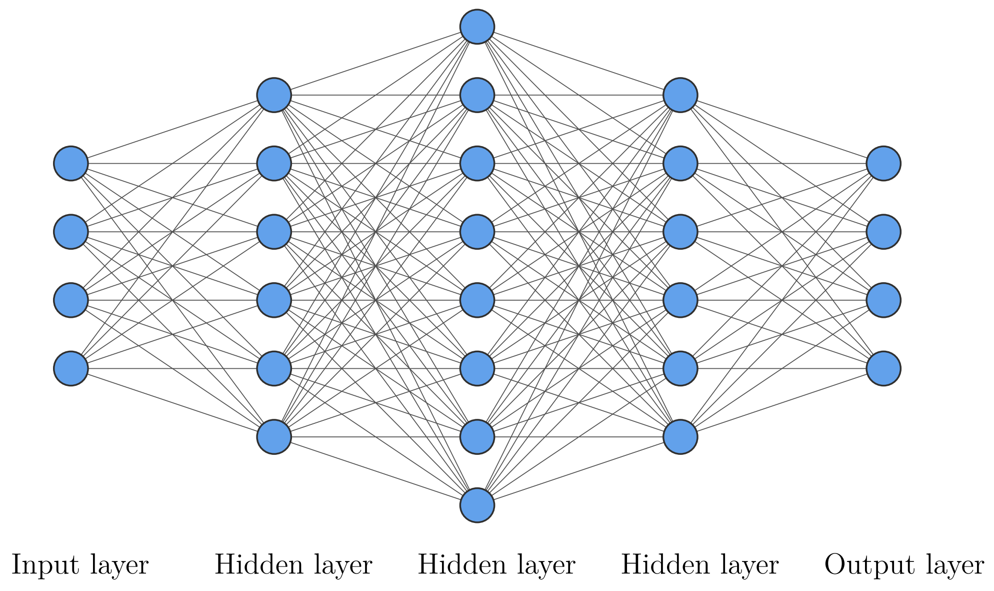

Figure 3.1: The figure shows an example of ANN. The first layer is known as input layer, the last layer is known as output layer. The central layers are known as hidden layers.

This feed-forward ANN models are trained by adjusting the weights and the biases of the network nodes so that the inputs of the training set are close to the outputs of the training set. This can be formalized as the minimization of the loss function, which is an error function between data outputs and network outputs computed from data inputs.

The most used algorithm is known as Backpropagation, which is an iterative algorithm that minimize a loss function step by step by adjusting weights and biases. Each step usually consider a subset of data with which a loss function is built. Now, some definitions usually related to this training algoritm are given.

- The batch is the subset of data processed at each step. The batch size is the number of training samples in the batch.

- The epochs defines a single pass through the entire training dataset.

More informations can be found at (11).

3.2.1 I-Darkvec Model

Different ANN architectures have been proposed for differents tasks. Here, we present i-DarkVec which is an architecture, presented in (10), used for the pattern extraction of malicious traffic. This is based on Word2Vec, a Natural Language Processing technique based on ANN. This technique uses the words as input by using an word-to-vector analogy i.e. allows each word to be identified by a vector. To do it, a vocabulary of N possible words is used. Several word-to-vector maps have been proposed, but the most popular is based on one-hot-encoding. The vector obtained is then used as ANN input, and the output is the word embedding which is a representation of the word in a smaller vector space with dimension $e \ll N$. Two words close in the word embedding space are similiar in meaning. Note that the weights of the ANN that goes from vector space to embedding words space are known as embeddings.

i-Darkvec exploit the Word2Vec approach, but it uses IPs addresses instead of using words from human sentences. More details are presented in (10).

3.3 Federated Learning

3.3.1 Introduction to Federated Learning

As machine learning continues to evolve, new methodologies and paradigms emerged to address limitations and enhance its capabilities.

One of major limitations is that a large number of architecture are data-hungry i.e. requires a large training dataset to be stored in a single machine. This data can be also collected from other machines, and then sent to a central machine, e.g. a server, which will train the model using data provided by the other nodes. Although this approach is easy to be implement, it has two main drawbacks:

- There is significant network bandwidth usage, since each node must upload their entire dataset in the server. This can be time-consuming and resource-intensive.

- There are privacy concerns due to the data visibility over the entire network. This raises issues related to privacy regulations.

Federated Learning addresses these limitations with a new approach. Instead of group data of every node on a single centralized server and train a single model, federated learning use a different approach. In this case, a model is trained in each node with data locally available, then the parameters, e.g. weights and biases for an ANN, are sent to the central node where all the differents models are assembled into single one. As last step, the parameters of the aggregated model are sent back to the nodes which continue to adjust the model using data locally available. Note that to evaluate the performances of the aggregated model, in each node can be computed a testing metric using the agggregated model and the testing dataset locally available. This information is sent to the central node. The difference of the classical approach and the federated approach is illustrated in the Figure 3.2.

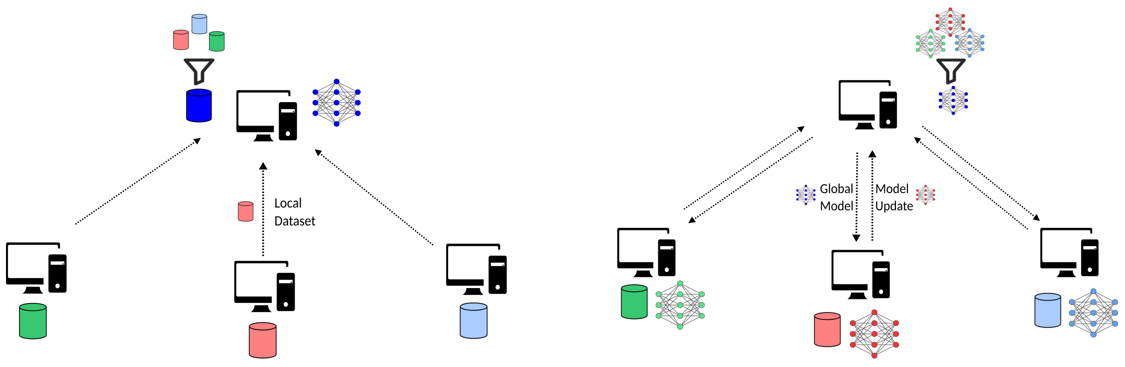

Figure 3.2: The picture on the left shows the classical Machine Learning approach. The picture on the right shows the Federated Learning approach. In each of the two system the data are collected on the nodes.

3.3.2 Federated Learning Advantages

As discussed in (12), federated learning has several advantages. Some of them are listed and discussed below.

- *Privacy by Design.* The most important advantage is to guarantee data privacy. In fact, models are trained on decentralized data without sharing any raw data between nodes. For this reason, the risk of data leakage or exposure is significantly reduced.
- *High scalability.* In the classical approach, data-scaling becomes impractical. Federated Learning is inherently more scalable because it leverages local computation on distributed devices or servers, avoiding the bottlenecks of central data collection.
- *Reduced network usage.* Federated Learning reduces the amount of data shared. In fact, data dimension of model parameters is tipically much lower than the dataset size with which the model is trained.

3.3.3 Federated Learning Scenarios

Federated Learning is increasingly applied across multiple fields. Some examples are presented below.

- *Google Gboard* is a widely used Android keyboard application. GBoard uses Federated Learning to train the Next Word Prediction (NWP) model which provides the suggested next words to appear above the keyboard while typing. More details can be found at (13).
- *Hospitals* are often constrained by strict privacy regulations. In such cases, bandwidth and computational resources are rarely a problem compared to data privacy. A practical usage of federated learning in this sector is for analyze chest computed tomography scan performed by different hospitals. More details can be found at (23).

3.3.4 Federated Learning Classifications

Federated Learning implementations can be classified either by the topology of the network created by nodes or by how features[1] and samples of data are distributed among the nodes.

Network topology classification

Federated Learning can be implementeed using different network topologies.

The most used topology is the *centralized* one. In this setup, a central node manages the communication with the clients nodes.

In contrast, *decentralized* topology eliminates the need of a central node by enabling all the participants to communicate directly with one another. In this approach, each node shares the model's parameters with neighbors, forming a peer-to-peer network.

In some scenarios, *hybrid* topology are prefered. These combines elements of centralized and decentralized approaches.

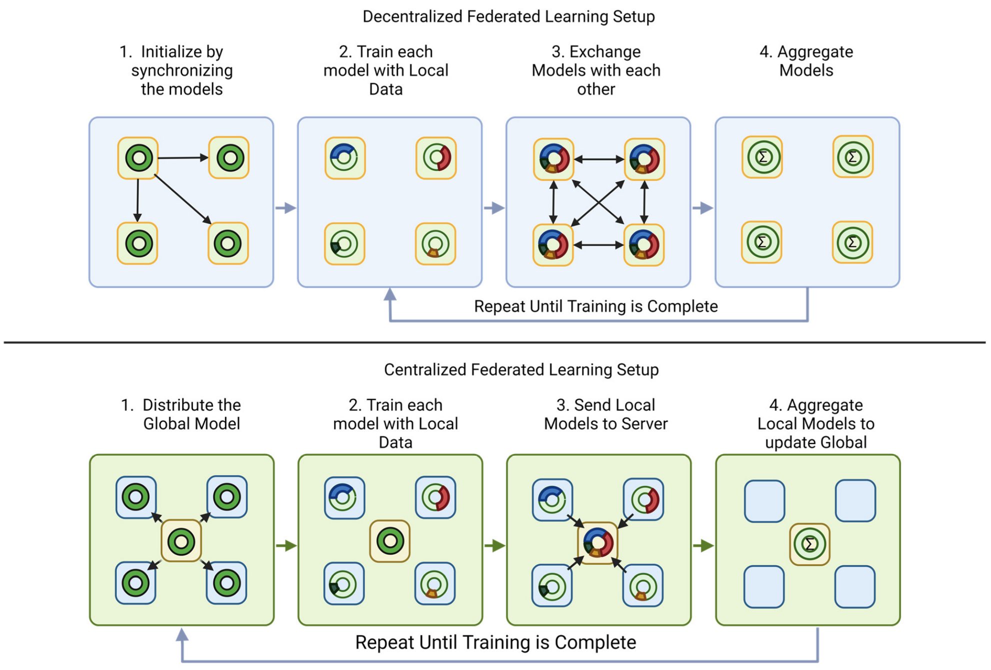
Decentralized Federated Learning Setup
1. Initialize by synchronizing the models
2. Train each model with Local Data
3. Exchange Models with each other
4. Aggregate Models
Repeat Until Training is Complete
Centralized Federated Learning Setup
1. Distribute the Global Model
2. Train each model with Local Data
3. Send Local Models to Server
4. Aggregate Local Models to update Global
Repeat Until Training is Complete

Figure 3.3: The figure shows the centralized and decentralized topologies used in federated learning. Credits: (20).

[1]A data instance is represented by a collection of its features.

Data distribution classification

Federated Learning can be classified, according to (24), looking at distribution of the data samples and data features present in different nodes. Three type of distribution are presented below, and shown in Figure 3.4.

In *Horizontal Federated Learning* (HFL), data of different nodes shares the same features but they can contain differents samples of data. This distribution comes out in cases where organizations have similar types of data but distinct sets of users or objects.

In *Vertical Federated Learning* (VFL), data of different nodes shares the same samples of data, but each sample can contain different features. This distribution comes out in cases where differents organizations hold complementary data on the same users.

In *Federated Transfer Learning* (FTL), data of different nodes can not shares the same features, and can contain different samples.

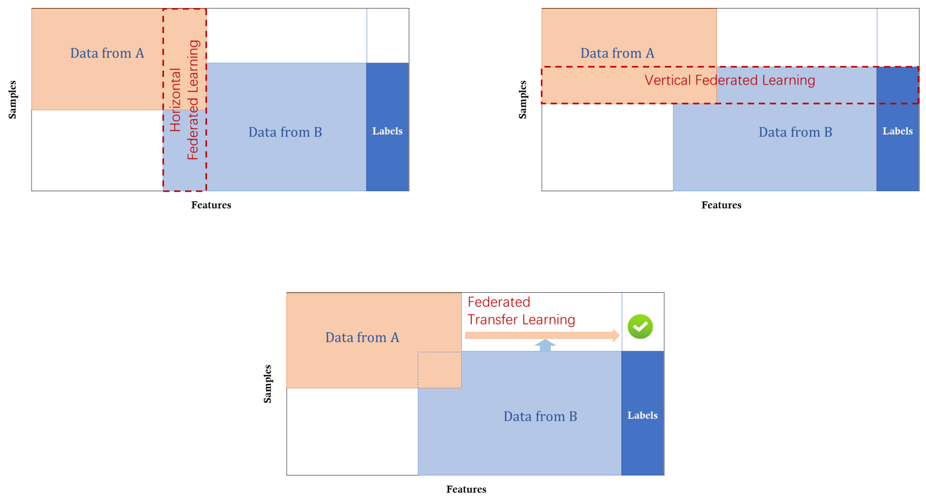

Figure 3.4: The figure shows the difference between horizontal federated learning, vertical federated learning and federated transfer learning. Credits: (24).

3.3.5 Federated Learning challeges

According to (2), Federated Learning approach has some drawbacks. Some of them are listed and discussed below.

- *Systems heterogeneity.* Federate Learning network consists of nodes with differents storages, computational powers, and communication capabilities. Therefore, the training process must be adapted to the capabilities of the least performing device.
- *Model Poisoning.* Nodes might intentionally or accidentally mislead the server with malicious model's parameters.

- *Data Distribution.* Data contained in the local dataset are not independent and identically distributed. Machine learning tecniques typically relies on this assumption, therefore more sophisticated techniques must be developed to handle statistical heterogeneity.
- *Federated Fairness.* Federated learning does not guarantee fairness between nodes. Factors like connection quality, device kind, geographic location, and dataset size may influence how samples are selected in the process.

Chapter 4

Frameworks for Federated Learning

In this chapter, we present some of the most known frameworks and libraries which implement the essential logic and communication mechanisms required for Federated Learning. An overview of their capabilities and characteristics is provided. As conclusion, we choose Flower as framework, and we explain why it is the best fit for our solution.

4.1 NVIDIA FLARE

NVIDIA Federated Learning Application Runtime Environment (NVIDIA FLARE) (5) is an a open-source development kit (SDK) designed to facilitate the implementation of Federated Learning solutions. It is highly flexible and business-ready, but it is not user-friendly at the beginning (21).

The framework consists of several key components. The most important is the FLARE API, which serves as the foundation for running a Federated Learning application. Others components include a Federated Learning Simulator for prototyping and testing workflows, and the FLARE Dashboard, which is used to manage and monitor deployments.

The central concept of FLARE is a unit of work, known as *Task*, used to define specific steps in the federated learning process. Tasks can be executed by both clients and servers. Example of tasks are to train, to aggregate, and to evaluate. The *worflow* defines the sequence and order of tasks that has to be performed. One example of workflow is the *Scatter and Gather* as shown in Figure 4.1.

The collaboration between clients and servers is facilitated through *controller-worker* interactions. The *controller* assigns tasks to *workers*, processes the results returned, and makes decisions about the next steps. The Figure 4.2 shows how a controller and a worker interact.

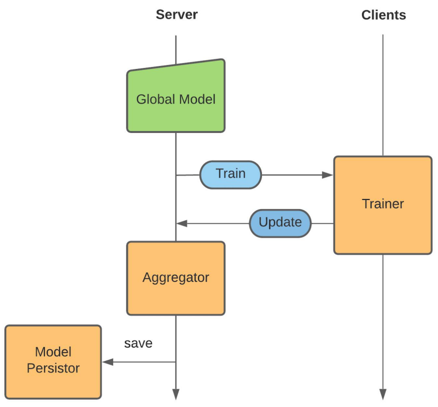

Figure 4.1: The figure shows the tasks performed during *Scatter and Gather* workflow. Credits: (5).

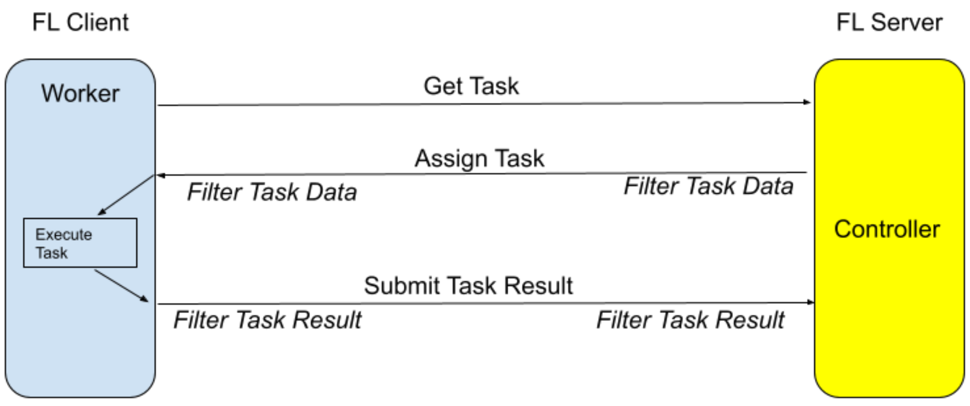

Figure 4.2: The figure shows the interaction between a controller and a worker. Credits: (5).

4.1.1 Federated Learning Simulator

The Federated Learning Simulator is a lightweight tool for a running a NVIDIA FLARE deployment, and it allows researchers to test and debug their application without provisioning a real project.

This simulator is designed to allow to implements federated learning systems using single machines, such as laptops. Once the application has been developed and debugged, it can be directly deployed on a production system without any change.

4.2 FATE

FATE (17) is an open-source project launched by WeBank's AI Department. It is designed to provide a secure computing framework for the Federated Learning ecosystem. It includes numerous modules for data preprocessing and various machine learning algorithms. It offers backends for popular Deep Learning libraries like PyTorch and TensorFlow.

The application is composed by several modules. These includes the ones listed below.

- *FederatedML* is the core machine learning library. It contains algorithms to facilitate Federated Learning.
- *FATE Cloud* is designed to scale federated learning tasks, deploy across different cloud providers, and manage resources dynamically. It is compatible with cloud-native technologies like Kubernetes and Docker.
- *FATE Board* is as a visualization tool. It provides an interface to monitor and evaluate key metrics such as loss, accuracy, and other performance indicators throughout the model training process.
- *FATE Flow* is the workflow engine used to orchestrate Federated Learning tasks, manage job scheduling, execute tasks, and manage models.

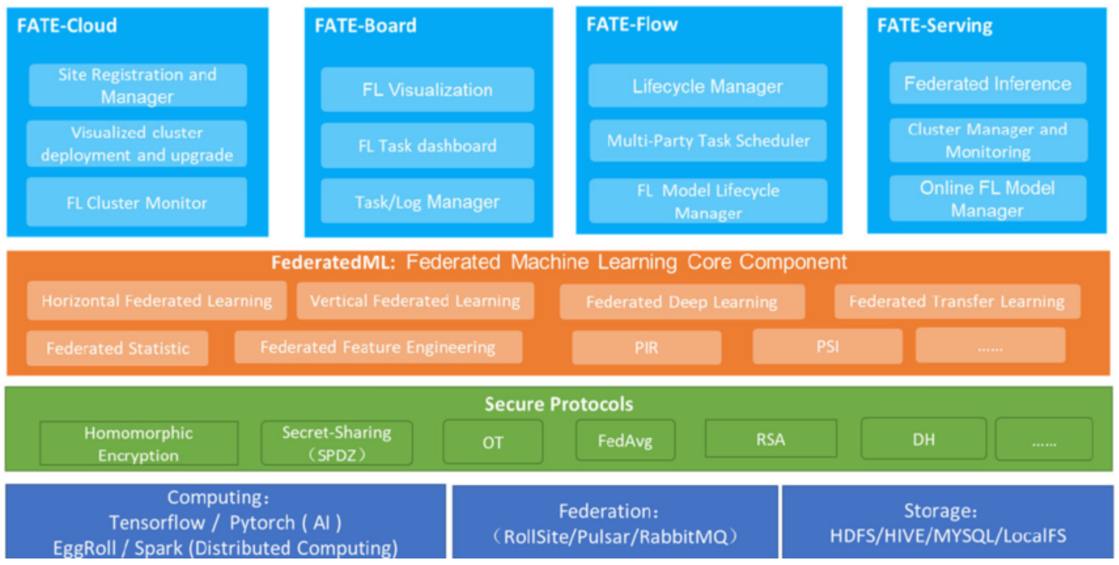
FATE-Cloud
Site Registration and Manager
Visualized cluster deployment and upgrade
FL Cluster Monitor
FATE-Board
FL Visualization
FL Task dashboard
Task/Log Manager
FATE-Flow
Lifecycle Manager
Multi-Party Task Scheduler
FL Model Lifecycle Manager
FATE-Serving
Federated Inference
Cluster Manager and Monitoring
Online FL Model Manager
FederatedML: Federated Machine Learning Core Component
Horizontal Federated Learning
Vertical Federated Learning
Federated Deep Learning
Federated Transfer Learning
Federated Statistic
Federated Feature Engineering
PIR
PSI
......
Secure Protocols
Homomorphic Encryption
Secret-Sharing (SPDZ)
OT
FedAvg
RSA
DH
......
Computing: Tensorflow / Pytorch (AI) EggRoll / Spark (Distributed Computing)
Federation: (RollSite/Pulsar/RabbitMQ)
Storage: HDFS/HIVE/MYSQL/LocalFS

Figure 4.3: The figure shows the FATE basic components. Credits: (18).

4.3 OpenFL

Open Federated Learning (OpenFL) (19) is an open-source project developed by Intel. It offers a modular, flexible, and secure platform to implement Federated Learning.

The architecture includes several key component which are listed below.

- A *collaborator* is a client which performs the training and owns the data in the experiment.
- An *aggregator* is the central server responsible for coordinating and aggregating model updates from collaborators.
- A *task runner* is a component which execute tasks like data preprocessing, model training, and the communication to the orchestrator of the trained model updates. Moreover, it defines the training and testing settings.

In the Figure 4.4, it is shown how the components works together.

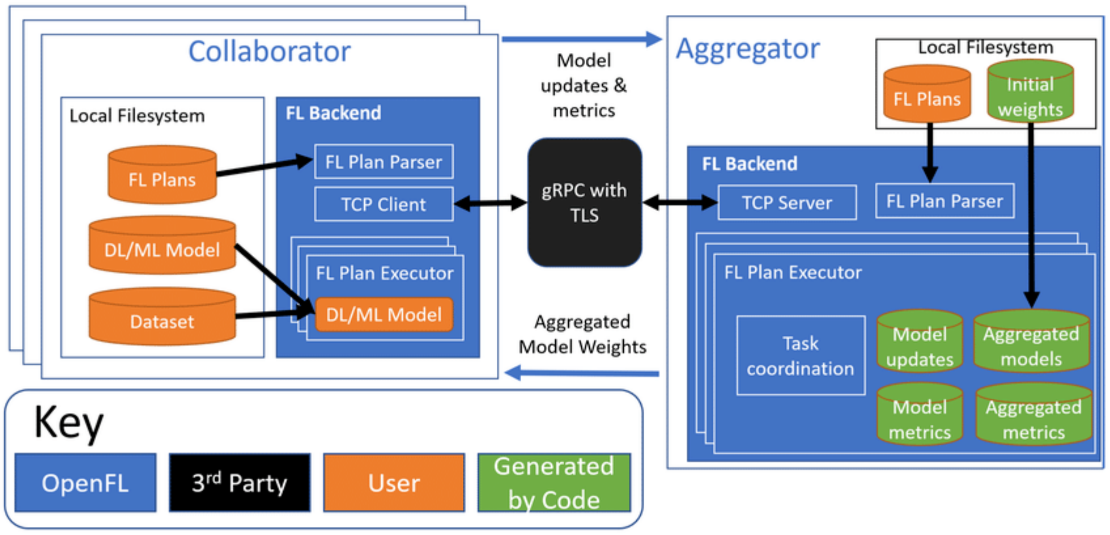

Figure 4.4: The figure shows the OpenFL components and how they interact. The *collaborator* uses a local dataset to train a model, while the *aggregator* receives model updates from *collaborators*, and combines them to form an aggregated model. Credits: (19).

OpenFL supports four ways to set up experiments. These are shown below.

- The Task Runner API is recommended for production scenarios where workload verification is required before execution.
- The Python Native API provides a streamlined Python interface intended for simulations.
- The Interactive API simplifies the setup of a federation and introduces long-lived components such as the *Director* and *Envoy*.
- Workflow API offers increased flexibility for researchers and developers.

4.4 Tensorflow FL

TensorFlow Federated Learning (Tensorflow FL) (6) is a framework developed by Google, and it is integrated with the known TensorFlow ecosystem. This helps to take advantage of the related Machine Learning tools and libraries.

The architecture of the framework consists of the two layers described below.

- *Federated Learning API* is a layer which offers high-level interfaces to allow developers to apply pre-built Federated Learning implementations to existing TensorFlow models.
- *Federated Learning Core* is the layer which provides lower-level interfaces to express custom algorithms of Federated Learning.

Although the framework shows great potential, it is relatively new, and studies such as (15) pointed out it still lacks of several essential components. Currently, it can only operate in simulation mode, since remote mode is not fully developed yet.

4.5 Flower

Federated Learning Framework (Flower) (22) is an open-source framework designed by Adap. It facilitates Federated Learning across various environments, including edge devices, mobile devices, and cloud infrastructure. It is highly flexible. For this reason, it is popular choice for researchers and developers. It allows to work with several machine learning libraries like TensorFlow, PyTorch, and JAX. Moreover, it allows to define own strategies for client selection, aggregation of model updates, and communication protocols. Finally, it offers detailed documentation and tutorials.

As side note, Flower is a very active project going through a lot of changes. The information reported are based on version 1.8. We do not guarantee everything will work as expected in more recent versions.

4.5.1 Flower Infrastructure

Flower framework has several components to build Federated Learning systems. We now describe the most important components.

Flower Client

A *client* is a Python class implemented with one method to train the model, and one method to evaluate the model. This class is built upon one abstract class which can be chosen from two alternatives. The first one is NumpyClient and it uses Numpy Python library to handle models parameters. The second one is Client, and it offers more flexibility by representing the models parameters using list of bytes.

Flower Server

A *flower server* is an instance invoked by a method, which waits clients to connect. This is not a class, in contrast to *flower clients*. However, the server does not permit any customization which is managed through another component known as *flower strategy*.

Flower Strategy

A *flower strategy* is a class which defines the logic and details of the tasks performed by the server. They can be implemented to accommodate various aggregation rules, optimization criteria, and the number of clients involved in each operations. Therefore, this allows significant customization of the server behaviour. However we can only modify certain parameters of the server.

4.5.2 Flower Networking

Flower facilitates communication between server and clients using gRPC[1] or HTTP networking technologies. It operates in a star topology, in which the server serves as the central hub. The flower networking architecture is shown in 4.5.

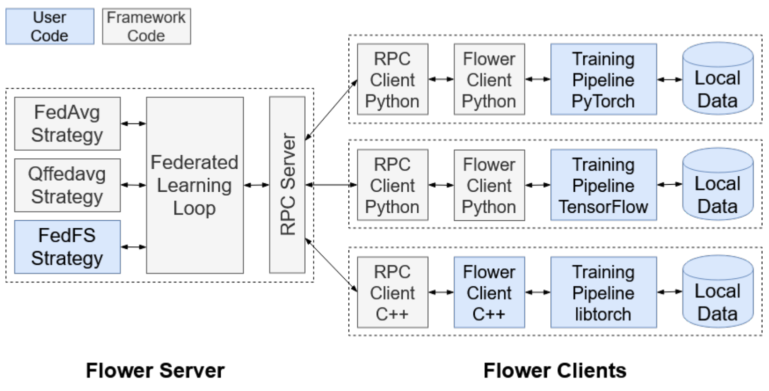

Figure 4.5: This picture shows the flower networking architecture.

4.6 Conclusion

In this chapter, we explored several known frameworks and libraries for Federated Learning. For each of them we provided an overview of their capabilities and their components. After a thorough comparison, we selected Flower as the most suitable framework for this project. It was considered the best choice due to its flexibility, framework-agnostic design, and ease of integration with popular machine learning libraries such as PyTorch and TensorFlow. Moreover, its ability to handle Federated Learning across different environments, including edge devices, aligns with the needs of this project. Finally, the well maintained documentation and the activity community are additional benefits.

[1]gRPC is a remote procedure call (RPC) framework. It uses HTTP/2 and Protocol Buffers.

Chapter 5

Enabling Federated Learning for evolving scenarios

In this chapter, we propose a solution to accommodate changes in model architecture during training within Federated Learning.

In section 5.1, we present some scenarios with a model's architecture evolving over time. These scenario are not strictly related to our cybersecurity context, but they are considered to approach the problem. In particular, we focus on i-DarkVec since it is one of the work which inspired this thesis.

In section 5.2 we propose an high-level solutions to be used in scenarios previously identified.

Then, in section 5.3, we analyze how Flower, chosen as Federated Learning framework, handles model's architectures during training. In particular, we identify its limitations. Finally, in section 5.4, we implement our solution within the Flower library. We introduce additional data structures and new functions to accommodate dynamic model updates.

5.1 Dynamic Model's Architecture

5.1.1 i-DarkVec Scenario

In this section we present a scenario based on i-DarkVec. As already explained in section 3.2.1, i-DarkVec is a Machine Learning model which leverages Word2Vec. However, i-DarkVec method is in contrast with the traditional method used to define a Word2Vec model. The latter one defines the model by using the number of words in the vocabulary, which were already been defined a-priori. This approach is not possible in our context for two main reasons. In one hand, we don't know all the possible IPs to be encountered during our traffic capture. In the other hand we can not define an embedding size of all possible IPs addresses. Note that IPv4 has $2^{32} \approx 4^9$ possible addresses, and IPv6 has $2^{128} \approx 3^{38}$ possible addresses. For this reason, i-DarkVec adds a new IPs to the vocabulary once detected. This results in an update of the embedding of the model i.e. an update on the model's architecture. The approach related to i-DarkVec is shown in Figure 5.1.

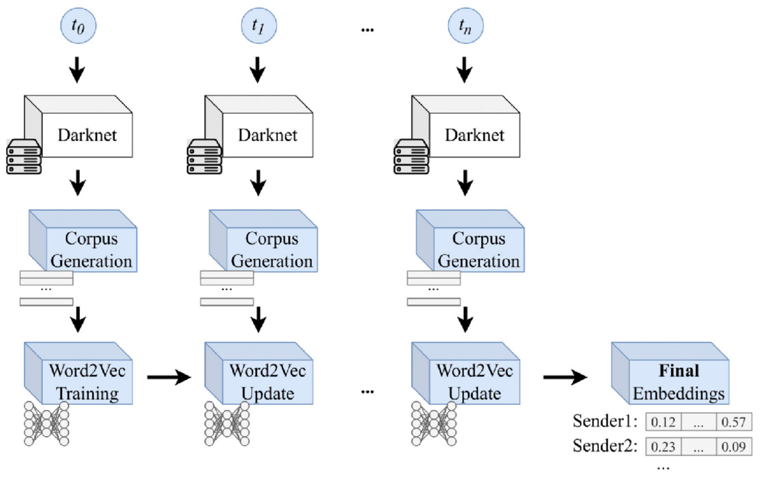

Figure 5.1: The figure shows the i-DarkVec incremental approach. Credits: (10).

i-Darkvec & Federated Learning

The i-DarkVec approach works well when the model is trained on a single node. Novel problems arises going federated. In this case, dataset is distributed across different clients, and each client has a unique set of IP addresses. As a result, each client creates has its own local vocabulary, and it creates the model's embeddings based on their own dataset.

With Federated Learning, each client sends its model parameters to the server for aggregation. However, the size of model's parameters varies between clients because each one builds the model based on its own vocabulary. This makes more difficult for the server to aggregate the parameters since vocabularies are different. The solution proposed is to establish a common vocabulary for clients and the server, and so a common architecture is determined. This makes easier to merge the same embeddings related to different models.

5.1.2 Other Scenarios

In this section, we present other scenarios where model's architecture may change per clients or server requests. These scenarios are listed below.

- In an online topology optimization, the server may decide to change the hidden layers of the topology e.g. online hyperparameter optimization.
- In continuous learning, a client may request to add a new class, so that the output layer can grow e.g. new application in traffic classification.
- In word-embedding with unknown dictionary, the clients may request to add new tokens, so that the internal connections and the input and output need to change.

5.2 Changes Proposed for Federated Learning

In this thesis we addressed the problem related to changes in model's architecture in the context of Federated Learning. Our idea consist of an additional step before the usual training phase. The server asks clients to send in their proposals for model's update. The format of the proposals depends on the specific use case. These proposals contains request i) to add nodes, ii) to remove nodes, or iii) to tweak the parameters in the network. Once the server has received these proposals, they are used to refine the model accordingly. This process of proposal's aggregation is tailored to the specific use case, and it may require a shared understanding of how features are mapped across clients. Finally, the server sends the architecture and the parameters of the updated model to the clients for training. The clients then update their local instance of the models with the new one received from the server and proceeds as usual in the training process. Once training is complete, the clients send back the updated parameters, and the server aggregates these parameter as usual.

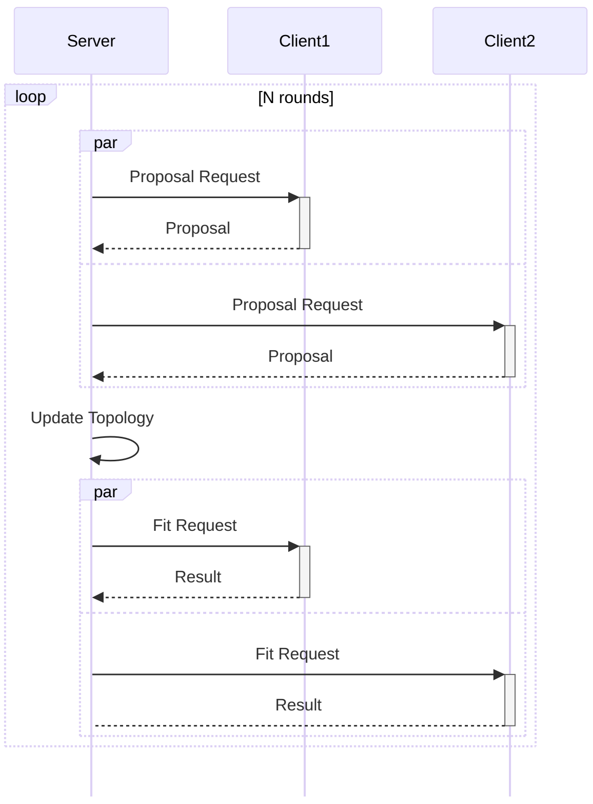

Figure 5.2: The figure shows our solution to support the model's architecture to changes.

5.3 Flower Current Limitations

Although Flower offers great flexibility and customization, it has some limitations described in the following list.

- There is the assumption that all `flower clients` use the same model architecture.
- If one `flower client` has a different model's architecture from other clients, then the joining process fails.
- One random picked client's model is used to initialize the models of all the other clients.
- `Flower server` role is limited to aggregate the parameters received from `flower clients`.
- The messages used by `flower server` to communicate with `flower clients` offers limited customization.
- `Flower server` is the only one to have the privileges to initiate interactions. `Flower clients` remain idle until prompted by the server and cannot send data unless it is for a reply for a server's request.

5.4 Flower Implementation of the Changes Proposed

In this section we describe our solution based on Flower.

In this section, we analyze our implementation proposed in Section 5.2. We first examine how the Flower library currently manages client-server operations, and then we describe the integration of our solution into the library.

5.4.1 Data Structures

In this section we present our proposed implementation for data structures. The file `trasport.proto` contains the format's definition of the client-server messages to be exchanged. This definition is given using Protocol Buffers[1]. This file is compiled using a simple application provided by Flower, known as `flwr_tool.protoc`. More details on the compilation process are presented in (7).

The four types of messages implement are presented within this section.

Server request for proposal - `PropIns`

`PropIns` is the message sent by the server to the clients to ask for their proposal. When a client receives this message, it has to formulate the proposal. We decided to include the possibility to insert some optional configurations for clients to customize the way they generate a proposal. Two examples are to ask for features that appear more than a certain number of times, or to limit the number of features returned.

[1]Protocol Buffers are language-neutral, platform-neutral extensible mechanisms for serializing structured data.

Client reply with the proposal - `PropRes`

`PropRes` is a message sent by clients to reply to `PropIns` message. The message contains the proposals of the client and the status of the reply. The latter is used by Flower itself for error signaling and it is present in all the reply messages.

Server request for training with the new Model - `ArchitectureFitIns`

`ArchitectureFitIns` is the message sent by the server to the clients to start the training process. This is sent after the server has aggregated their proposal and updated the aggregated model. The fields within `ArchitectureFitIns` are similar to the original Flower`FitIns` message. `ArchitectureFitIns` contains the parameters and the architecture of the aggregated model, the vocabulary as defined by the server with an word-index map, and a optional fields related to hyperparameters configuration. This map is required for clients to map their input data to the corresponding indexes. Note that in i-DarkVec case, the word are IPs.

Client response after training with the new model - `FitRes`

`FitRes` is a message sent by the clients to respond after local training. This message contains the information related to local trained model. The message is already implemented in Flower, and no changes were required.

5.4.2 Server Side Library

The server-side library customization focuses on the implementation of methods to request client's proposals, to update the model after the proposals have been received, and to instruct the clients to train using the updated model.

Flower implementation of the Federated Learning server-side process is structured as a loop. At each iteration, the server requests clients to train the model, and subsequently asks for an evaluation of the aggregated model. We modified this process by introducing new functions within the loop. These functions request client proposals, and update the model with these proposals before training starts. The implementation is outlined in Algorithm 1, and the methods are described in this section.

Request the client's proposal - `request_proposal_round()`

`request_proposal_round()` is the method used to request the client's proposal. It is shown in Algorithm 2.

The initial phase selects the clients which will participate in the operation, and it configures necessary settings. Once this preparation is complete, a new thread is created for each selected client to allow interactions with it. Each thread executes a function that effectively contacts the client. This function is responsible for sending the serialized `PropIns` request to the client. Once the requests is sent, the function waits for the client's reply, which is deserialized once receipt. Note that serialization and deserialization use the Protocol Buffer format. Finally, the thread returns the reply. When the replies from

Algorithm 1 The algorithm shows the basic operations of Federated Learning made by the Flower server.

```
1: function PERFOM_FED_LERN                          ▷ It is called once server is activated.
2:     for $i$ < Round_Number do
3:         REQUEST_PROPOSAL_ROUND()                  ▷ It asks proposals to clients.
4:         COMPUTE_NEW_TOPOLOGY()                    ▷ It updates model.
5:         FIT_ROUND()                               ▷ It asks client to train the mode.l
6:         To aggregate parameters;
7:         EVALUATE_ROUND()                          ▷ It asks client to evalutate the model.
8:     end for
9: end function
```

all the clients are received, they are collected and passed to the function responsible for computing the new model's architecture.

Algorithm 2 The algorithm show how proposals are used to update the model.

```
1: function REQUEST_PROPOSAL_ROUND
2:     To select clients for the operation;
3:     To set configuration and parameters;
4:     for each client do                            ▷ Each thread is associated with one client.
5:         To spawn a thread with the TOPOLOGY_UPDATE() task;
6:     end for
7:     To collect reply from clients;
8: end function

1: function TOPOLOGY_UPDATE                          ▷ This function is called for each client.
2:     proto_msg ← SERIALIZE(req_msg)                ▷ It serializes the message.
3:     reply ← REQUEST(proto_msg)                    ▷ It send the message to the client.
4:     reply_msg ← DESERIALIZE(reply)                ▷ It deserializes the reply.
5: end function
```

Update model architecture - `compute_new_topology()`

`compute_new_topology()` is the method which updates model's architecture. It is shown in Algorithm 3.

The method uses the proposals received from the clients to modify the model. The details of the implementation is up to the developer, since it depends on the application context.

Train using the updated model - `fit_round()`

`fit_round()` is the method which ask the clients to train the model. This function is already implemented in Flower, but it has been edited to use the message `ArchitectureFitIns`,

Algorithm 3 The algorithm shows how proposals are used to update the model.

```
1: function COMPUTE_NEW_TOPOLOGY
2:     changes ← proposals received from clients;
3:     for each change do
4:         To update model architecture using the proposed change;
5:     end for
6: end function
```

instead of `FitIns` of the original implementation.

5.4.3 Client Side Library Implementation

The Flower library's client-side implementation is responsible for receiving messages from the server, for processing them, and for replying with the results. It operates within a loop. The implementation is outlined in Algorithm 4.

Once a message is received from the server, it is deserialized using the Protocol Buffer format. The message is processed by extracting its type, such as `PropIns` or `ArchitectureFitIns`. The corresponding local function defined in the `Flower client` is invoked based on the message type.

Our development in the client-side library focuses on adding the support for a new message's type. We connected the new message's type with the appropriate `flower client` function, which implements the required operation. The function is implemented by leveraging the original `flower client` abstract classes. Furthermore, a small adjustment is made to the `flower client` function responsible for training the model. The modified function handles the new message type `ArchitectureFitIns`, which contains both the parameters and the architecture of the model, instead of the only `FitIns` message which includes just the parameters.

Algorithm 4 The algorithm shows how clients handles messages.

```
1: function START__CLIENT__INTERNAL()   ▷ It is invoked when the Flower client starts.
2:     To connect to the server;
3:     while the client is connected to the server do
4:         message ← RECEIVE()   ▷ It waits and it deserialize the message received from the server.
5:         reply ← PROCESS(message)
6:         SEND(reply)   ▷ It serializes and it send the message to the server.
7:     end while
8: end function

1: function PROCESS(message)
2:     message_type ← message.type   ▷ It extracts the message type;
3:     To call the corresponding method according to the message_type;
4:     result ← CLIENT.FN(message)   ▷ It calls the local function.
5: end function
```

5.5 Conclusion

In this chapter we presented some scenario in which the model's architecture evolves over time. In particular, we focused on i-DarkVec's model. We proposed an high-level solution to support the changes of the model's architecture in the Federated Learning. We discussed our implementation based on Flower, the framework we chose to manage Federated Learning.

Finally, we were able to use this library to build applications that utilize Federated Learning to train models with a dynamic architecture. Some examples of these applications are presented in Chapter 6.

Chapter 6

Results

In this chapter, we validate the enhanced Flower's implementation with two experiments by using a Convolutional Neural Network (CNN) for image classification, and i-Darkvec for malicious traffic identification.

We consider the first experiment, presented in 6.1, as initial benchmark to check the correctness of the new model's architecture update process.

The second experiment, presented in 6.2, is made using i-DarkVec. The aim of the experiment is to verify the correctness of i-DarkVec's model trained using Federated Learning.

6.1 Federated Image Classification

In this scenario, we use a simple Convolutional Neural Network (CNN) for image classification. The architecture of the CNN is shown in Figure 6.1. An important feature of this CNN is its final layer, which is responsible for classifying the image labels. The size of this layer corresponds to the number of labels in the dataset, making it straightforward to update and to add new classes. The CNN can divided in to two main parts. The first part handles the feature extraction, by identifying important details in the image.The second part performs the classification, assigning a label to the image.

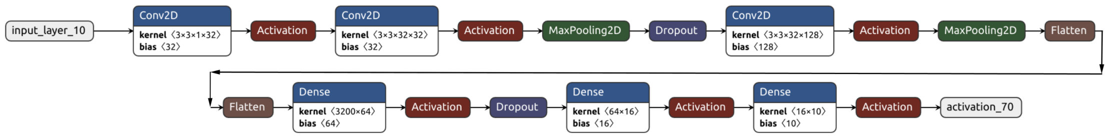

Figure 6.1: The figure shows the CNN model used for image classification.

The images we use for training/validating the model come from the MNIST dataset (16), which is a large set of handwritten digits. Some images are shown in Figure 6.2.

When working with a CNN model for images classification, we typically define the last layer in advance. This layer has 10 output classes, one for each digit from 0 to 9. For our

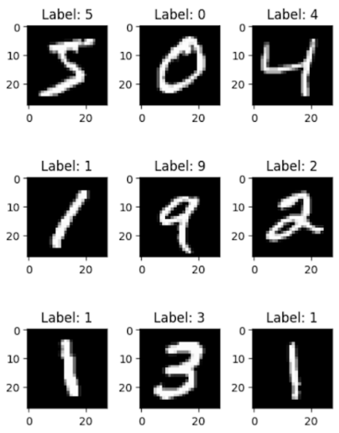

Figure 6.2: The figure shows some handwritten digits taken from the MNIST dataset.

experiment, we make the size of the last layer to growth as new labels are proposed by clients, instead to define the last layer at the beginning with already 10 classes. Note that the hidden layers, responsible for features extraction, maintain the same architecture.

6.1.1 Description

For our experiment, we use two clients, each with half of the MNIST dataset. We define the procedure on how client discover new labels from the dataset. This procedure consists of each client splitting its own dataset to generate 10 subsets containing the same digits. At each Federate Learning's iteration, a client adds, to its own dataset, one of the subsets obtained at the beginning. Then, the client inform the server about the labels contained in the actual dataset, and the server updates the CNN classification layer accordingly to the labels received. At this point, the client receives the models, and it trains the updated model using its own dataset.

We propose several ways for a client to choose the subset's labels to be added in its dataset. As example, two clients could propose the same label over the iterations, or they could choose their label randomly.

We perform some benchmark with different label's discovery strategies. The results are presented in Section 6.1.2, and they are analyzed in Section 6.1.3.

6.1.2 Results

In this section we present the experiment's results in Table 6.1, Table 6.2, Table 6.3, and Table 6.4.

All labels known in advance					
Round	Local model		Aggregated model		Labels
	Loss	Accuracy	Loss	Accuracy	
1	0.0828	0.9755	0.3107	0.9695	[0,1,2,3,4,5,6,7,8,9]
2	0.0551	0.9827	0.0442	0.9862	[0,1,2,3,4,5,6,7,8,9]
3	0.0506	0.9847	0.0313	0.9903	[0,1,2,3,4,5,6,7,8,9]
4	0.0341	0.9885	0.0239	0.9915	[0,1,2,3,4,5,6,7,8,9]
5	0.0285	0.9907	0.0253	0.9911	[0,1,2,3,4,5,6,7,8,9]
6	0.0245	0.9920	0.0191	0.9943	[0,1,2,3,4,5,6,7,8,9]
7	0.0260	0.9921	0.0209	0.9938	[0,1,2,3,4,5,6,7,8,9]
8	0.0323	0.9905	0.0234	0.9927	[0,1,2,3,4,5,6,7,8,9]
9	0.0313	0.9914	0.0212	0.9934	[0,1,2,3,4,5,6,7,8,9]
10	0.0210	0.9943	0.0168	0.9954	[0,1,2,3,4,5,6,7,8,9]

Table 6.1: In this scenario, all labels are known in advance and the model's architecture, once defined, is keep fixed for all ten rounds. The clients train the CNN using all subsets since the first round. The table shows the metrics's performance before and after the model aggregation. The performance of this scenario is used as a reference for other scenarios.

Same Labels Discovered					
Round	Local model		Aggregated model		Labels
	Loss	Accuracy	Loss	Accuracy	
1	0.0000	1.0000	0.0000	1.0000	[0]
2	0.0036	0.9991	0.0160	0.9962	[0, 1]
3	0.0207	0.9930	0.0179	0.9936	[0, 1, 2]
4	0.0136	0.9954	0.0141	0.9949	[0, 1, 2, 3]
5	0.0255	0.9934	0.0229	0.9944	[0, 1, 2, 3, 4]
6	0.0326	0.9910	0.0264	0.9935	[0, 1, 2, 3, 4, 5]
7	0.0432	0.9868	0.0440	0.9878	[0, 1, 2, 3, 4, 5, 6]
8	0.0544	0.9839	0.0450	0.9878	[0, 1, 2, 3, 4, 5, 6, 7]
9	0.0794	0.9797	0.0921	0.9819	[0, 1, 2, 3, 4, 5, 6, 7, 8]
10	0.1279	0.9726	0.1691	0.9726	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Table 6.2: In this scenario, the two clients propose the same label at each round e.g., at round 1, each client propose the labels 0. The table shows the metrics's performance before and after the model aggregation.

Orthogonal label discovery					
Round	Local model		Aggregated model		Labels
	Loss	Accuracy	Loss	Accuracy	
1	0.0000	1.0000	2.2882	0.5128	[0, 5]
2	0.0425	0.9904	0.9034	0.6741	[0, 5, 1, 6]
3	0.0162	0.9940	0.5041	0.9491	[0, 5, 1, 6, 2, 7]
4	0.0253	0.9920	0.6152	0.8700	[0, 5, 1, 6 ,2, 7, 3, 8]
5	0.0289	0.9922	0.8624	0.7968	[0, 5, 1, 6 ,2, 7, 3, 8, 4, 9]
6	0.0227	0.9926	0.4883	0.8518	[0, 5, 1, 6 ,2, 7, 3, 8, 4, 9]
7	0.0157	0.9950	0.2618	0.9285	[0, 5, 1, 6 ,2, 7, 3, 8, 4, 9]
8	0.0137	0.9960	0.2360	0.9309	[0, 5, 1, 6 ,2, 7, 3, 8, 4, 9]
9	0.0220	0.9940	0.2498	0.9223	[0, 5, 1, 6 ,2, 7, 3, 8, 4, 9]
10	0.0131	0.9962	0.2103	0.9341	[0, 5, 1, 6 ,2, 7, 3, 8, 4, 9]

Table 6.3: In this scenario, the first client has a subset of labels {0,1,2,3,4} different from the labels {5,6,7,8,9} of the second client. In this way, each client propose a label that other client does not have. After five rounds, the two clients have proposed all their subset's labels. In the following rounds any, new label is propose. The table shows the metrics performance before and after the model aggregation.

Random label discovery					
Round	Local model		Aggregated model		Labels
	Loss	Accuracy	Loss	Accuracy	
1	0.0000	1.0000	0.6296	0.5683	[3, 2]
2	0.0363	0.9857	0.8939	0.6559	[3, 2, 4, 5]
3	0.0413	0.9882	0.7585	0.7662	[3, 2, 4, 5, 9, 8]
4	0.0409	0.9850	0.9615	0.6771	[3, 2, 4, 5, 9, 8, 6]
5	0.0458	0.9850	0.4387	0.8720	[3, 2, 4, 5, 9, 8, 6]
6	0.0458	0.9883	0.9004	0.7458	[3, 2, 4, 5, 9, 8, 6, 1]
7	0.0543	0.9833	1.0324	0.6865	[3, 2, 4, 5, 9, 8, 6, 0, 1, 7]
8	0.0430	0.9867	0.4609	0.8310	[3, 2, 4, 5, 9, 8, 6, 0, 1, 7]
9	0.0443	0.9881	0.1592	0.9461	[3, 2, 4, 5, 9, 8, 6, 0, 1, 7]
10	0.0377	0.9898	0.0306	0.9916	[3, 2, 4, 5, 9, 8, 6, 0, 1, 7]

Table 6.4: In this scenario, the two clients randomly pick a new label for each round. In some round the label proposed are already proposed by the other client. The table shows the metrics performance before and after the model aggregation.

6.1.3 Comments

In this section, we compare the different scenarios presented in the Section 6.1.2. We analyze their metric's performance obtained in terms of loss and accuracy.

Fixed labels vs Dynamic labels

The first comparison is done between Table 6.1 and Table 6.2. This comparison is shown in Figure 6.3.

In the scenario of Table 6.1, we observe that the model starts with low accuracy and a high loss. However, during training, the accuracy improves, and the loss decreases. This behavior aligns with expectations.

In the scenario 6.2, we observed the opposite behaviour i.e., the model starts with a very high accuracy and low loss. In the initial rounds, the number of labels is limited, which allows the model to easily identify images, even though the hidden layers are not well-trained. However, as new labels are added, the model accuracy decrease. This occurs due to the introduction of additional labels which complicate the identification process.

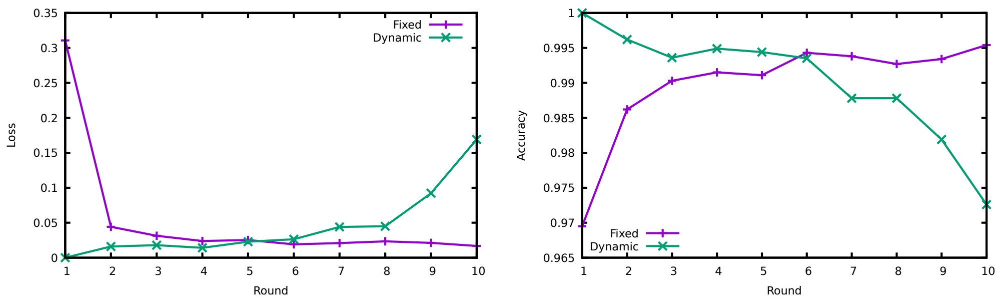

Figure 6.3: The figure shows the comparison between the scenario of Table 6.1 and Table 6.2. We refer to scenario of Table 6.1 as fixed, and to the one of Table 6.2 as dynamic. We compare the loss on the left picture, and the accuracy on the right picture.

Dynamic labels discovery

The second comparison is between the three different dynamic approach Table 6.2, Table 6.3, and in Table 6.4. This comparison is shown in Figure 6.4.

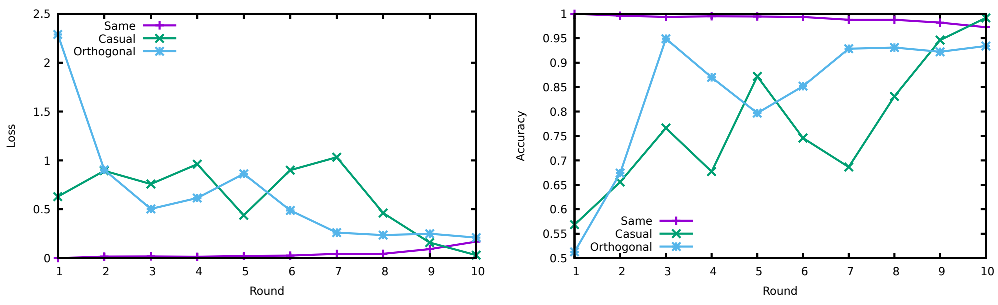

Figure 6.4: The figure shows the comparison between the scenario of Table 6.2, Table 6.3, and Table 6.4. We refer to the scenario of Table 6.2 as same, to the one of Table 6.3 as orthogonal, and to the one of Table 6.4 as casual. We compare the loss on the left picture, and the accuracy on the right picture.

6.2 Federated i-DarkVec

In this experiment, verify the correctness of i-DarkVec's model training in a Federated Learning context. For this experiment we use two clients. In Section 6.2.1, we present the dataset used for this experiment. In Section 6.2.2, we discuss how clients generate the vocabulary, and how the server aggregates them to update the models. In Section 6.2.3, we compare the performance metrics of the model trained using Federated Learning with the performance of a model trained using a traditional Machine Learning approach based on a single dataset. In section 6.2.4, we analyze the resources consumed during training e.g., RAM and CPU. This is done to understand if the model could be trained on resource-constrained edge devices. Finally, in Section 6.2.5, we look at how much time is taken for the vocabulary request's phase and to train the model. This is done to understand if the additional client-server interaction impacts on the overall timing of training.

6.2.1 Dataset

The dataset used to train i-DarkVec consists of traffic's capturs from two different networks. Each client utilizes different network traffic capture dataset. The datasets are stored in 31 files, one for each day of traffic's capture. The traffic's trace are normalized into the format presented in Table 6.5.

Timestamp	Source IP	Source port	Destination IP	Destination port	Protocol
1619827200.160742	92.63.196.13	57340	130.192.166.135	11676	6
1619827200.2047381	172.245.79.122	57026	130.192.166.46	23	6
1619827200.312591	1.85.44.226	38567	130.192.166.84	1433	6

Table 6.5: The table shows some example of data contained in the dataset used to train i-DarkVec's model.

6.2.2 Training

At each round of Federated Learning, the clients select one file corresponding to one day's traffic capture as their dataset. They generate the vocabulary, they send the vocabulary to the server, and from the server they obtain the model to be train for that round. Then, at each new round of Federated Learning, the clients choose the next file as their dataset, generating the new vocabulary for that round.

Vocabulary and models

In order to generate the vocabulary, we filtered the dataset with an approach explained in (9). The remaining IPs form the vocabulary sent from the client to the server. The server receives these vocabularies, and they are aggregated according to the Algorithm 5. The Figure 6.6 shows how many IPs are contained in the client vocabularies and, the size of the aggregated vocabulary created at each round. Note that the clients propose the vocabulary extracted from a single day of capture, while the aggregated vocabulary contains also the vocabularies of the previous days. It is important to notice that, there is no procedure to forget IPs. For this reason, the model's embedding continues to grow.

Figure 6.6 shows the size of the model after one round of Federated Learning and after thirty-one round of Federated Learning. The size of the embeddings for is 7842 on the first round, and it increase to 127664 on the thirty-one round. These sizes correspond also to the sizes of the aggregated vocabulary for that rounds.

Algorithm 5 Aggregation process of the vocabularies proposed by the clients.

```
1: IPs ← IPs contained in the clients' vocabularies received;
2: To sort the IPs;                                        ▷ Optional
3: for each IP in IPs do
4:     if IP not in vocabulary then
5:         To add IP to the vocabulary;
6:     end if
7: end for
```

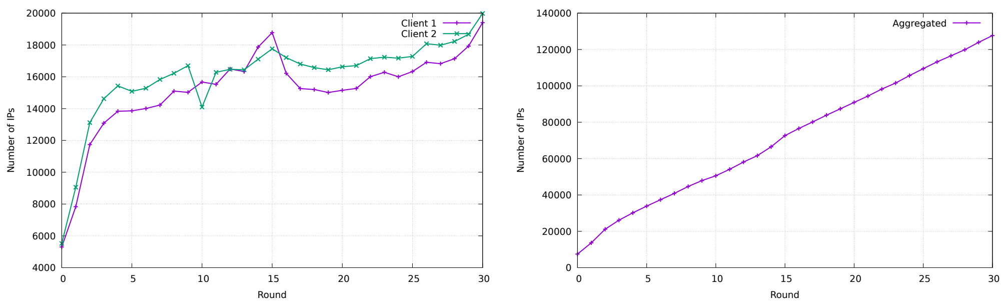

Figure 6.5: The picture on the left shows the size of the vocabularies proposed by clients during rounds. The picture on the right shows the size of the aggregated vocabulary on the server.

6.2.3 Federated Learning versus Traditional Machine Learning

In this section, we compare the performance of an i-DarkVec model trained using traditional Machine Learning with an i-DarkVec model trained using the Federated Learning. For the traditional Machine Learning training, we create a single dataset by combining the datasets used in Federated Learning approach. We train the Federated Learning model with 5 rounds. Each round represent a day of traffic capture. We compared the two approaches using a k-Nearest Neighbors (k-NN). The results are shown in Table 6.6.

We can notice that the performance obtained from the federated model is similar to the one obtained from the traditional approach. Moreover, we notice that the traditional model is able to classify more IPs than the federated model.

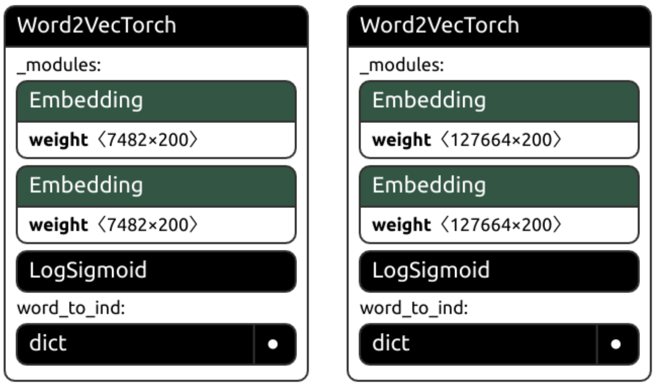

Figure 6.6: The picture on the left shows the i-DarkVec model after one round of Federated Learning. The picture on the right shows the i-DarkVec model after thirty-one round of Federated Learning.

Category	Traditional Machine Learning				Federated Learning			
	Precision	Recall	F1-Score	Support	Precision	Recall	F1-Score	Support
shadowserver	1.00	1.00	1.00	574.0	1.00	1.00	1.00	574.0
rapid7	1.00	1.00	1.00	344.0	1.00	1.00	1.00	344.0
censys	0.97	1.00	0.99	334.0	0.97	1.00	0.99	334.0
internetcensus	0.98	0.99	0.99	169.0	0.94	1.00	0.97	169.0
onyphe	0.99	0.99	0.99	147.0	0.98	0.98	0.98	147.0
netsystems	1.00	1.00	1.00	88.0	1.00	1.00	1.00	88.0
shodan	1.00	0.89	0.94	53.0	1.00	0.77	0.87	53.0
binaryedge	1.00	0.77	0.87	35.0	1.00	0.94	0.97	34.0
ipip	1.00	0.26	0.41	31.0	0.78	0.23	0.35	31.0
alphastrike	1.00	0.50	0.67	24.0	NaN	NaN	NaN	NaN
u_mich	1.00	1.00	1.00	20.0	0.95	1.00	0.98	20.0
recyber	1.00	0.14	0.25	14.0	1.00	0.29	0.44	14.0
pnap	1.00	1.00	1.00	6.0	1.00	1.00	1.00	6.0
cymru	0.00	0.00	0.00	3.0	0.00	0.00	0.00	3.0
intrinsec	0.00	0.00	0.00	2.0	0.00	0.00	0.00	2.0
u_caida_ark	1.00	1.00	1.00	2.0	0.00	0.00	0.00	2.0
arbor	0.00	0.00	0.00	2.0	0.00	0.00	0.00	2.0
errata	0.00	0.00	0.00	1.0	0.00	0.00	0.00	1.0
leakix	0.00	0.00	0.00	1.0	0.00	0.00	0.00	1.0
inversepath	0.00	0.00	0.00	1.0	0.00	0.00	0.00	1.0
cybergreen	0.00	0.00	0.00	1.0	0.00	0.00	0.00	1.0
threatsinkhole	0.00	0.00	0.00	1.0	0.00	0.00	0.00	1.0
Weighted Average	0.99	0.96	0.96	1853.0	0.98	0.96	0.96	1828.0
Macro Average	0.63	0.52	0.55	1853.0	0.55	0.49	0.50	1828.0
Micro Average	0.99	0.96	0.97	1853.0	0.98	0.96	0.97	1828.0

Table 6.6: The table shows the performance metrics obtained for an i-DarkVec model trained using traditional Machine Learning, and for an i-DarkVec model trained using Federated Learning approach.

6.2.4 Benchmark & Resource Usage

In this section we analyze the resource consumption during the training of i-DarkVec's model with 31 rounds of Federated Learning, and 2 clients involved.

RAM Usage

Figure 6.7 shows the 2 clients's and server's RAM consumption. Overall, the RAM consumption stays under 5 GB. If we look the behaviour of clients's RAM, we can identify the round of training and the curve change in correspondence to the start of the round. The consumption of RAM of the server is smaller than the one required by the clients, since the server keep in memory only the vocabulary and the model's parameters.

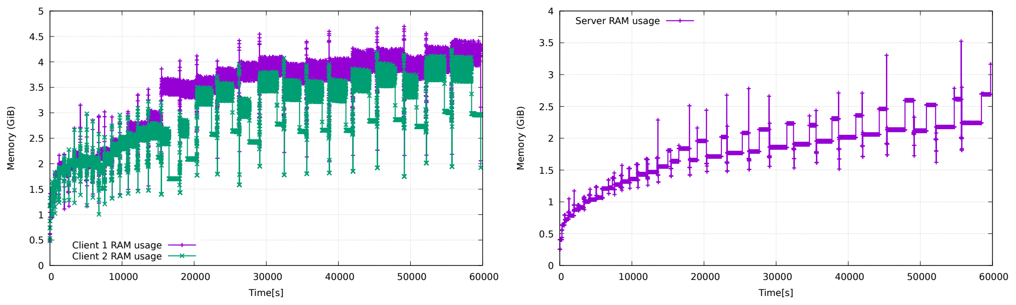

Figure 6.7: The picture on the left shows the RAM consumption of the two clients. The picture on the right shows the RAM consumption of the server.

CPU Usage

Figure 6.8 shows the consumption of CPU of clients and server. The peak of client's CPU usage occurs at the start of a new training round, since the dataset is being loaded. The lowest usage is observed at the end of the training round, when the clients are waiting for the server. The CPU usage of the server is very limited. the peak corresponds to the phase of proposal requests and during aggregation of model's parameters. Moreover, when the clients are training, the server's CPU consumption is close to 0.

Disk Usage

Figure 6.9 shows clients's disk usage. The disk is mainly used to read the dataset and to write logs. The read operations increase on the last rounds of training, due to the fact that i-DarkVec IPs filtering's process needs to read the files related to previous days, in order to remove rare IPs from the dataset. The write operations of the clients, are limited to write the logs of the Federated Learning process.

Figure 6.10 shows server disk's usage. The server has to write on disk the following data: clients's proposed vocabularies, aggregated vocabulary, clients's model parameters, aggregated model's parameters. The server does not performs any significant read operation from disk.

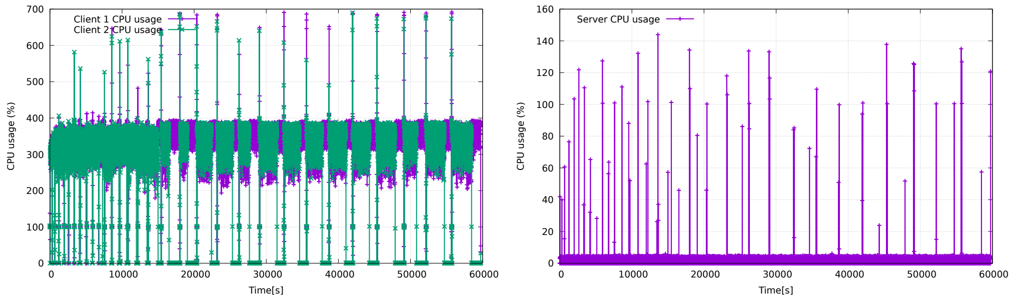

Figure 6.8: The picture on the left shows the CPU consumption of the two clients. The picture on the right shows the CPU consumption of the server.

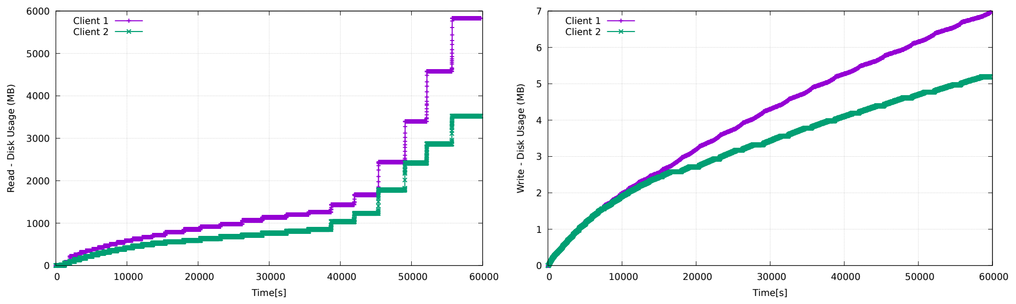

Figure 6.9: The picture on the left shows the disk usage to read operations by the client. The picture on the left shows the disk usage to write operation by the client

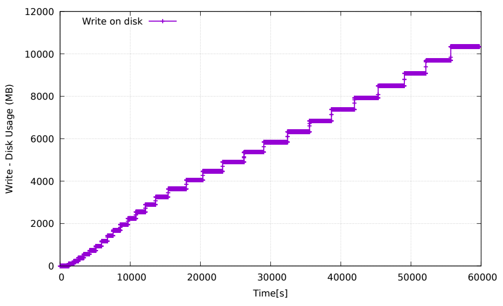

Figure 6.10: The figure shows the cumulative write disk operations performed by the server.

Network usage

Figure 6.11 shows the cumulative data sent over the network by the clients and server.

6.2.5 Operations Timing

In this section, we analyze how much time each client-server interaction takes to be performed. With this analysis, we understand if the additional phase introduced with our

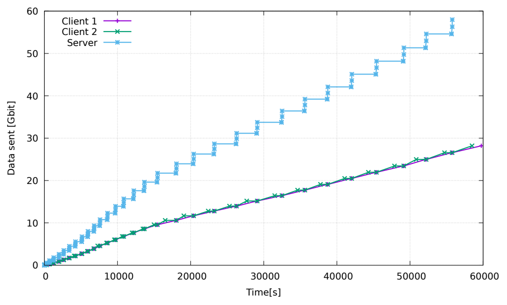

Figure 6.11: The figure shows the cumulative network usage for clients and server.

solution has an impact on the overall Federated Learning training timing.

Proposal request

The operation of generate a proposal request `PropIns` from the server takes 0.2 ms. The transmission and the consequent reception of `PropIns` takes less than 1 ms. The generation of a vocabulary from clients takes from 27 s to 94 s. This time is required to read the dataset, to filter and to extract the IPs. The transmission of `PropRes` with the vocabulary takes from 50 ms to 100 ms, depending on the size of the vocabulary. The aggregation of these vocabularies on the server takes less than 1 ms.Overall, the proposal request phase takes from 28 s to 95 s.

Training process

The operation of generate a training request `ArchitectureFitIns` from the server takes 1 ms. The transmission and the consequent reception of `ArchitectureFitIns` takes from 0.1 ms to 3 s. The training operation on the clients takes from 249 s to 2713 s. The transmission of `FitRes` with the model's parameters takes from 0.1 ms to 3 s. The aggregation of models's parameters on the server takes from 2 ms to 3 s. Overall, the training process phase takes from 250 s to 2720 s.

6.3 Conclusion

These experiments demonstrated the effectiveness of our proposed solution and its implementation using our enhanced Flower library. In particular, the i-DarkVec experiment's results offered several key insights listed below.

- The performance achieved is comparable to a traditionally trained model, confirming that the federated nature of our approach does not compromise accuracy or efficiency.
- The model's training does not request excessive resources, showing that the solution is feasible even in environments with limited computational resources.
- The proposal phase introduces minimal overhead, and it did not substantially impact the overall training time.

Chapter 7

Conclusions and Future Works

7.1 Conclusions

In this thesis, we presented an improvement of a distributed platform for network traffic's analysis, and we proposed a solution to use Federated Learning with dynamic Machine Learning models for the same scope.

The improvement of the distributed platform allows third-party organizations to join with their machines. Thanks to Ansible Playbooks we developed, the process to add these nodes was seamlessly. Morever, Helm Charts enabled us to easily control and deploy applications on the platform. The available applications within the platform allows i) to collect network's traffic, ii) to activate darknets and honeypots, and iii) to use the application we developed to analize the captured traffic using a Machine Learning model with Federated Learning.

We successfully developed an application capable of training ML models with dynamic architectures using Federated Learning. This application was developed upon an existing framemork, known as Flower, which we selected conducting a survey. This application is capable of collect proposals regarding model's update from the clients, to update the model accordingly, and to distribute this model to clients for further training.

We validated the obtained application in two experiments to analyze the behavior of the trained models collecting metrics. The first experiment was conducted using a Convolutional Neural Network (CNN) to classify images. The results show us that our solution works fine. The second experiment was conducted using i-DarkVec. We performed an analysis of several aspects of the model's training process. We looked at how the vocabularies and embeddings grows during the Federated Learning rounds. We compared the Federated Learning model with another one trained using a traditional Machine Learning approach. Finally, we analyzed the resources consumption used during the training. We looked at how much time it takes the vocabularies proposal phase compared to the overall time required for training. The most important results are listed below.

- The size of i-DarkVec embeddings continues to grow. Although the embedding size is acceptable after some rounds of training, the model requires pruning techniques to maintain a reasonable embedding size over longer periods.

- The performance achieved was comparable to a traditional model. This provides the confirmation that the distributed approach does not compromise accuracy or efficiency.
- The model training does not request excessive resources, showing that the solution is feasible even in environments with limited computational resources.
- The proposal phase introduces minimal overhead, and it did not substantially impact the overall training time.

As conclusion, the results demonstrate effectiveness and quality of our proposed solution and the applications we developed.

7.2 Future Works

In this section, we list some of the possible future works.

The distributed platform should be tested capturing real network's traffic from the distributed nodes. Currently, only small and limited traffic's capture have been performed. Moreover, it is required to implement pre-processing phase to trasform captured data in a proper format to train i-DarkVec's model in real-time.

We employed a limited kinds of honeypots within the platform. Others kinds of honeypots can be integrated into our platform to improve the quality and insight of captured traffic. Moreover, the reliability of nodes in the platform can be investigated.

Altought we made our analysis by only employing i-DarkVec's model, other Machine Learning models can be introduced. An example is DANTE (4).

The Federated Learning application we developed could be optimized by reducing the amount of data sent during various client-server interactions. Currently, the impact is quite significant, especially in the i-DarkVe context where the models and vocabularies grow rapidly.

Bibliography

[1] Ansible, Inc. Ansible documentation, 2024. URL https://docs.ansible.com/.

[2] Subrato Bharati, M. Rubaiyat Hossain Mondal, Prajoy Podder, and V. B. Surya Prasath. Federated learning: Applications, challenges and future directions. *International Journal of Hybrid Intelligent Systems*, 18(1-2):19–35, 2022. doi: 10.3233/HIS-220006. URL https://arxiv.org/abs/2205.09513.

[3] Google Cloud. Artificial intelligence vs. machine learning, n.d. URL https://cloud.google.com/learn/artificial-intelligence-vs-machine-learning?hl=en. Accessed: 2024-10-07.

[4] Dvir Cohen, Yisroel Mirsky, Yuval Elovici, Rami Puzis, Asaf Shabtai, Manuel Kamp, and Tobias Martin. Dante: A framework for mining and monitoring darknet traffic. *Department of Software and Information Systems Engineering, Ben Gurion University of the Negev*, 2024.

[5] NVIDIA Corporation. Nvidia flare documentation. https://nvflare.readthedocs.io/en/main/index.html.

[6] TensorFlow Developers. Tensorflow: An end-to-end open source machine learning platform, 2023. URL https://www.tensorflow.org/. Accessed: 2024-10-12.

[7] Flower Framework. Contributor guide: How to create new messages, 2024. URL https://flowerai.net/docs/framework/contributor-how-to-create-new-messages.html.

[8] Alejandro Ayala Gil. Honeypot in a box: A distributed cluster network for honeypot deployment. Master's thesis, Politecnico di Torino, Corso di laurea magistrale in Ict For Smart Societies (Ict Per La Società Del Futuro), 2024. Relatori: Marco Mellia, Idilio Drago.

[9] Luca Gioacchini, Luca Vassio, Marco Mellia, Idilio Drago, Zied Ben Houidi, and Dario Rossi. DarkVec: Automatic Analysis of Darknet Traffic with Word Embeddings. pages 76–89, December 2021. doi: 10.1145/3485983.3494863. URL https://doi.org/10.1145/3485983.3494863.

[10] Luca Gioacchini, Luca Vassio, Marco Mellia, Idilio Drago, Zied Ben Houidi, and Dario Rossi. i-darkvec: Incremental embeddings for darknet traffic analysis. *ACM Trans.*

Internet Technol., 23(3), August 2023. ISSN 1533-5399. doi: 10.1145/3595378. URL https://doi.org/10.1145/3595378.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[12] Cloud Hacks. Federated learning: A paradigm shift in data privacy and model training, n.d. URL https://medium.com/@cloudhacks_/federated-learning\-a-paradigm-shift-in-data-privacy-and-model-training-a41519c5fd7e. Accessed: 2024-10-07.

[13] Andrew Hard, Chloé M Kiddon, Daniel Ramage, Francoise Beaufays, Hubert Eichner, Kanishka Rao, Rajiv Mathews, and Sean Augenstein. Federated learning for mobile keyboard prediction, 2018. URL https://arxiv.org/abs/1811.03604.

[14] Helm Project. Helm documentation, 2024. URL https://helm.sh/docs/.

[15] Ivan Kholod, Evgeny Yanaki, Dmitry Fomichev, Evgeniy Shalugin, Evgenia Novikova, Evgeny Filippov, and Mats Nordlund. Open-source federated learning frameworks for iot: A comparative review and analysis. *Sensors*, 21:167, 12 2020. doi: 10.3390/s21010167.

[16] Yann LeCun, Corinna Cortes, and Christopher J. C. Burges. The mnist database of handwritten digits, 1998. URL http://yann.lecun.com/exdb/mnist/. Accessed: 2024-10-15.

[17] Y. Liu, T. Fan, T. Chen, Q. Xu, and Q. Yang. Fate: An industrial grade platform for collaborative learning with data protection. 2021.

[18] Yi Liu. Federated learning frameworks: From research to industry. Master's thesis, 03 2022.

[19] Gianpaolo Reina, Alexey Gruzdev, Patrick Foley, Olga Perepelkina, Igor Davidyuk, Ilya Trushkin, Maksim Radionov, Aleksandr Mokrov, Dmitry Agapov, Jason Martin, Brandon Edwards, Micah Sheller, Sarthak Pati, Prakash Moorthy, Shih-han Wang, Prashant Shah, and Spyridon Bakas. Openfl: An open-source framework for federated learning. *arXiv preprint arXiv:2105.06413*, 2021. doi: 10.48550/arXiv.2105.06413. URL https://arxiv.org/abs/2105.06413.

[20] Sukhveer Sandhu, Hamed Taheri Gorji, Pantea Tavakolian, Kouhyar Tavakolian, and Alireza Akhbardeh. Medical imaging applications of federated learning. *Diagnostics*, 13:3140, 10 2023. doi: 10.3390/diagnostics13193140.

[21] Simon Stojanovic. Flower & pysyft – federated learning frameworks in python. https://medium.com/elca-it/flower-pysyft-co-federated-learning-frameworks-in-python-b1a8eda68b0d, 2023. Accessed: 2024-09-30.

[22] Flower Team. Flower: A friendly federated learning framework, n.d. URL https://flower.ai/. Accessed: 2024-10-07.

[23] Dong Yang, Ziyue Xu, Wenqi Li, Andriy Myronenko, Holger R. Roth, Stephanie Harmon, Sheng Xu, Baris Turkbey, Evrim Turkbey, Xiaosong Wang, Wentao Zhu, Gianpaolo Carrafiello, Francesca Patella, Maurizio Cariati, Hirofumi Obinata, Hitoshi Mori, Kaku Tamura, Peng An, Bradford J. Wood, and Daguang Xu. Federated semi-supervised learning for covid region segmentation in chest ct using multi-national data from china, italy, japan. *Medical Image Analysis*, 70:101992, 2021. ISSN 1361-8415. doi: https://doi.org/10.1016/j.media.2021.101992. URL https://www.sciencedirect.com/science/article/pii/S1361841521000384.

[24] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications, 2019. URL https://arxiv.org/abs/1902.04885.