UNIVERSITÀ
DEGLI STUDI
DI PALERMO

NDS
NETWORKING AND DISTRIBUTED SYSTEMS
RESEARCH GROUP

Riconoscimento di Anomalie nel Traffico di Rete Generato da Sistemi IoT

Tesi di Laurea Magistrale in Ingegneria Informatica

Domenico Giosuè
Relatore: Prof. Giuseppe Lo Re
Correlatore: Dott. Andrea Augello

UNIVERSITÀ DEGLI STUDI DI PALERMO

Dipartimento di Ingegneria

Laurea Magistrale in Ingegneria Informatica

RICONOSCIMENTO DI ANOMALIE NEL TRAFFICO DI RETE GENERATO DA SISTEMI IOT

Tesi di Laurea di:
Domenico Giosuè

Relatore:
Prof. Giuseppe Lo Re

Controrelatore:
-

Correlatore:
Dott. Andrea Augello

ANNO ACCADEMICO 2023/2024

UNIVERSITÀ DEGLI STUDI DI PALERMO
DIPARTIMENTO DI INGEGNERIA

LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

RICONOSCIMENTO DI ANOMALIE NEL TRAFFICO DI RETE GENERATO DA SISTEMI IOT

Tesi di Laurea di
Domenico Giosuè

Relatore:
Prof. Giuseppe Lo Re

Controrelatore:
Prof. Valeria Seidita

Correlatore:
Dott. Andrea Augello

Sommario

L'utilizzo delle tecnologie informatiche, nonostante sia a oggi fondamentale, espone a una serie di rischi dovuti alla presenza di cybercriminali che sfruttano la comunicazione mediante la rete internet al fine violare la sicurezza dei sistemi. Nel corso degli anni, l'affinamento delle tecniche per l'individuazione dei tentativi di attacchi informatici ha portato all'integrazione degli algoritmi di machine learning nei più moderni meccanismi di sicurezza.

Un caso particolare riguarda i sistemi di Internet of Things (IoT), critici per il funzionamento delle infrastrutture aziendali, ospedaliere e cittadine. Tali sistemi spesso producono e gestiscono dati sensibili, motivo per cui possono essere oggetto di attacchi informatici che mirano all'esfiltrazione dei dati. Proprio per la necessità di garantire alti livelli di privacy, è preferibile non esporre i dati generati da questi sistemi ad analisi esterne, basate per esempio su architetture cloud. I sistemi IoT, tuttavia, contano su risorse limitate e non sono quindi in grado di implementare intere pipeline di machine learning.

Viene proposto un sistema di rilevamento delle anomalie in grado di identificare prontamente il traffico di rete tipico di un attacco informatico. Tale sistema è progettato per tenere conto della topologia dei sistemi IoT, del loro vincolo riguardo la limitatezza delle risorse e della necessità di garantire la privacy dei dati sul traffico di rete, sia durante l'addestramento del sistema che in fase di inferenza. Poiché il sistema è decentralizzato non è possibile avere un controllo sui dati utilizzati in fase di addestramento e, di conseguenza, non si può essere certi che i dispositivi partecipanti non siano già compromessi. Per mitigare il problema derivante dalla presenza di dispositivi infetti durante la fase di addestramento, il sistema si basa su un'architettura di machine learning robusta ad attacchi di avvelenamento dei dati.

I risultati ottenuti dall'architettura proposta mostrano come questa sia in grado di superare le alternative allo stato dell'arte più recenti, mantenendo un focus sulle criticità dei sistemi IoT e rimanendo allo stesso tempo flessibile.

Indice

Capitolo 1

Introduzione

Il tema della cybersicurezza risulta essere critico per tutte quelle aree che fanno affidamento, anche indirettamente, sulle tecnologie informatiche. In seguito all'evoluzione delle tecniche utilizzate per condurre cyberattacchi, gli strumenti difensivi esistenti hanno perso la loro efficacia nell'individuazione delle azioni malevole perpetrate verso i sistemi; per questo motivo è necessario proporre nuove metodologie difensive che possano bloccare o mitigare i danni dovuti ai suddetti attacchi. L'alta complessità delle moderne tecniche di attacco, contemporaneamente al progredire delle capacità di calcolo dei sistemi informatici, ha portato all'utilizzo di diversi strumenti di Intelligenza Artificiale, in particolar modo di Machine Learning, con la finalità di riuscire a contrastare attacchi di cui non si conoscono le caratteristiche.

Tra i vari sistemi informatici soggetti ad attacchi è possibile citare i sistemi di Internet of Things (IoT) [1]; ovvero, reti di dispositivi dotati di sensori, attuatori, risorse di calcolo e connettività di rete che abilitano la raccolta e lo scambio di informazioni. La possibilità di connettersi alla rete internet consente a questi dispositivi di essere controllati e monitorati da remoto utilizzando le infrastrutture di rete già esistenti.
Tra i requisiti di un sistema IoT [2] è presente il requisito della privacy, motivo per cui non sempre vi è interesse nello spostamento dei dati verso sistemi o dispositivi esterni all'ecosistema IoT. Nonostante non esista una specifica architettura standardizzata, l'architettura generale dei sistemi IoT prevede una suddivisione gerarchica basata su differenti livelli [3, 4] che dividono il livello dei dispositivi fisici dal livello di controllo e gestione delle informazioni da questi ricevute. In [4] viene fatto riferimento a diverse categorie di architetture tra cui quelle basate sul cloud [5, 6] e quelle basate sul fog computing [7–9], le quali consentono di spostare la fase di processamento dei dati al di fuori del sistema dove questi vengono acquisiti; tuttavia, nonostante tale topologia architetturale consenta di spostare le operazioni di calcolo al di fuori del sistema IoT, questa porterebbe a una perdita di privacy.

Seppure i requisiti di privacy e sicurezza possano essere considerati comuni a tutti i sistemi IoT, è bene distinguere i diversi sistemi in base alla loro criticità, ovvero in base a quanto risultino essere gravi i danni provocati da un loro malfunzionamento. Al giorno d'oggi i dispositivi IoT vengono utilizzati in diversi ambiti, tra cui quello medico [10], industriale [11] e domestico [12], ognuno dei quali può presentare diversi requisiti legati alla sicurezza del sistema e alla privacy dei dati.
I sistemi IoT possono essere infettati e utilizzati per condurre attacchi informatici distribuiti [13] (es. DDoS) oppure violati con la finalità di ottenerne i dati o negarne la disponibilità. Per questo motivo, risulta fondamentale l'utilizzo di strumenti appropriati per identificare la presenza di anomalie [14, 15], riconducibili ad attacchi informatici, partendo da un'analisi del traffico di rete associato ai diversi dispositivi, impedendo in questo modo che traffico dannoso sia indirizzato verso il sistema e i dispositivi che lo compongono. Il problema nel voler garantire la sicurezza

informatica di tali dispositivi risiede nella loro caratteristica di limitatezza delle risorse [16]. I dispositivi utilizzati nei sistemi IoT posseggono risorse limitate in termini di calcolo e di memoria, motivo per cui risulta difficile l'implementazione di intere pipeline di machine learning. La suddetta caratteristica sposta l'interesse verso l'utilizzo di approcci di addestramento distribuito [17] che permettano di addestrare modelli di machine learning suddividendone gli strati tra più dispositivi e consentendo così di distribuire efficientemente il carico di lavoro dell'intera rete. Inoltre, poiché un sistema IoT include diversi dispositivi, è opportuno sfruttare tecniche di addestramento collaborativo [18] al fine di strutturare un modello unico e rappresentativo per tutti i dispositivi. L'utilizzo delle tecniche di addestramento collaborativo distribuito, oltre che a un'efficienza tecnica, porta a grandi vantaggi per ciò che riguarda la confidenzialità dei dati presenti in ogni dispositivo. La distribuzione del modello su più nodi consente infatti di mantenere i dati all'interno di ogni dispositivo e inviare alle parti di modello successive solamente un output già processato. Analogamente, le tecniche di addestramento collaborativo prevedono di aggregare modelli già addestrati con la finalità di ottenere un modello globale rappresentativo senza la necessità di trasferire i dati di ogni modello verso un server centralizzato.

Il sistema proposto, a cui si farà riferimento con il nome "Anti Poisoning Split Federated Learning" (AP-SFL), fa uso di tecniche di addestramento non supervisionato al fine di poter apprendere il pattern del traffico di rete specifico dell'ecosistema IoT, adattandosi a questo. Tale sistema, inoltre, mira a distribuire l'onere computazionale tra i dispositivi mediante un approccio di Split Learning [19], raggiungendo alti livelli di privacy grazie all'utilizzo della tecnica di Federated Learning [20, 21].

Considerando la grande varietà di traffico di rete che potrebbe essere associato a un sistema IoT, è fondamentale ricorrere ad approcci di addestramento non supervisionato [22, 23] così da adattare il modello di machine learning al comportamento specifico del sistema, mantenendone comunque la struttura interna. Questa scelta, tuttavia, può generare problemi nel momento in cui alcuni dispositivi infetti partecipano alla fase di addestramento collaborativo; infatti, qualora uno o più dispositivi generassero traffico di rete tipico di un attacco, il sistema ne apprenderebbe il pattern e lo classificherebbe come traffico benevolo.

AP-SFL tiene conto del suddetto problema e presenta una nuova metodologia per mitigare l'impatto dei dispositivi infetti durante la fase di addestramento del modello. In particolare, grazie a una fase di filtraggio, AP-SFL esclude dalla fase di addestramento i dispositivi infetti, facendo in modo che il modello venga addestrato solamente sui dispositivi che generano traffico di rete normale. Le caratteristiche topologiche del sistema in questione garantiscono la privacy dei dati sul traffico di rete associato a ogni dispositivo e consentono un bilanciamento del carico di lavoro tra i dispositivi IoT e un server centralizzato.

I seguenti capitoli descrivono il problema, illustrando lo stato dell'arte e dettagliando il sistema proposto, confrontando quest'ultimo con le più recenti soluzioni presenti in letteratura.

Nel corso del Capitolo 2 sono illustrate le più recenti tecniche utilizzate nell'ambito del rilevamento di anomalie, con particolare riferimento alla loro applicazione su dispositivi a risorse limitate. Sono inoltre dettagliati i paradigmi di Split Learning, Federated Learning e le tecniche allo stato dell'arte per mitigare il fenomeno dell'avvelenamento dei dati. Nel Capitolo 3 viene proposta l'architettura di rilevamento delle anomalie AP-SFL, la quale risulta essere robusta alla presenza di client infetti durante la fase di addestramento. Il Capitolo 4 vede la presentazione dei risultati dei test sperimentali condotti sull'architettura AP-SFL e su alcune sue possibili varianti. Il Capitolo 5, infine, è dedicato alle considerazioni finali riguardo l'architettura proposta e le sue possibili applicazioni.

Capitolo 2

Stato dell'Arte

Questo capitolo vuole illustrare alcuni tra i principali approcci esistenti in letteratura per ciò che riguarda il rilevamento delle minacce all'interno di un sistema informatico, illustrando anche i paradigmi di Split Learning e Federated Learning. Viene inoltre affrontato il tema della mitigazione del fenomeno di data poisoning, con particolare attenzione ad alcune delle più recenti tecniche allo stato dell'arte.

2.1 Rilevamento delle minacce informatiche

Ogni sistema informatico è soggetto a numerose minacce che possono concretizzarsi in seguito all'individuazione, da parte di un soggetto malintenzionato, di una o più vulnerabilità del sistema. Per individuare le minacce informatiche possono essere utilizzati approcci differenti tra cui, per esempio, i sistemi di rilevamento basati su firme [24].

I sistemi di rilevamento basati su firme hanno come obiettivo il riconoscimento di specifiche caratteristiche proprie di un comportamento anomalo. La scelta delle caratteristiche da utilizzare per costruire la firma di un attacco è fondamentale affinché il sistema di rilevamento non sia facilmente aggirabile. Il seguente elenco specifica alcune delle più comuni caratteristiche utilizzate per generare la firma di un malware:

- Firma basata su hash, vede l'utilizzo del digest ottenuto applicando una funzione hash al file binario del programma. Questo approccio è però facilmente aggirabile da uno sviluppatore di malware che, modificando il codice sorgente del programma malevolo è in grado di cambiare totalmente la firma del software, rendendo questo irriconoscibile al sistema.
- Firma basata su sequenze di Byte [25], viene generata sfruttando delle sequenze di Byte specifiche del malware. Un malintenzionato può aggirare il controllo della firma offuscando le suddette sequenze inserendo, per esempio, istruzioni nulle (NOP) o cambiando l'organizzazione del codice mantenendo le funzionalità logiche originali.
- Firma basata sulla struttura [26], viene generata a partire dalla struttura del file malevolo, per esempio i suoi metadata, e non dal suo codice. Un malintenzionato può però aggirare i controlli cambiando la struttura del file, per esempio comprimendolo, rendendolo così non identificabile dal sistema.

La combinazione delle suddette caratteristiche può consentire la creazione di firme più robuste e complesse in grado di generalizzare la struttura dei malware. Tale approccio risulta essere estremamente efficace fino a quando la minaccia è conosciuta [27]; tuttavia, a oggi esistono molteplici attacchi informatici che sfruttano le cosiddette *vulnerabilità zero-day*, ovvero delle

vulnerabilità del software sconosciute e alle quali non è mai stato posto rimedio. Per aggirare la necessità di conoscere la firma di un attacco sono stati introdotti diversi sistemi che sfruttano approcci di machine learning per classificare il soggetto di analisi come benevolo o malevolo. Un esempio di utilizzo degli strumenti di machine learning è mostrato in [28] dove gli autori, partendo da un insieme di caratteristiche di basso livello, si pongono come obiettivo quello di rilevare la presenza di malware. Altri approcci, utilizzati per esempio dai *sistemi di rilevamento delle intrusioni* [29], che monitorano il traffico di rete di un sistema al fine di individuare potenziali minacce, possono prevedere l'utilizzo di politiche specifiche. Le suddette politiche possono vietare comportamenti non autorizzati, prevedere l'utilizzo di regole in grado di bloccare il traffico di rete proveniente da indirizzi IP e domini con scarsa affidabilità, oppure possono concentrarsi sul comportamento del protocollo utilizzato al fine di individuare particolari tipologie di attacco (es. DoS in seguito a ripetute richieste provenienti da uno stesso indirizzo IP).

A causa della grande mole di attacchi sconosciuti e alla crescente complessità nella configurazione di alcune tipologie di sistemi, i moderni sistemi di rilevamento delle minacce si basano sul concetto di rilevamento delle anomalie. Secondo l'approccio di rilevamento delle anomalie, si vuole strutturare un modello che sia in grado di apprendere ciò che è il normale pattern comportamentale del sistema. La finalità di tale approccio è quella poter individuare qualunque comportamento che si distacchi dal suddetto pattern, contrassegnandolo come anomalo. L'implementazione di tali modelli può avvenire secondo differenti approcci: 1. Apprendimento supervisionato, 2. Apprendimento semi-supervisionato, 3. Apprendimento non supervisionato.

Apprendimento supervisionato: Prevede l'addestramento di un modello di Machine Learning in modo supervisionato, fornendo al modello dei dati etichettati e consentendo così a questo di associare le caratteristiche del dato a una certa etichetta di classificazione, così da poter apprendere sia il pattern di dati normali che il pattern di dati anomali. In questo modo, quando al modello viene presentato un dato, questo potrà determinarne l'appartenenza alla categoria appropriata. I dati utilizzati per l'addestramento di un modello possono essere rappresentati in modi differenti in base alla loro struttura e al loro contesto di utilizzo. Tra le varie rappresentazioni utilizzate è possibile citare:

- Rappresentazione in forma tabellare, i dati sono rappresentati secondo una forma tabellare le cui colonne corrispondono alle caratteristiche comuni tra i dati e le righe corrispondono ai vari record. Ogni dato, dunque, è un vettore multidimensionale le cui dimensioni sono in numero pari alle caratteristiche del dato; per esempio, nel caso specifico del traffico di rete, questo può essere caratterizzato da informazioni sui pacchetti quali IP del mittente, IP di destinazione, numero di porta, protocollo utilizzato, ecc... Questa tipologia di rappresentazione è tra le più semplici ed è trattata in molti lavori tra cui [30–32].

- Rappresentazione tramite immagini, ogni dato è rappresentato come un tensore tridimensionale le cui prime due dimensioni corrispondono ad altezza e larghezza dell'immagine, mentre la terza dimensione rappresenta la profondità dei canali di colore; nel caso di immagini in bianco e nero la rappresentazione si limita a una matrice bidimensionale. L'approccio di machine learning più comune per l'apprendimento mediante immagini riguarda l'utilizzo delle reti neurali convoluzionali [33], le quali consentono di trovare pattern specifici nelle immagini, costituendo autonomamente un insieme di caratteristiche salienti. Secondo questo approccio si possono strutturare le immagini partendo dalla sequenza di byte dei pacchetti di rete e modificando queste con la finalità di costruire un'immagine; per esempio, un totale di 784 byte può essere utilizzato per costruire un'immagine 28x28 pixel.

- Rappresentazione testuale, ogni parola è rappresentata come un vettore numerico secondo diverse tecniche presenti in letteratura come, per esempio, quelle di Word Embedding [34]. Tra i più recenti e conosciuti modelli utilizzati nel campo dell'elaborazione del linguaggio naturale è possibile citare BERT [35] e GPT [36]. Secondo questa rappresentazione è possibile trasformare il traffico di rete in una sequenza di parole, componendo delle frasi rappresentative di una comunicazione tra computer [37].

- Rappresentazione di sequenze temporali, i dati sono organizzati seguendo un ordine cronologicamente sequenziale. Alcuni dei modelli di machine learning utilizzati per l'apprendimento dei dati basati su serie temporali sono le Recurrent Neural Networks (RNNs) [38] e le Temporal Convolutional Neural Networks (TCNs) [39]. Secondo questo approccio si vede il tempo come attributo principale che viene utilizzato per determinare gli eventi in un intervallo temporale specifico, il numero di pacchetti trasmessi, il throughput o il numero di connessioni attive.

- Rappresentazione mediante grafi, i dati sono organizzati secondo dei grafi in cui i nodi rappresentano le istanze di dati e gli archi le relazioni tra queste. Secondo questa rappresentazione il dato è da considerarsi come l'insieme di un nodo e dei suoi archi. Tra i modelli maggiormente discussi in letteratura è possibile citare le Graph Neural Networks (GNN) [40] e le Graph Attention Networks (GATs) [41]. Un esempio di come possono essere strutturati i dati secondo questa rappresentazione prevede di considerare come nodi i dispositivi tra loro connessi e come archi le sessioni, etichettate opportunamente per evidenziare il protocollo utilizzato e altre informazioni utili.

Nell'approccio illustrato in [42] gli autori propongono un modello di rilevamento delle anomalie che viene addestrato in modo supervisionato su dati di serie temporali. Un altro esempio di utilizzo della tecnica di apprendimento supervisionato è presentato in [43], dove viene illustrato un modello che, dopo aver trasformato i Bytecode del malware in un immagine, sfrutta una Rete Convoluzionale per apprenderne le caratteristiche salienti. Nonostante l'apprendimento supervisionato consenta a un modello di ottenere delle buone performance, tale approccio talvolta può risultare difficoltoso a causa della mancanza di dati classificabili come anomali. Implementare un sistema di questo tipo su una nuova rete necessita infatti di un grande quantitativo di dati etichettati relativi al traffico specifico della rete stessa, ciò rende gli approcci di addestramento supervisionato poco scalabili.

Per poter ovviare alla mancanza di dati, è possibile sfruttare gli approcci di transfer learning [44] che consistono nell'utilizzo di un modello pre-addestrato su un certo problema come base per la risoluzione di un problema correlato. Seppure questo approccio abbia rivoluzionato diversi campi del machine learning, tra cui il processamento del linguagio naturale e il riconoscimento di immagini [45], questo non può essere adattato bene alla realizzazione di un sistema di rilevamento delle anomalie. Il motivo di ciò è principalmente dovuto alle possibili discrepanze tra i domini di applicazione; infatti, ogni sistema ha una propria tipologia di dati e, di conseguenza, addestrare un classificatore basandosi su dati poco correlati con il dominio del sistema obiettivo può risultare poco efficace o addirittura degradante per il sistema stesso [46].

Apprendimento semi-supervisionato: Prevede che il modello sia addestrato mediante l'utilizzo di dati etichettati e non etichettati [47]. La motivazione principale che spinge all'utilizzo di questa tecnica è dovuta alla mancanza di dati etichettati [48]. Un pratico esempio, presentato in [49], prevede un meccanismo di trasferimento della conoscenza che consente la creazione di pseudo-etichette di classe a partire da un modello addestrato secondo un approccio supervisionato su una quantità di dati etichettati ridotta. Approccio simile viene adottato dagli autori di [50] che, partendo da un insieme di addestramento composto prevalentemente da dati misti (anomali e non) non etichettati, sfruttano pochi dati etichettati, divisi in normali e anomali, per

addestrare il modello a caratterizzare gli esempi normali e, di conseguenza, ad apprenderne il pattern.

Nonostante questo approccio riesca a risolvere, almeno in parte, il problema della scarsa quantità di dati etichettati, l'utilizzo di un modello addestrato su dati poco correlati a quelli del sistema obiettivo porterebbe al medesimo problema di negative learning [46] discusso per gli approcci di apprendimento supervisionato. Di fatto, servirebbe avere degli insiemi di addestramento ad-hoc, composti da dati etichettati e non, per ogni specifico sistema IoT, così da addestrare il modello direttamente sui dati appartenenti al dominio di interesse.

Apprendimento non supervisionato: L'assunzione di base dietro a questa metodologia di apprendimento è quella secondo cui tutti i dati di addestramento appartengano alla stessa classe e che quindi il modello possa essere in grado di apprendere uno specifico pattern, considerato normale, classificando come anomalo qualunque dato si discosti da questo. Questo approccio, oltre a risolvere il problema legato alla mancanza di dati etichettati, consente di adattare il modello al sistema su cui questo viene implementato in quanto l'addestramento avviene proprio sui dati generati dal sistema stesso.

2.2 Apprendimento non supervisionato

Tra i vari approcci di addestramento, le tecniche di Apprendimento non Supervisionato rientrano tra le più appropriate per ciò che riguarda i sistemi IoT in quanto il traffico generato all'interno di tali sistemi risulta essere molto variegato e dipendente dalle caratteristiche della rete specifica. Poiché ogni rete è differente, l'impiego di forza lavoro esperta nell'analisi e nell'etichettatura del traffico di rete risulterebbe eccessivamente dispendioso, sia dal punto di vista economico che logistico. Inoltre, l'utilizzo di modelli addestrati su dataset pubblici, che quindi non contengono dati specifici per la rete su cui viene implementato il sistema, porterebbe a scarse performance [51, 52].

Sfruttando le tecniche di apprendimento non supervisionato, è possibile realizzare un modello che possa essere installato su sistemi già in funzione e che si addestri sui dati prodotti dagli stessi senza la necessità di essere supervisionato da parte di un operatore umano. L'idea è quella di partire dalla struttura del modello, implementarla all'interno del sistema oggetto di interesse e avviare la raccolta dei dati da esso generato. Una volta terminata la fase di raccolta, l'insieme di record costituito può essere utilizzato per addestrare il modello, rendendo questo capace di apprendere il normale pattern dei dati generato dal sistema. Nel caso in questione, l'obiettivo è quello di determinare se uno specifico dato possiede o meno le caratteristiche di un attacco informatico. Si vuole quindi addestrare un modello che sia in grado di apprendere il pattern di dati considerati benevoli in modo tale da poter contrassegnare come anomalo qualunque dato si discosti dal pattern appreso.

In letteratura sono presenti diverse possibilità per ciò che riguarda il rilevamento di anomalie basato su tecniche di addestramento non supervisionate. Tra i modelli di machine learning più comunemente utilizzati [53] al fine di rilevare la presenza di anomalie nelle reti IoT, basandosi su un approccio di apprendimento non supervisionato, è possibile citare: 1. Isolation Forest, 2. One-Class Support Vector Machine, 3. Autoencoder.

Isolation Forest: L'algoritmo di Isolation Forest [54] si basa sull'idea che le anomalie sono osservazioni rare e differenti, quindi più facili da isolare rispetto ai punti normali. Secondo tale tecnica, si utilizzano degli alberi decisionali mediante cui si procede a separare i dati ponendo delle soglie su caratteristiche scelte casualmente, dividendo i dati in due gruppi differenti. Il principio alla base di questo algoritmo è quello per cui i dati anomali riescono a essere isolati

prima dal momento in cui è più facile trovare delle caratteristiche per cui il gruppo di dati anomali venga separato dal gruppo di dati normali [55]. Gli insiemi di dati che più spesso sono isolati rapidamente possono essere considerati insiemi anomali. Problemi legati a questo approccio, come descritto in [55], riguardano gli alti tassi di falsi positivi e falsi negativi per dataset con numerosi record ad alta dimensionalità.

One-Class Support Vector Machine: Secondo il modello One-Class SVM tutti i dati vengono trattati come normali, cercando di costruire un iperpiano che sia in grado di separare questi dal resto dello spazio [56]. Qualunque punto individuato che non appartiene alla parte di iperpiano su cui sono presenti i dati di addestramento viene identificato come dato anomalo. Problema principale riguarda l'alta sensibilità del modello alla presenza di valori anomali nel dataset di addestramento [57].

Autoencoder: L'Autoencoder è uno dei modelli di machine learning maggiormente utilizzato nell'ambito del rilevamento di anomalie che, come specificato dagli autori di [58], rientra tra i modelli che ha suscitato più interesse nell'ultimo periodo.

L'Autoencoder è una rete neurale la cui struttura, mostrata in figura 2.1, prevede due moduli fondamentali:

- Encoder, trasforma il dato di input in una codifica, tipicamente a minore dimensionalità, che prende il nome di "rappresentazione latente".
- Decoder, tenta di ricostruire il dato di input originale a partire dalla sua rappresentazione latente.

L'addestramento di un autoencoder ha come obiettivo la minimizzazione dell'errorre di ricostruzione del dato di input. Dato il modello, definito da un insieme di parametri $W = w_1, w_2, ..., w_n$, i dati di input X, il modulo Encoder E e il modulo Decoder D, l'obiettivo dell'addestramento di un Autoencoder può essere definito dal seguente problema:

$$\underset{W}{\operatorname{argmin}} \|X - D_W(E_W(X))\|_2 \tag{2.1}$$

Ciò corrisponde a voler trovare quella configurazione di parametri per cui viene minimizzato l'errore di ricostruzione definito come la norma L_2 della differenza tra il dato di input e la sua ricostruzione. I dati significativamente diversi da quelli su cui è stato addestrato il modello avranno una ricostruzione peggiore; assumendo che le anomalie non siano simili ai dati normali, queste avranno probabilmente un errore di ricostruzione maggiore rispetto a quello calcolato per dati non anomali. Tenendo conto di ciò, è possibile determinare un valore di soglia sull'errore di ricostruzione oltre il quale un dato è considerato anomalo, costruendo così un classificatore. Per esempio, gli autori di [58] utilizzano un modello di Deep Autoencoder centralizzato, addestrato sui soli dati normali provenienti dai vari dispositivi presenti all'interno del sistema IoT, al fine di rilevare le anomalie legate alle attività di Botnet.

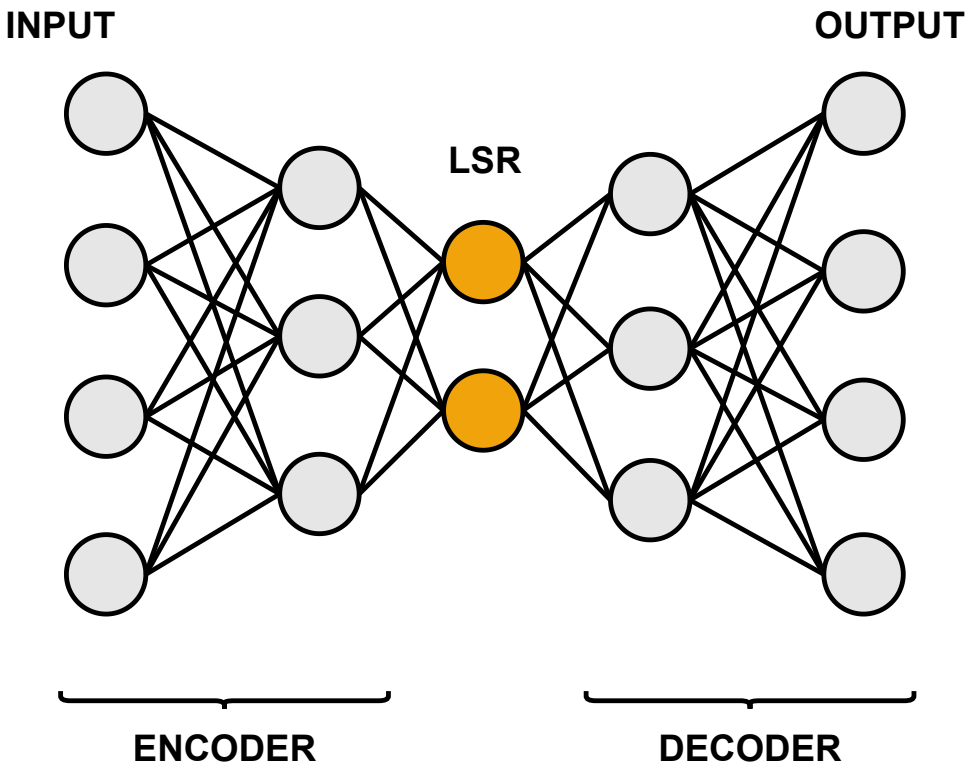


Figura 2.1: Struttura di un Autoencoder: La rete riceve in input un vettore di caratteristiche che vengono trasformate dall'Encoder in una rappresentazione latente del dato. Tale rappresentazione latente sarà poi utilizzata dal Decoder per ricostruire il vettore di caratteristiche di input.

2.3 Addestramento collaborativo distribuito

L'addestramento di un modello di machine learning centralizzato, situato in un sistema che vede la presenza di molteplici dispositivi, necessita lo scambio e la memorizzazione di grandi quantitativi di dati. L'acquisizione dei dati associati a ogni dispositivo comporta inoltre un rischio per la privacy [59], motivo per cui risulta utile strutturare approcci di addestramento collaborativo. Gli approcci di addestramento collaborativo [18] hanno la finalità di addestrare un modello globale distribuito tra tutti i dispositivi partecipanti all'addestramento, evitando così che siano i dati a dover essere trasmessi verso un modello centralizzato. Questi approcci, inoltre, consentono una divisione dell'intero modello tra più dispositivi, consentendo quindi un bilanciamento dell'onere computazionale. Come specificato dagli autori di [30], l'utilizzo di tecniche di addestramento collaborativo contribuisce a mantenere un certo livello di riservatezza dei dati, evitando che questi vengano inviati a entità esterne al dispositivo su cui hanno origine.

Nel caso particolare dei sistemi IoT, composti da dispositivi a risorse limitate, distribuire il modello di machine learning può risultare utile. La suddivisione del modello su più dispositivi consente di alleggerire il carico di lavoro del singolo dispositivo evitando comunque di doverne trasmettere i dati di addestramento [19].

2.3.1 Federated Learning

Il Federated Learning [20, 21] consente a più dispositivi di collaborare, eseguendo l'addestramento di questi in parallelo e sui propri dati locali, con la finalità di addestrare un modello di Machine Learning comune. Secondo questo approccio, un server centrale inizializza un modello

globale e lo distribuisce tra i vari dispositivi partecipanti all'addestramento. Ogni dispositivo addestra, sui propri dati, la propria copia locale del modello e la invia al server che si occupa di aggregare i pesi di tutti i modelli ricevuti dai vari dispositivi, aggiornando così i pesi del modello globale con il risultato di tale aggregazione. Una volta aggiornato il modello globale, questo viene nuovamente distribuito tra i dispositivi, passando a un nuovo round di apprendimento federato.

La tecnica tradizionale utilizzata per calcolare i pesi del modello globale, come riportato dagli autori di [51], prende il nome di **FedAVG** e prevede di calcolare il modello globale aggiornato come:

$$W = \sum_{k \in Clients} \frac{n_k}{n} * W^k \tag{2.2}$$

dove W^k indica il *k-esimo* modello e $\frac{n_k}{n}$ indica il rapporto tra il numero di dati di addestramento utilizzati dal *k-esimo* modello locale e la numerosità totale dei dati di addestramento. In figura 2.2 sono riportati i passaggi necessari per addestrare un modello secondo la tecnica di Federated Learning.

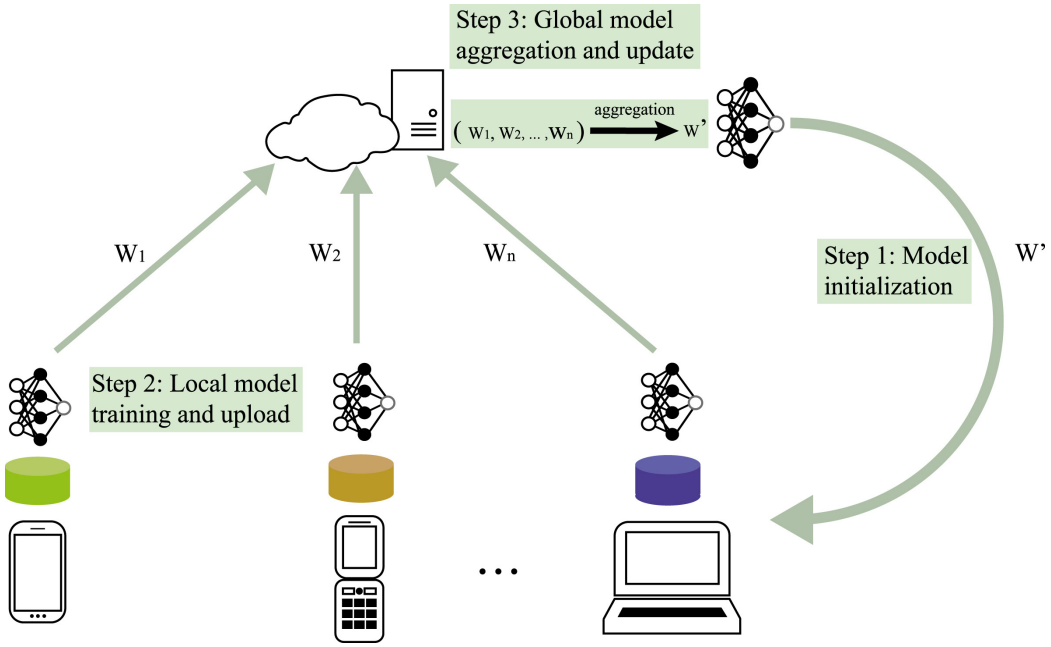


Figura 2.2: Struttura Federated Learning: Ogni client, partendo da un modello globale comune, addestra il proprio modello locale inviando infine i rispettivi pesi al server che si occuperà di effettuarne un aggregazione volta ad aggiornare il modello globale che verrà nuovamente ridistribuito tra tutti i client.
Immagine tratta da: [20]

Il numero di round di apprendimento federato è un iperparametro dell'algoritmo di addestramento e consente di gestire il trade-off tra l'onere computazionale e la rappresentatività del modello globale rispetto ai modelli locali. Nello specifico, ogni $\boldsymbol{e}$ epoche di addestramento dei singoli modelli locali avviene uno degli $\boldsymbol{r}$ diversi round di aggregazione in cui il modello globale risultante viene ridistribuito tra tutti i client, i quali procedono così ad avviare un nuovo round di addestramento. La procedura di FedAVG è dettagliata dall'algoritmo 1.

Algoritmo 1 Algoritmo FedAVG

Input: Numero di round r, numero di epoche locali e, valore di learning rate η, funzione di loss $\mathcal{L}$, lista dei client C, dataset locali $D = \{D_1, ..., D_C\}$.

Server:

Inizializzazione del modello globale W_0
for each client k in C **do**
    $W^k \leftarrow W_0$
for t in range(r) **do**
    for each client k in C **do**
        $W^k_{t+1} \leftarrow ClientUpdate(k, W_t)$
    $W_{t+1} \leftarrow \sum_{k=1}^{C} \frac{n_k}{n} * W^k_{t+1}$

function *ClientUpdate*(k, W): # Funzione eseguita per il k-esimo client
    for each epoch in e **do**
        $W^k \leftarrow W^k - \eta * \nabla\mathcal{L}(W^k)$
    return W^k

Poiché i dati di addestramento dei singoli dispositivi non vengono mai trasferiti, come dettagliato in [21], l'approccio di apprendimento federato offre la possibilità di realizzare modelli di machine learning robusti sotto il punto di vista della privacy. L'utilizzo dei soli pesi dei modelli locali per addestrare un modello globale consente di seguire un approccio model-to-data [60] secondo cui nessun dispositivo necessita la condivisione dei propri dati.

2.3.2 Split Learning

Lo Split Learning è un paradigma di apprendimento distribuito secondo cui un modello di Machine Learning viene suddiviso in frammenti situati su due o più dispositivi. La principale finalità è quella di alleggerire l'onere computazionale a cui ogni dispositivo sarebbe sottoposto durante le fasi di addestramento e inferenza.
Il paradigma di Split Learning prevede di eseguire i passi in avanti di ogni frammento di modello fino all'ultimo layer dello stesso, generandone gli smashed data (attivazioni dell'ultimo layer) $f(D; W^c_t)$, dove D rappresenta l'insieme di addestramento e W^c_t la parte di modello corrente. Gli smashed data prodotti sono inviati al frammento di modello topologicamente successivo e, una volta raggiunto l'output layer dell'ultimo frammento di modello, viene avviato il processo di backpropagation. Secondo questo processo, ogni dispositivo invia i gradienti calcolati fino a quel momento verso il dispositivo che implementa il frammento di modello che lo precede, dando a questo la possibilità di aggiornare i pesi dei propri neuroni secondo la formula
$W^c_{t+1} \leftarrow W^c_t - \eta * \nabla\mathcal{L}(A_t; W^s_t)$, dove η indica il tasso di apprendimento, A_t sono le attivazioni del frammento di modello precedente e $\mathcal{L}(A_t; W^s_t)$ rappresenta il calcolo della funzione di costo.

Lo pseudocodice in 2 fornisce una visione più dettagliata del suddetto algoritmo applicato a una configurazione in cui il modello è suddiviso tra un client e un server. In questo caso, il client mantiene i dati in locale, addestrando il suo frammento di modello e inviando gli smashed data verso il server che completerà l'addestramento della sua parte di modello e avvierà il processo di backpropagation. Questa configurazione consente di alleggerire il carico di lavoro del client, sfruttando un server con maggiori capacità computazionali e implementando in esso la parte

più onerosa del modello di machine learning, il tutto mantenendo i dati del client in locale, proteggendone la privacy.

Algoritmo 2 Algoritmo Split Learning

Input: Numero di epoche e, valore di learning rate η, funzione di loss $\mathcal{L}$, dataset D.

Server:

Inizializzazione del modello lato server W_0^s e lato client W_0^c
for t in range(e) **do**
    $A_t = GetClientSmashed(t)$
    $W_{t+1}^s \leftarrow W_t^s - \eta * \nabla\mathcal{L}(W_t^s; A_t)$
    $ClientBackprop(\nabla\mathcal{L}(A_t; W_t^s), t)$

Funzioni lato client
function $GetClientSmashed(t)$:
    $A_t = f(D; W_t^c)$
    return A_t

function $ClientBackprop(\nabla\mathcal{L}(A_t; W_t^s), t)$:
    $W_{t+1}^c \leftarrow W_t^c - \eta * \nabla\mathcal{L}(A_t; W_t^s)$

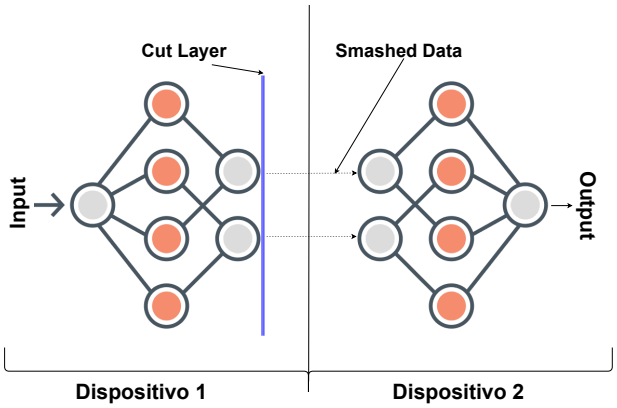
La scelta riguardante la disposizione dei frammenti di rete sui differenti dispositivi è del tutto arbitraria e non pone alcuna limitazione riguardo quali e quante parti di modello possano essere situate su uno specifico dispositivo. La figura 2.3, mostra due possibili configurazioni di Split Learning in cui il modello viene suddiviso tra due dispositivi. La figura 2.3a illustra una configurazione in cui la parte iniziale del modello è situata sul primo dispositivo, mentre la parte terminale è situata sul secondo. In questa configurazione l'output del modello è generato su un dispositivo esterno rispetto a quello da cui deriva l'input, causando una perdita di confidenzialità qualora l'output, o il numero di classi del modello, fosse un'informazione sensibile. La figura 2.3b, invece, mostra una configurazione secondo cui la parte iniziale e la parte terminale del modello sono situate nel primo dispositivo, mentre la parte centrale del modello è situata su un secondo dispositivo. Questa configurazione risolve il problema di confidenzialità emerso per la topologia in figura 2.3a, mantenendo l'output del modello all'interno del dispositivo da cui proviene l'input.

Una situazione di questo tipo, diversamente da ciò che avviene per la configurazione precedente, riesce a risolvere il problema di confidenzialità dei dati di un autoencoder poiché output e input, trovandosi sullo stesso dispositivo, permettono di calcolare in locale l'errore di ricostruzione.

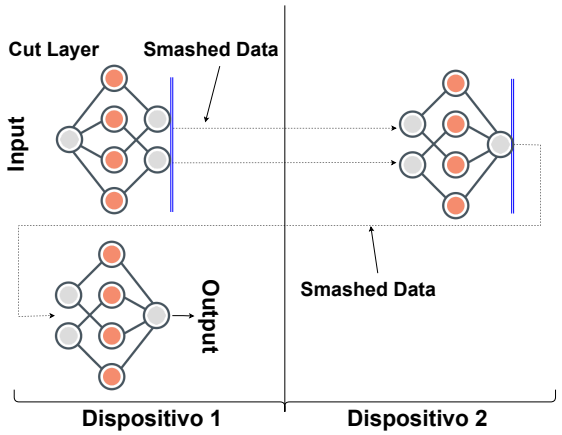
Come illustrato dagli autori di [19], oltre che un'efficienza dal punto di vista dell'onere computazionale dei singoli dispositivi coinvolti nell'addestramento del modello, questa tecnica consente di raggiungere un certo grado di riservatezza per ciò che riguarda il trasferimento dei dati di addestramento verso un dispositivo esterno; di fatto, non vengono mai trasferiti i dati di input, bensì gli smashed data delle varie porzioni di modello. Inoltre, come mostrato in figura 2.3b, è possibile evitare la condivisione delle etichette partizionando le parti iniziale e terminale del modello nello stesso dispositivo.

2.3.3 SplitFed Learning

Nonostante gli approcci di Federated Learning e Split Learning offrano diversi vantaggi, sia dal punto di vista dell'efficienza che dal punto di vista della privacy, come illustrato in [60], questi



(a) Il modello è distribuito in due frammenti ognuno dei quali è situato su un differente dispositivo.



(b) Il modello è distribuito in modo non equo tra i dispositivi; in particolare, il primo dispositivo mantiene sia la parte iniziale che la parte terminale del modello.

Figura 2.3: Struttura Split Learning

presentano alcune limitazioni. Per ciò che riguarda il Federated Learning, lo svantaggio principale risiede nella necessità per tutti i dispositivi partecipanti all'addestramento di addestrare un intero modello di machine learning, utilizzando così gran parte delle risorse computazionali [61].

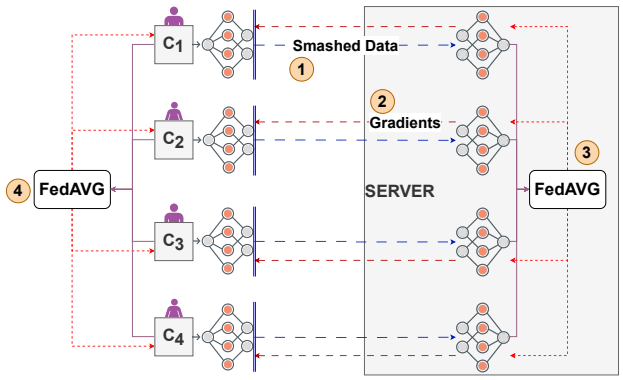
Lo Split Learning presenta una buona soluzione per un equo bilanciamento delle risorse tra i dispositivi; tuttavia, dal momento in cui il server entra in comunicazione con un solo client alla volta, le risorse dei client rimanenti rimarranno inutilizzate. Un approccio di questo tipo genera alti tempi di inattività tra i dispositivi che non stanno comunicando con il server, portando a un addestramento lato client sequenziale [60] e, di conseguenza, lento. Inoltre, l'approccio di Split Learning non consente una collaborazione tra i client coinvolti nell'addestramento; pertanto, ogni client necessita di un quantitativo di dati sufficiente ad addestrare il proprio modello.

Per sopperire ai problemi dei sopracitati approcci, cogliendone però i vantaggi, gli autori di [60] presentano l'architettura SplitFed Learning in cui viene mantenuto l'approccio *model-to-data* al fine di mantenere la privacy all'interno del sistema. Questa architettura consente di strutturare un approccio di apprendimento federato senza la necessità, per ogni dispositivo, di mantenere una copia dell'intero modello di machine learning. Secondo l'architettura in questione, il modello è suddiviso tra client e server, riducendo l'onere computazionale e spaziale necessario per ogni singolo dispositivo. Inoltre, poiché la parte di addestramento lato client può avvenire in parallelo, questo approccio risulta essere più efficiente sotto il punto di vista temporale. In figura 2.4, oltre al flusso di lavoro, sono illustrate le componenti della struttura in questione. Le componenti principali sono i client, con i rispettivi modelli, il server e i modelli associati a ogni client che il server mantiene in memoria. I blocchi denominati *FedAVG*, invece, corrispondono all'omonima formula (eq. 2.2) e sono utilizzati per implementare la fase di addestramento federato.

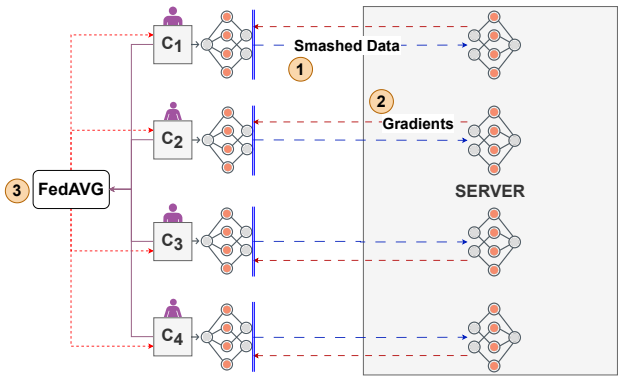
I passaggi numerati in figura 2.4a possono essere descritti come segue:

1. I modelli client-side, in parallelo, eseguono i primi passi in avanti della parte di rete in essi implementata, provvedendo a inviare gli smashed data verso il server, seguendo un classico approccio di split learning.

2. Il server, ricevuti gli smashed data, addestrerà per ogni client una differente seconda porzione del modello, avvierà la backpropagation e invierà i gradienti verso il cut layer dei corrispettivi client.



(a) SFLv1: Ogni client invia i propri smashed data verso il server che si occupa di avviare, per ogni modello client-side, il processo di backpropagation. Al termine di tale processo sia i modelli client che i modelli server vengono aggregati.



(b) SFLv2: Variante in cui il server, dopo aver inviato i gradienti verso i modelli client-side, non esegue l'operazione di aggregazione e mantiene un modello server-side univoco per ogni client con lo scopo di strutturare un modello individuale per ogni dispositivo presente nel sistema.

Figura 2.4: Struttura SplitFed Learning

3. Una volta terminato l'aggiornamento dei pesi, il server provvederà ad applicare FedAVG 2.2 per costituire un modello server-side globale e sostituirlo ai modelli server-side dei rispettivi client.

4. Quando i client ricevono i corrispettivi gradienti, terminano l'operazione di backpropagation e aggiornano i propri modelli locali, utilizzandone i pesi per contribuire alla costruzione di un modello globale, secondo tecnica di federated learning, che verrà poi ridistribuito tra i client.

Gli autori dell'articolo di riferimento propongono anche una seconda versione della sopracitata struttura. Tale versione, denominata *SFLv2* (SplitFed Learning v2), ha come obiettivo il raggiungimento di un maggiore livello di accuratezza che può essere ottenuto eliminando completamente la fase di aggregazione dei modelli server-side. Così facendo, il server addestra sempre un modello server-side individuale per ogni client, garantendone una rappresentatività specifica. La figura 2.4b mostra il flusso di lavoro dell'architettura modificata dove è possibile notare come la parte client rimanga invariata rispetto alla versione precedente.

L'algoritmo 3 dettaglia la procedura di addestramento utilizzata in SplitFed suddividendo il Main Server dal server che si occupa dell'aggregazione dei modelli lato client (Fed Server). I metodi *GetClientSmashed* e *ClientBackprop* sono equivalenti a quelli definiti in 2.

Algoritmo 3 Algoritmo SplitFed Learning

Input: Insieme di client C, numero di epoche e, numero di round r, valore di learning rate η, funzione di loss $\mathcal{L}$.

Main Server:

for each round t in range(r) **do**
  if $t = 0$ **then**
    Inizializzazione del modello globale lato server W_0^s
    for each client k in C **do**
      $W_0^{s,k} = W_0^s$
  else
    for each client k in C **do**
      while *epoch* $\neq e$ **do**
        $A_t^k = GetClientSmashed(t)$
        $W_{t+1}^{s,k} \leftarrow W_t^{s,k} - \eta * \nabla\mathcal{L}(W_t^{s,k}; A_t^k)$
        $ClientBackprop(\nabla\mathcal{L}(A_t^k; Ws, k_t))$
        $t+ = 1$
  $W_{t+1}^s \leftarrow \sum_{k=1}^{C} \frac{n_k}{n} * W_{t+1}^{s,k}$

Fed Server:

for each round t in range(r) **do**
  if $t = 0$ **then**
    Inizializzazione del modello globale lato client W_0^c
    for each client k in C **do**
      $W_0^{c,k} = W_0^c$
  else
    $W_{t+1}^c \leftarrow \sum_{k=1}^{C} \frac{n_k}{n} * W_t^{c,k}$
    for each client k in C **do**
      $W_{t+1}^{c,k} \leftarrow W_t^c$

Considerando l'alta numerosità ed eterogeneità dei dispositivi presenti in un sistema IoT, gli autori di [51] propongono un'architettura gerarchica che combina gli approcci proposti in [60] e [62] al fine di raggiungere degli obiettivi di efficienza per ciò che riguarda i tempi di addestramento. L'architettura in questione consente un bilanciamento dell'onere computazionale, incrementando però il numero di comunicazioni tra le diverse entità che la compongono. Considerando l'implementazione di un'architettura di questo tipo all'interno di un sistema IoT, è possibile considerare le modalità secondo cui il modello andrebbe suddiviso tra i vari dispositivi. Date le scarse capacità computazionali dei dispositivi, a ognuno di questi può essere associata una parte di modello poco profonda che non comporti un onere computazionale eccessivo. La restante parte di modello, quella più corposa, può essere implementata nei nodi edge che sono dotati di maggiori capacità computazionali e, dunque, risultano in grado di sostenere degli oneri maggiori. In particolare, una prima parte di modello poco profonda è situata nei dispositivi IoT (Client) i quali, suddivisi in gruppi, comunicano con un nodo edge su cui è situata un'ulteriore parte di modello. Infine, tutti i nodi edge rappresentativi dei diversi gruppi di client sono coinvolti nell'addestramento dell'ultima parte di modello.

Come mostrato in figura 2.5, l'architettura vede la presenza di tre categorie di elementi principali: *client*, *edge* e *cloud*. Nel workflow in figura, i nodi *edge* faranno da aggregatori per gruppi distinti di client, ognuno dei quali addestra una prima parte di modello locale, addestrandone

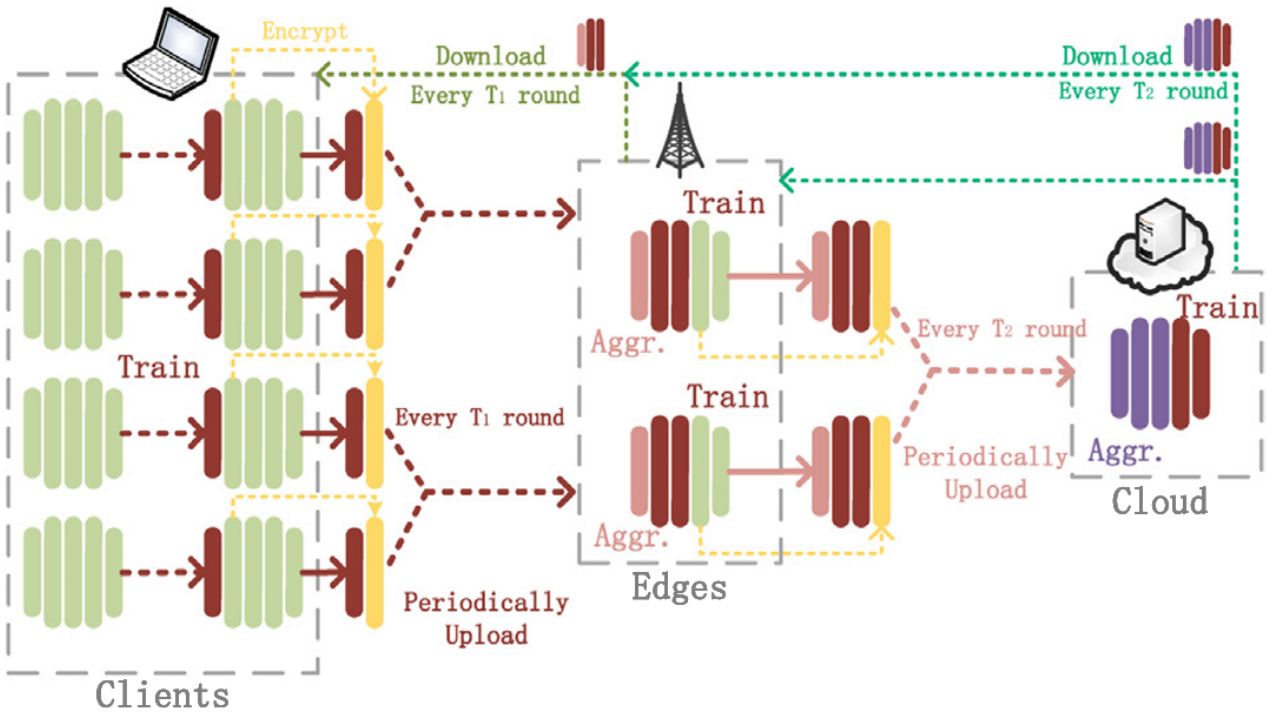


Figura 2.5: Struttura Hier-SFL: I client avviano la procedura di addestramento su una prima parte di modello sfruttando gli aggregatori come supporto per addestrare, mediante SFLv1, la parte centrale del modello. Infine, gli stessi aggregatori collaboreranno per addestrare la parte terminale situata sul server cloud. Immagine tratta da: [51]

la parte centrale secondo l'approccio *SFLv1*. In seguito, gli stessi edge collaboreranno tra loro, utilizzando la medesima tecnica, per addestrare la parte terminale di modello situata nel cloud. In figura è mostrata anche una fase di cifratura precedente alla trasmissione dei dati tra i diversi elementi; tuttavia, tale fase è del tutto opzionale e la sua applicazione non incide in alcun modo con la topologia della struttura proposta.

Utilizzando un approccio di Split Learning, gli autori di [63] hanno definito un'architettura gerarchica che vede la suddivisione di un autoencoder su tre livelli differenti:

- Client
- Aggregator
- Server

Tale variante nasce con lo scopo di rilevare i furti di energia elettrica senza la necessità di violare la privacy dei client coinvolti. In questa architettura, mostrata in figura 2.6, ogni livello mantiene una parte di modello che viene addestrato secondo un approccio di Split Learning. Differentemente dall'architettura gerarchica presentata in [51], l'aggregazione dei modelli non avviene seguendo l'approccio di apprendimento federato, bensì aggregando gli smashed data prodotti dai dispositivi che seguono il paradigma di Split Learning. Analogamente a quanto detto per la precedente architettura, l'integrazione di una struttura di questo tipo all'interno di un sistema IoT consentirebbe un buon bilanciamento degli oneri computazionali tra i diversi client e i vari nodi edge.

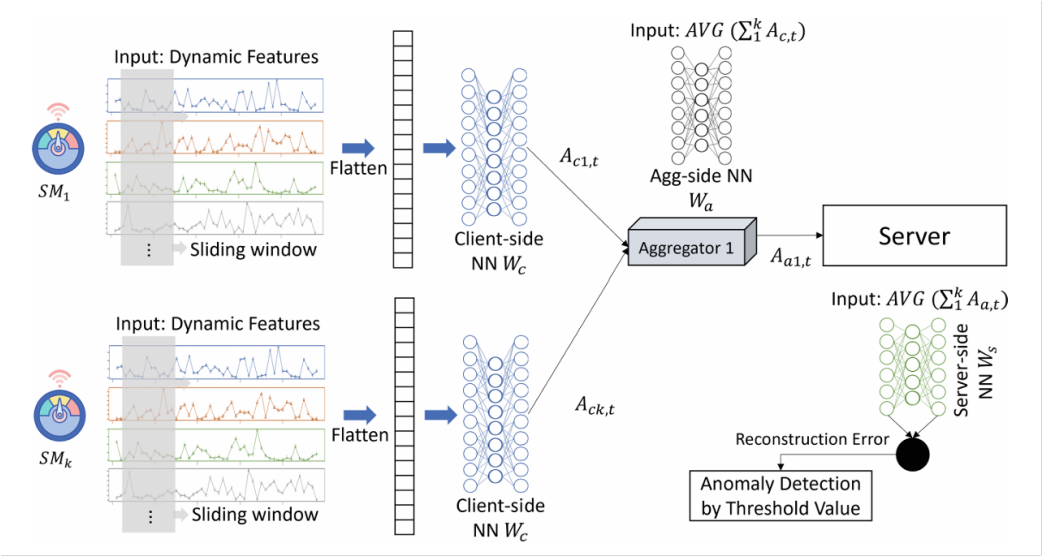


Figura 2.6: Struttura Three-Tier Split Learning: Ogni client avvia l'addestramento del proprio modello locale le cui attivazioni $A_{ck,t}$ al tempo t sono inviate verso un nodo comune denominato aggregatore. L'aggregatore si occupa di aggregare i valori delle attivazioni ottenute, eseguendone la media aritmetica, e prosegue l'addestramento della parte centrale del modello. Infine, ogni aggregatore invia le attivazioni A_{ak} della propria parte di modello verso un server globale che procede ad aggregarne i valori e ad addestrare la parte terminale dell'intera rete. Immagine tratta da: [63]

Secondo l'architettura in figura 2.6, l'errore di ricostruzione è calcolato dal server globale sfruttando solamente la parte di modello in esso implementata ed evitando dunque la necessità di accedere al dato di input originale. Seppure in questo modo viene evitato lo scambio dei dati grezzi, è opportuno evidenziare la necessità di trasferire frequentemente il valore delle attivazioni dei client verso gli aggregatori e, in seguito, le attivazioni di questi ultimi verso il server centrale.

Come specificato in [64], esistono diversi attacchi contro le architetture che seguono il paradigma di Split Learning; questi attacchi, tra le altre cose, possono avere come obiettivo la ricostruzione dei dati di input della prima frazione di modello. Tale ricostruzione può essere ottenuta a partire dagli smashed data mediante attacchi basati su tecniche di inferenza [65–68]. I più recenti attacchi sono presentati in [64, 69, 70] e possono essere mitigati rendendo il modello più profondo e complesso. Nonostante una maggiore stratificazione riesca a mitigare il rischio associato ai suddetti attacchi, è bene ribadire la necessità di diminuire la complessità dei modelli sui client all'interno dei sistemi IoT. Per i sopracitati motivi è necessario considerare come la superficie d'attacco dell'intero sistema cresca con il numero di dispositivi su cui il modello è suddiviso; motivo per cui architetture come quelle presentate in [51, 63] possono rivelarsi altamente vulnerabili.

2.4 Mitigazione del Data poisoning

Un attacco di avvelenamento dei dati (Data Poisoning) avviene quando un'entità malintenzionata immette dei dati di natura malevola all'interno del dataset di addestramento, facendo in modo che il modello apprenda un pattern di dati differente da quello desiderato. Nel caso specifico di un sistema IoT che addestra un modello di rilevamento delle anomalie secondo un approccio di apprendimento non supervisionato, la presenza di un client compromesso che genera traffico di rete malevolo porta il modello ad adattarsi a quel pattern di dati, perdendo così la capacità di discriminare il traffico di rete anomalo da quello normale. Questa tipologia di attacchi risulta essere più semplice da attuare in scenari di addestramento collaborativo distribuito in quanto, per raggiungere gli obiettivi sulla privacy, non si ha un controllo diretto sui dati e sulla loro integrità [71]; la manomissione di un singolo client del sistema federato comporterebbe infatti un peggioramento nelle performance del modello globale risultante [72]. Nello scenario di addestramento non supervisionato, inoltre, il problema è accentuato in quanto la rete non può essere addestrata a classificare i dati come malevoli o meno poiché, per definizione, non vi è la conoscenza delle etichette di questi. In un approccio di rilevamento delle anomalie, è quindi fondamentale che il dataset su cui il modello viene addestrato comprenda solamente il normale pattern di dati atteso [73]. Proprio per questo motivo, in letteratura vengono presentate diverse soluzioni e confronti tra differenti modelli al fine di comprenderne le caratteristiche di robustezza e le capacità di mitigazione del problema. In [74] viene svolta un'analisi riguardo le performance del modello di Deep Autoencoder ottenuto in seguito a un addestramento in presenza di dati infetti, tale analisi descrive un buon livello di robustezza del modello in questione quando la percentuale di dati infetti non risulta essere troppo elevata. Gli autori, mantenendo fissato la frazione di falsi positivi, analizzano i risultati di una serie di addestramenti al variare della percentuale di dati infetti e ottengono percentuali di veri positivi superiori al 50% quando appena 1/50 dei dati risulta essere infetti.

Tuttavia, all'aumentare della numerosità di dati infetti le performance del modello tendono comunque a degradare, arrivando a minare totalmente le capacità di classificazione di quest'ultimo.

2.4.1 FedLS

L'architettura descritta in [75] costituisce un meccanismo di esclusione degli aggiornamenti anomali che vede come dominio di applicazione i *Federated Network Intrusion Detection Systems*, ovvero dei sistemi di rilevamento delle intrusioni che vengono addestrati secondo un approccio di apprendimento federato. Gli autori illustrano in che modo il federated learning sia prono ad attacchi di data poisoning; in particolare, specificano che gli avversari agiscano addestrando i modelli di specifici client (client infetti) su dati malevoli. Così facendo, i client infetti contribuiscono all'addestramento del modello globale modificandone i pesi in modo anomalo rispetto a quanto fatto dai client benevoli (non infetti).

Come mostrato in figura 2.7, la struttura in questione agisce al momento in cui i modelli locali dei client vengono aggregati, non discostandosi dalla tipica topologia di un'approccio di apprendimento federato; in particolare, il modulo FedLS, dettagliato in figura 2.8, si occupa di rimuovere tutti gli aggiornamenti malevoli, generando così una lista di soli client benevoli i cui modelli possono essere aggregati secondo la formula 2.2.

L'architettura FedLS si basa sull'utilizzo delle Penultimate Layer Representations (PLRs), definite come le attivazioni del penultimo strato del modello, e di un autoencoder. Nel dettaglio, prima che una copia del modello globale venga distribuita tra i client partecipanti all'addestramento federato, il server di aggregazione avvia una fase di addestramento del proprio autoencoder, sfruttando un proprio dataset e collaborando con altre organizzazioni topologicamente uguali, con l'assunzione che queste non siano infette, al fine di apprendere le caratteristiche delle

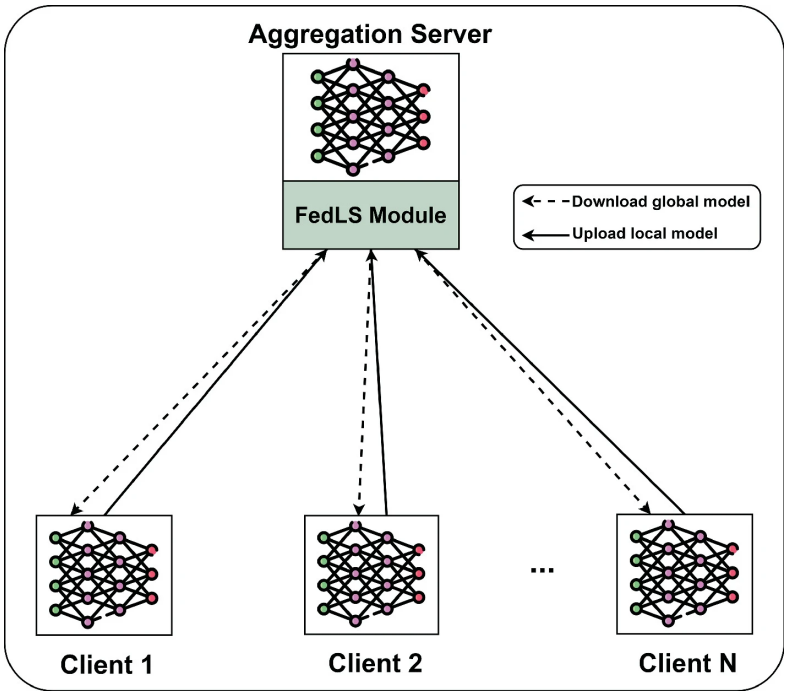


Figura 2.7: Meccanismo di addestramento federato proposto in [75] che vede l'aggregazione dei modelli locali in seguito a un'interazione di questi con il modulo FedLS.
Immagine tratta da: [75]

PLRs benevole. Una volta addestrato l'autoencoder a ricostruire le PLRs benevole, le PLRs del modello globale e dei modelli locali vengono estratte e passate allo strato di encoding dell'autoencoder, ottenendo così le corrispettive rappresentazoni latenti (LSR); a questo punto, vengono calcolate le misure di similarità tra le LSR provenienti dal modello globale e le LSR di ogni modello locale sfruttando la formula 2.3 basata sul criterio di indipendednza di Hilbert-Schmidt [76].

$$CKA = \frac{HSIC(X,Y)}{\sqrt{HSIC(X,X)HSIC(Y,Y)}} \tag{2.3}$$

Fatto ciò, viene eseguito l'algoritmo di clustering *KMeans* per creare due cluster e selezionare quello avente il centroide con valore più alto come cluster dei client benevoli; infine, la lista di tali client viene utilizzata per selezionare i client partecipanti all'aggregazione dei modelli.

Seppure tale architettura riesca a mitigare il fenomeno dell'avvelenamento dei dati, questa necessita di pre-addestrare un autoencoder, collaborando con organizzazioni fidate, al fine di apprendere le rappresentazioni latenti del pattern tipico di dati considerati benevoli. Tale assunzione, tuttavia, non è pratica per ciò che riguarda il rilevamento di anomalie nei sistemi IoT dal momento in cui ogni organizzazione presenta un diverso pattern di traffico di rete e, di conseguenza, l'addestramento dell'autoencoder sul traffico prodotto da organizzazioni esterne non produrrebbe un modello rappresentativo del sistema IoT in questione. Inoltre, l'assunzione riguardo la collaborazione con organizzazioni prive di client infetti risulta troppo forte e non consente di mantenere il sistema di rilevamento delle anomalie indipendente dalla rete esterna. I sopracitati motivi portano dunque alla necessità di adottare metodologie in grado di mitigare l'impatto dovuto alla presenza di client infetti senza la necessità di introdurre terze parti fidate.

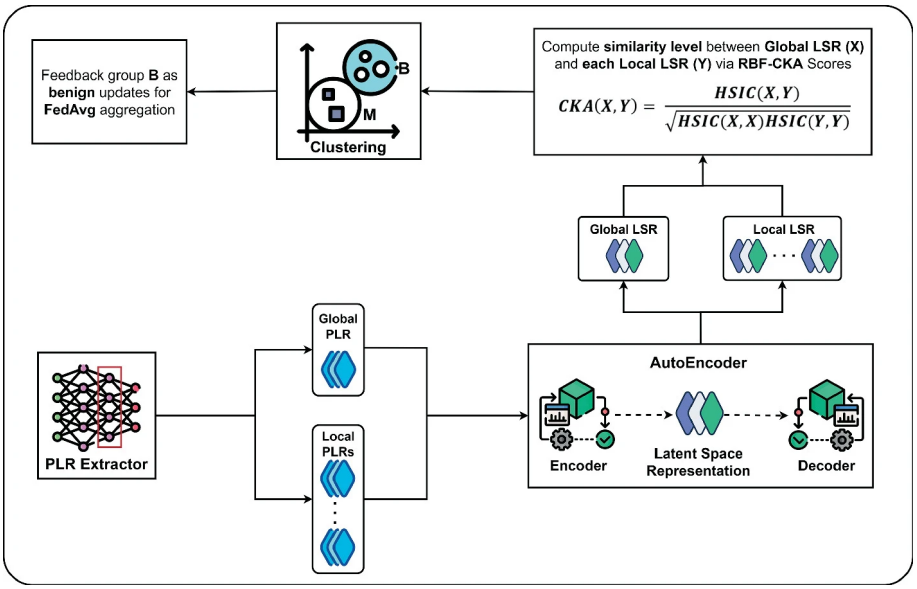


Figura 2.8: Modulo FedLS: Le PLRs dei modelli locali vengono estratte e passate all'Autoencoder con la finalità di ottenerne le rappresentazioni latenti. Ognuna di queste, in seguito, viene utilizzata per calcolarne la similarità con le LSRs provenienti da PLRs globali. Partendo dai valori di similarità ottenuti è possibile formare due cluster e considerare quello con centroide a valore maggiore, corrispondente quindi al cluster i cui punti hanno maggiori livelli di similarità con le PLRs globali, come il cluster di client benevoli.
Immagine tratta da: [75]

2.4.2 Layer-Based Anomaly-Aware FedAVG

L'architettura Layer-Based Anomaly-Aware FedAVG (LBAA-FedAVG), presentata in [77], offre una metodologia di aggregazione secondo cui vengono esclusi dalla fase di aggregazione tutti quei livelli, provenienti dai modelli locali, considerabili come compromessi. Come specificato dagli autori, LBAA-FedAVG è un algoritmo di aggregazione a conoscenza zero basato sul clustering e, dunque, si presta bene a essere utilizzato nel dominio dei sistemi IoT dove è fondamentale mantenere una comunicazione chiusa e dove è fondamentale basare il classificatore sul traffico di rete specifico del sistema su cui questo è implementato. L'assunzione alla base del funzionamento di tale metodologia è quella secondo cui il numero di client compromessi coinvolti nell'addestramento risulta essere inferiore al numero di client benevoli. Il flusso di funzionamento di tale architettura, dettagliato graficamente in 2.9, prevede di suddividere i livelli dei modelli locali in modo tale che i pesi del *k-esimo* livello, provenienti dai diversi client, siano raggruppati.

L'algoritmo prevede di clusterizzare i pesi di ogni livello in due cluster C_1 e C_2, mostrati in figura 2.10, e di trovare le corrispettive distanze euclidee massime intra-cluster D_1 e D_2, nonché la distanza D_3 tra i centroidi dei due cluster. Qualora la distanza inter-cluster fosse maggiore delle due distanze massime intra-cluster, ovvero se $D_3 > D_1$ e $D_3 > D_2$, si procederebbe ad aggregare, mediante media aritmetica, i pesi situati nel cluster di dimensione maggiore. Qualora la suddetta condizione non fosse rispettata per quel livello del modello si procederebbe al calcolo di FedAVG (2.2). Il procedimento, ripetuto per ogni livello dei modelli locali, consentirà di costruire i livelli del modello globale che verrà poi distribuito tra tutti i client partecipanti all'addestramento.

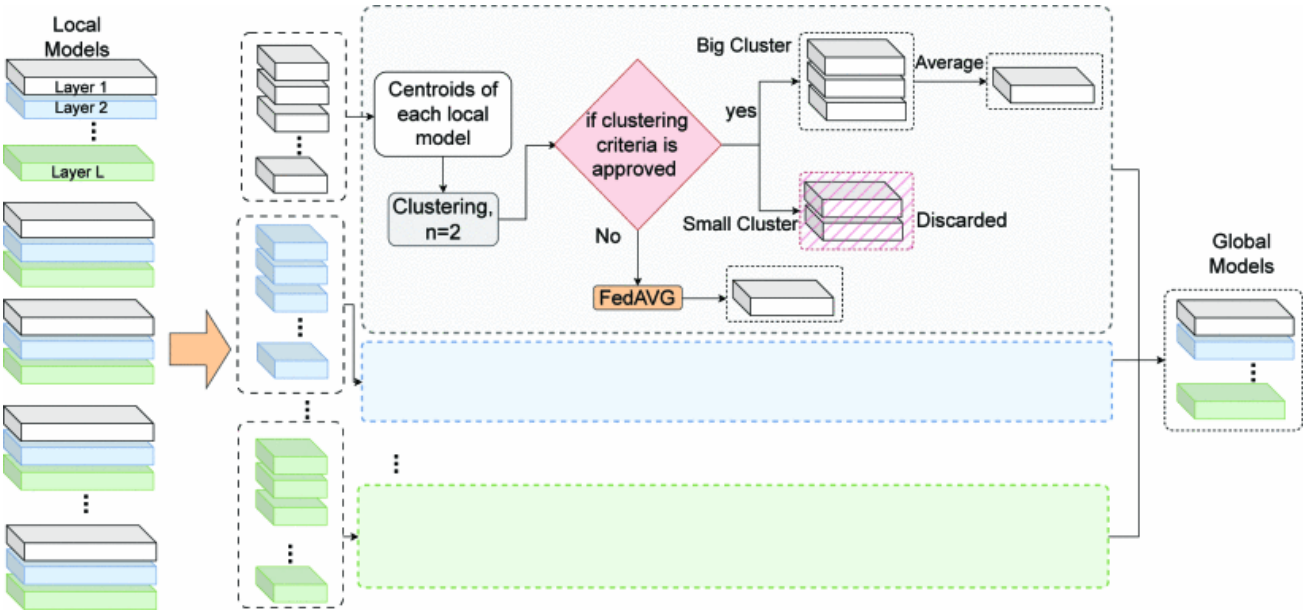


Figura 2.9: Workflow LBAA-FedAVG: I pesi provenienti da ogni modello locale vengono raggruppati in base al livello di rete a cui essi appartengono per poi essere clusterizzati. I pesi situati nel cluster con maggiore numerosità vengono poi aggregati per formare il rispettivo livello del modello globale.
Immagine tratta da: [77]

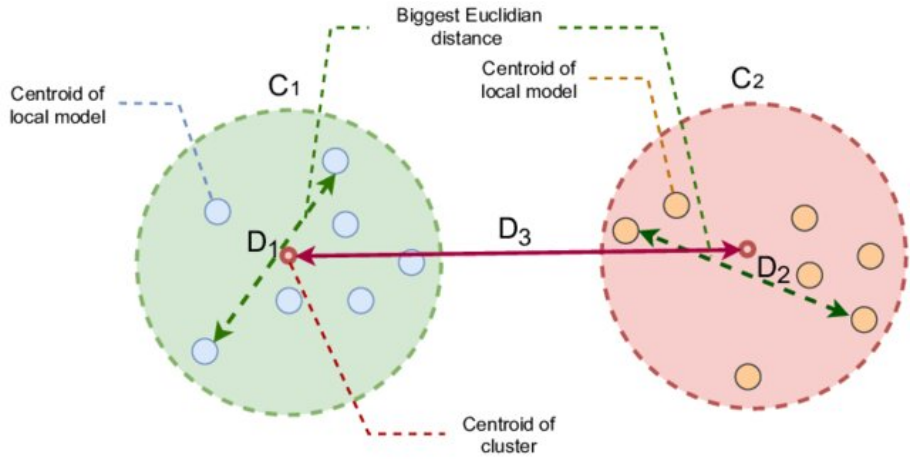


Figura 2.10: Clustering LBAA-FedAVG: Gli elementi dei cluster corrispondono ai pesi provenienti dai diversi modelli locali. Le distanze D_1 e D_2 indicano le distanze intra-cluster, mentre la distanza D_3 indica la distanza tra i centroidi dei due cluster C_1 e C_2.
Immagine tratta da: [77]

2.4.3 FLDetector

Rimanendo nel dominio degli algoritmi di aggregazione robusti al data poisoning, gli autori di [78] presentano un'architettura in grado di individuare client malevoli in base a un indice di diffidenza, denominato "suspicious score", determinato a partire dalla consistenza degli aggiornamenti del modello stimati rispetto agli aggiornamenti reali. L'idea alla base di ciò è quella per cui l'addestramento su dati anomali genera degli aggiornamenti nei parametri del modello che risultano essere differenti da quelli generati in seguito all'addestramento su dati normali. Dopo aver determinato l'indice di diffidenza di ogni client, si procede eseguendo il clustering di tali indici; in particolare, si utilizza la "Gap Statistics", presentata in [79], per determinare il numero di cluster e, se questo risulta essere superiore a uno, si prosegue utilizzando l'algoritmo KMeans per formare due cluster. Infine, si etichettano come malevoli tutti quei client appartenenti al cluster con valore medio di indice di diffidenza più alto. Tutti i client etichettati come malevoli possono essere esclusi dalla fase di apprendimento federato che consente quindi di ottenere il modello globale a partire dall'aggregazione dei modelli presenti sui soli client normali.

L'algoritmo 4 illustra i passaggi necessari alla Gap Statistics per determinare il numero di cluster.

Algoritmo 4 Gap Statistics

Input: Suspicious score dei client s, numero di campionamenti B, numero massimo di cluster K, numero di client n

Output: Numero di clusters k

for k in range(K) **do**
  $\hat{s} = \frac{s-min(s)}{max(s)-min(s)}$
  kmeans = KMeans($\hat{s}$)
  $C := \{C_1, ..., C_k\} = kmeans.clusters$
  $\mu := \{\mu_1, ..., \mu_k\} = kmeans.centroids$
  $V_k = \sum_{i=1}^{k} \sum_{x_j \in C_i} \|x_j - \mu_i\|^2$
  for b in range(B) **do**
    $x := \{x_{1b}, ..., x_{nb}\} = random([0, 1])$ # Campioni estratti da una distribuzione uniforme nel range [0,1]
    kmeans = Kmeans(x)
    $C_b := \{C_{1b}, ..., C_{kb}\} = kmeans.clusters$
    $\mu := \{\mu_{1b}, ..., \mu_{kb}\} = kmeans.centroids$
    $V^*_{kb} = \sum_{i=1}^{k} \sum_{x_{jb} \in C_{ib}} \|x_{jb} - \mu_{ib}\|^2$
  Gap(k) = $\frac{1}{B} * \sum_{i=1}^{B} \log(V^*_{kb}) - \log(V_k)$
  $v' = \frac{1}{B} * \sum_{i=1}^{B} \log(V^*_{kb})$
  $sd(k) = ((\frac{1}{B} * \sum_{i=1}^{B} (\log(V^*_{kb})) - V')^2)^{\frac{1}{2}}$
  $s'_k = \sqrt{\frac{1+B}{B}} * sd(k)$
$\hat{k} = \text{argmin}_k \ (Gap(k) - Gap(k+1) + s'_{k+1} \ \geq 0)$
return $\hat{k}$

Come illustrato nel corso delle pagine precedenti, i modelli di machine learning che sfruttano approcci di tipo non supervisionato sono una scelta appropriata per il caso di studi in questione. Un altro sistema di rilevamento delle anomalie presente in letteratura è descritto in [80] e vede l'implementazione di un Deep Autoencoder per ogni dispositivo IoT. Nonostante venga preservata la privacy dei dati, l'implementazione di un modello su un dispositivo a risorse limitate può risultare difficile e poco efficiente. L'autoencoder, addestrato solamente sul traffico

di rete normale, è in grado di ricostruire i dati normali, producendo invece alti errori di ricostruzione per quanto riguarda i dati anomali; viene dunque utilizzata una soglia sull'errore di ricostruzione per classificare i dati sul traffico di rete. Gli autori dell'articolo non prendono in considerazione la possibilità che un dispositivo sia già infetto durante la fase di addestramento, per cui è necessario assumere che ciò non accada al fine di garantire il corretto funzionamento del sistema di rilevamento delle anomalie. Un approccio che più si adatta al caso dei sistemi IoT, facendo particolare attenzione al vincolo sulla limitatezza delle risorse, è quello discusso in [58]; nello specifico, tale modello minimizza l'onere computazionale dei dispositivi pagando tuttavia un alto costo per ciò che riguarda la riservatezza dei dati e perdendo la possibilità di rilevare attacchi provenienti dall'interno del sistema. Parlando di algoritmi in grado di mitigare il fenomeno dell'avvelenamento dei dati, è possibile citare GeoMed [81] il quale, basandosi su un approccio di Federated Learning, prevede un meccanismo di aggregazione dei modelli basato sul calcolo della mediana geometrica. Lo scopo di questa tipologia di aggregazione è quello di trovare un modello globale che minimizzi la somma delle distanze da tutti i modelli locali coinvolti nell'aggregazione. Sfruttando questo principio, la presenza di client infetti durante la fase di aggregazione porta un peso minore al calcolo del modello globale.

La seguente tabella riassume le caratteristiche principali di alcune tra le architetture citate che sfruttano approcci di apprendimento non supervisionato.

Confronto Architetture				
Modello	**Centralizzato**	**Rischi privacy**	**Ottimizzazione**	
			Computazionale	**Comunicazione**
DAE N-BaIoT [80]	**No**	**No**	No	**Si**
DAE IoT Botnet AD [58]	Si	Si	**Si**	**Si**
Three-Tier SL [63]	**No**	Si	**Si**	No
FedLS [75]	**No**	Si	**Si**	No

Tabella 2.1: Confronto architetture di rilevamento delle anomalie secondo approccio di apprendimento non supervisionato

Capitolo 3

Sistema Proposto

Alla luce delle peculiarità e dei difetti riconducibili ai sistemi per la mitigazione del data poisoning presenti in letteratura, l'architettura proposta si basa sui seguenti obiettivi di progettazione:

1. **Distribuzione del modello:** Poiché il sistema di rilevamento delle anomalie sarà implementato in un sistema a risorse limitate (Sistema IoT), è necessario che i client partecipanti all'addestramento abbiano un carico di lavoro ridotto.

2. **Privacy:** I dati associati a ogni dispositivo presente nel sistema non devono in alcun modo essere trasmessi verso nodi esterni al dispositivo stesso. Tale obiettivo di design è necessario al fine di evitare esfiltrazione dei dati dovuta a possibili intercettazioni nella comunicazione.

3. **Conoscenza zero:** Il sistema deve essere in grado di determinare la presenza di client anomali senza la necessità di confronto con terze parti fidate. Tale caratteristica è necessaria al fine di rendere l'architettura proposta adatta a qualunque tipologia di sistema e, dunque, a qualunque tipologia di traffico di rete.

L'architettura proposta, denominata "Anti Poisoning Split Federated Learning" (AP-SFL), basa le sue fondamenta sugli approcci di Federated Learning e Split Learning. In particolare, la topologia architetturale prende ispirazione dall'architettura di SplitFed Learning proposta in [60] mirando però a limitare le comunicazioni tra i dispositivi, riducendo gli oneri di comunicazione. L'addestramento del modello lato client, di fatto, è separato da quello del modello lato server, evitando quasi totalmente lo scambio di smashed data e gradienti durante la fase di addestramento.

OMISSIS

Capitolo 4

Valutazioni Sperimentali

In questo capitolo sono misurate e valutate le performance raggiunte da AP-SFL. Il dominio applicativo su cui viene valutata l'architettura è quello dei sistemi IoT, motivo per cui sono stati selezionati dei dataset, presenti in letteratura, sul traffico di rete tipico di tali sistemi. A partire da ogni dataset sono stati suddivisi i dati secondo la loro categoria (Traffico normale o traffico di attacco) con la finalità di strutturare un insieme di addestramento (Su soli dati di traffico normale) e un insieme di test (Su dati misti); inoltre, per simulare la presenza di dispositivi infetti durante l'addestramento, una parte dei dati di attacco è stata riservata per essere introdotta tra i dataset di addestramento dei dispositivi.

Gli stessi esperimenti sono stati svolti su tre differenti dataset [30–32] realizzati con lo scopo di essere utilizzati per la ricerca nell'ambito della sicurezza dei sistemi IoT. Il dataset Edge IoT [30] è stato generato a partire da un ambiente di simulazione, composto da diversi dispositivi IoT, all'interno del quale sono stati avviati dei cyberattacchi appartenenti a 14 differenti categorie. La raccolta dei dati è stata effettuata mediante la generazione di file .PCAP, attraverso lo strumento di scansione di rete *Wireshark*, a partire dai quali sono poi state estratte le 61 caratteristiche maggiormente rilevanti che, in seguito a una fase di preprocessing, diventano 95. Per quanto riguarda i dataset Ton IoT [32] e Bot IoT [31], questi seguono una procedura di raccolta dati, analoga alla precedente, con la finalità di produrre dei file .PCAP sul traffico di rete; in particolare, Ton IoT conta record di attacco appartenenti a 9 categorie differenti, mentre Bot IoT può contare su 4 categorie di attacchi. Per questi ultimi due dataset è stato scelto di utilizzarne la versione preprocessata in [82] dove, partendo dai file .PCAP originali, è stato utilizzato lo strumento *ISCXFlowMeter* [83, 84] per l'estrazione delle caratteristiche rilevanti sul flusso di rete bidirezionale, ottenendo un totale di 75 caratteristiche per ognuno dei due dataset. Ogni dataset conta infine due etichette che identificano la tipologia di traffico (Normale o attacco) e l'eventuale tipologia di attacco a cui è legato quel record, più dettagli sono descritti in appendice **??**.

4.1 Configurazione degli esperimenti

Tutti gli esperimenti sono condotti sui dataset [30–32] equamente suddivisi tra tutti i client partecipanti alla fase di addestramento. In particolare, per il dataset Edge IoT vengono utilizzati 1022998 record per l'insieme di addestramento e 750273 per l'insieme di test da cui vengono estratti 279968 record di attacco per formare il training set dei client infetti. Il dataset Ton IoT, invece, conta 750000 record per l'insieme di addestramento e 750000 di record per l'insieme di test (Tale scelta è giustificata dal maggiore quantitativo dei dati di attacco presenti nel test set). Infine, il dataset Bot IoT risulta essere quello con il minor numero di record a disposizione e prevede 45000 record per l'addestramento e 18435 istanze di test.

Tutte le componenti software per condurre gli esperimenti sono state scritte in linguaggio *Python 3.10.13* con il supporto del framework *PyTorch 2.2.0* per l'implementazione dei modelli di machine learning. L'esecuzione dei test è stata suddivisa su due calcolatori di cui uno dotato di una CPU "*AMD Ryzen 5 3600x*" con 6 cores e 32GB di memoria RAM e un altro dotato di una CPU "*Intel Core i7-6700HQ*" con 4 cores, una GPU "*Nvidia GeForce GTX 950M*" e 16GB di memoria RAM.

OMISSIS

Capitolo 5

Conclusioni

Il problema degli attacchi informatici rivolti verso i sistemi IoT è molto complesso e può essere affrontato secondo approcci differenti. Gli approcci moderni si basano sull'utilizzo delle tecniche di Machine Learning e alcuni di questi sugli algoritmi di apprendimento non supervisionato, mediante cui è possibile costruire sistemi di rilevamento delle anomalie in grado apprendere il pattern dei dati normali e individuare la presenza di dati riconducibili a un attacco informatico. Questi approcci, basandosi unicamente sui dati forniti come input, sono piuttosto sensibili alla presenza di dati anomali all'interno dell'insieme di addestramento, motivo per cui è fondamentale che la fase di apprendimento sia sicura. Poiché sistemi di questo tipo prevedono un approccio di apprendimento distribuito, in cui ogni dispositivo produce i propri dati, una vulnerabilità riguarda la possibilità che uno o più dispositivi siano già infetti durante la fase di addestramento e che quindi producano traffico di rete anomalo. Questa condizione porterebbe il modello di machine learning ad apprendere un pattern di dati che comprende sia il traffico di rete tipico della rete che quello malevolo, identificando il tutto come normale e non riuscendo più a distinguere correttamente i dati anomali.

Gli esperimenti condotti hanno mostrato come AP-SFL sia in grado di fronteggiare il problema dell'avvelenamento dei dati all'interno di un sistema IoT. I risultati ottenuti confermano la predominanza di AP-SFL su altri modelli allo stato dell'arte che prevedono meccanismi di difesa dagli attacchi di data poisoning ma che non hanno una topologia adatta a essere implementati all'interno di un sistema con dispositivi a risorse limitate.

La capacità di adattarsi al sistema su cui è implementata rende l'architettura AP-SFL estremamente versatile; inoltre, la modularità delle sue componenti consente di apportare delle modifiche a diverse parti dell'architettura senza le necessità di dover rielaborare il modulo di filtraggio. Queste peculiarità consentono l'implementazione di AP-SFL in scenari applicativi reali come, per esempio, nel contesto delle Smart Home in cui la tipologia di traffico di rete è specifica per il sistema in oggetto e risulta essere estremamente variabile. Analogamente, l'architettura in questione rappresenta un ottimo strumento di difesa per tutte le applicazioni IoT in ambito industriale (IIoT) dove viene impiegata una grande mole di dispositivi le cui capacità di calcolo risultano limitate.

AP-SFL ha mostrato risultati tali da renderla competitiva con gli approcci allo stato dell'arte più recenti. Data la sua promettente capacità di rendere i sistemi IoT robusti agli attacchi informatici, l'idea di rendere AP-SFL maggiormente flessibile e performante rientra tra i possibili sviluppi futuri dell'architettura in questione.

Una modifica volta a migliorare il sistema riguarda la possibilità di sviluppare il processo di addestramento su più fasi di filtraggio. L'idea è quella di utilizzare solamente una parte delle LSRs dei client a ogni fase, non incrementando troppo gli oneri computazionali. A ogni fase di filtraggio viene eseguita l'operazione di clustering sui client e, al termine di questa, vengono

etichettati i client individuati come infetti. Al termine di tutte le fasi di filtraggio, saranno esclusi dalla fase di addestramento i client etichettati più volte. La modifica ha come scopo quello di evitare l'esclusione di client normali che sono stati erroneamente identificati come infetti; di fatto, l'esclusione di client normali può avere un impatto negativo e non indifferente sulle performance del modello.

Altri possibili sviluppi possono prevedere l'utilizzo di differenti modelli di machine learning alla base dell'architettura; introducendo, per esempio, varianti di autoencoder come quella citata in [85].

Elenco delle figure

Elenco delle tabelle

Bibliografia

[1] J. Y. Khan. "Introduction to IoT systems". In: *Internet of Things (IoT)*. Jenny Stanford Publishing, 2019, pp. 1–24.

[2] T. Xu et al. "Security of IoT systems: Design challenges and opportunities". In: *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2014, pp. 417–423. DOI: 10.1109/ICCAD.2014.7001385.

[3] W. Lv et al. "A General Architecture of IoT System". In: *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*. Vol. 1. 2017, pp. 659–664. DOI: 10.1109/CSE-EUC.2017.124.

[4] P. Sethi et al. "Internet of things: architectures, protocols, and applications". In: *Journal of electrical and computer engineering* 2017.1 (2017), p. 9324035.

[5] L. Hou et al. "Internet of Things Cloud: Architecture and Implementation". In: *IEEE Communications Magazine* 54.12 (2016), pp. 32–39. DOI: 10.1109/MCOM.2016.1600398CM.

[6] P. P. Ray. "A survey of IoT cloud platforms". In: *Future Computing and Informatics Journal* 1.1-2 (2016), pp. 35–46.

[7] M. Aazam et al. "Fog computing and smart gateway based communication for cloud of things". In: *2014 International conference on future internet of things and cloud*. IEEE. 2014, pp. 464–470.

[8] H. F. Atlam et al. "Fog computing and the internet of things: A review". In: *big data and cognitive computing* 2.2 (2018), p. 10.

[9] F. Bonomi et al. "Fog computing: A platform for internet of things and analytics". In: *Big data and internet of things: A roadmap for smart environments* (2014), pp. 169–186.

[10] S. Vishnu et al. "Internet of medical things (IoMT)-An overview". In: *2020 5th international conference on devices, circuits and systems (ICDCS)*. IEEE. 2020, pp. 101–104.

[11] H. Boyes et al. "The industrial internet of things (IIoT): An analysis framework". In: *Computers in industry* 101 (2018), pp. 1–12.

[12] I.-I. Pătru et al. "Smart home IoT system". In: *2016 15th RoEduNet Conference: Networking in Education and Research*. 2016, pp. 1–6. DOI: 10.1109/RoEduNet.2016.7753232.

[13] C. Kolias et al. "DDoS in the IoT: Mirai and other botnets". In: *Computer* 50.7 (2017), pp. 80–84.

[14] D. Kwon et al. "A survey of deep learning-based network anomaly detection". In: *Cluster Computing* 22 (2019), pp. 949–961.

[15] M. Ahmed et al. "A survey of network anomaly detection techniques". In: *Journal of Network and Computer Applications* 60 (2016), pp. 19–31.

[16] A. Sehgal et al. “Management of resource constrained devices in the internet of things”. In: *IEEE Communications Magazine* 50.12 (2012), pp. 144–149. DOI: 10.1109/MCOM.2012.6384464.

[17] J. Verbraeken et al. “A survey on distributed machine learning”. In: *Acm computing surveys (csur)* 53.2 (2020), pp. 1–33.

[18] J. Wang et al. “Collaborative machine learning: Schemes, robustness, and privacy”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[19] P. Vepakomma et al. “Split learning for health: Distributed deep learning without sharing raw patient data”. In: *arXiv preprint arXiv:1812.00564* (2018).

[20] D. C. Nguyen et al. “Federated learning for internet of things: A comprehensive survey”. In: *IEEE Communications Surveys & Tutorials* 23.3 (2021), pp. 1622–1658.

[21] C. Zhang et al. “A survey on federated learning”. In: *Knowledge-Based Systems* 216 (2021), p. 106775.

[22] H. B. Barlow. “Unsupervised learning”. In: *Neural computation* 1.3 (1989), pp. 295–311.

[23] Z. Ghahramani. “Unsupervised learning”. In: *Summer school on machine learning*. Springer, 2003, pp. 72–112.

[24] D. Venugopal et al. “Efficient signature based malware detection on mobile devices”. In: *Mobile Information Systems* 4.1 (2008), pp. 33–49.

[25] K. Griffin et al. “Automatic generation of string signatures for malware detection”. In: *Recent Advances in Intrusion Detection: 12th International Symposium, RAID 2009, Saint-Malo, France, September 23-25, 2009. Proceedings 12*. Springer. 2009, pp. 101–120.

[26] Y. Ye et al. “An intelligent PE-malware detection system based on association mining”. In: *Journal in computer virology* 4 (2008), pp. 323–334.

[27] E. Gandotra et al. “Zero-day malware detection”. In: *2016 Sixth international symposium on embedded computing and system design (ISED)*. IEEE. 2016, pp. 171–175.

[28] A. De Paola et al. “Malware Detection through Low-level Features and Stacked Denoising Autoencoders.” In: *ITASEC*. 2018.

[29] H.-J. Liao et al. “Intrusion detection system: A comprehensive review”. In: *Journal of Network and Computer Applications* 36.1 (2013), pp. 16–24.

[30] M. A. Ferrag et al. “Edge-IIoTset: A new comprehensive realistic cyber security dataset of IoT and IIoT applications for centralized and federated learning”. In: *IEEE Access* 10 (2022), pp. 40281–40306.

[31] N. Koroniotis et al. *Towards the Development of Realistic Botnet Dataset in the Internet of Things for Network Forensic Analytics: Bot-IoT Dataset*. 2018.

[32] A. Alsaedi et al. “TON_IoT telemetry dataset: A new generation dataset of IoT and IIoT for data-driven intrusion detection systems”. In: *Ieee Access* 8 (2020), pp. 165130–165150.

[33] K. O’shea et al. “An introduction to convolutional neural networks”. In: *arXiv preprint arXiv:1511.08458* (2015).

[34] S. Selva Birunda et al. “A review on word embedding techniques for text classification”. In: *Innovative Data Communication Technologies and Application: Proceedings of ICIDCA 2020* (2021), pp. 267–281.

[35] J. Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[36] A. Radford et al. “Improving language understanding by generative pre-training”. In: (2018).

[37] B. J. Radford et al. “Network traffic anomaly detection using recurrent neural networks”. In: *arXiv preprint arXiv:1803.10769* (2018).

[38] L. Medsker et al. *Recurrent neural networks: design and applications*. CRC press, 1999.

[39] C. Pelletier et al. “Temporal convolutional neural network for the classification of satellite image time series”. In: *Remote Sensing* 11.5 (2019), p. 523.

[40] Z. Wu et al. “A comprehensive survey on graph neural networks”. In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 4–24.

[41] P. Veličković et al. “Graph attention networks”. In: *arXiv preprint arXiv:1710.10903* (2017).

[42] W. Jia et al. “Anomaly Detection using Supervised Learning and Multiple Statistical Methods”. In: *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. 2019, pp. 1291–1297. DOI: 10.1109/ICMLA.2019.00211.

[43] M. Abdel-Basset et al. “Efficient and lightweight convolutional networks for IoT malware detection: A federated learning approach”. In: *IEEE Internet of Things Journal* 10.8 (2022), pp. 7164–7173.

[44] F. Zhuang et al. “A comprehensive survey on transfer learning”. In: *Proceedings of the IEEE* 109.1 (2020), pp. 43–76.

[45] J. Fu et al. “Exploring adapter-based transfer learning for recommender systems: Empirical studies and practical insights”. In: *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*. 2024, pp. 208–217.

[46] K. Weiss et al. “A survey of transfer learning”. In: *Journal of Big data* 3 (2016), pp. 1–40.

[47] J. E. Van Engelen et al. “A survey on semi-supervised learning”. In: *Machine learning* 109.2 (2020), pp. 373–440.

[48] X. J. Zhu. “Semi-supervised learning literature survey”. In: (2005).

[49] X. Pei et al. “A knowledge transfer-based semi-supervised federated learning for IoT malware detection”. In: *IEEE Transactions on Dependable and Secure Computing* (2022).

[50] L. Ruff et al. “Deep semi-supervised anomaly detection”. In: *arXiv:1906.02694* (2019).

[51] T. Qin et al. “Hier-SFL: Client-edge-cloud collaborative traffic classification framework based on hierarchical federated split learning”. In: *Future Generation Computer Systems* 149 (2023), pp. 12–24.

[52] R. Zang et al. “MLAD: A Unified Model for Multi-system Log Anomaly Detection”. In: *arXiv preprint arXiv:2401.07655* (2024).

[53] S. Wang et al. “Machine learning in network anomaly detection: A survey”. In: *IEEE Access* 9 (2021), pp. 152379–152396.

[54] F. T. Liu et al. “Isolation forest”. In: *2008 eighth ieee international conference on data mining*. IEEE. 2008, pp. 413–422.

[55] W. S. Al Farizi et al. “Isolation forest based anomaly detection: A systematic literature review”. In: *2021 8th International Conference on Information Technology, Computer and Electrical Engineering (ICITACEE)*. IEEE. 2021, pp. 118–122.

[56] M. Hejazi et al. “One-class support vector machines approach to anomaly detection”. In: *Applied Artificial Intelligence* 27.5 (2013), pp. 351–366.

[57] M. Amer et al. “Enhancing one-class Support Vector Machines for unsupervised anomaly detection”. In: ago. 2013, pp. 8–15. DOI: 10.1145/2500853.2500857.

[58] I. Apostol et al. “IoT botnet anomaly detection using unsupervised deep learning”. In: *Electronics* 10.16 (2021), p. 1876.

[59] S. A. Rahman et al. “Internet of Things Intrusion Detection: Centralized, On-Device, or Federated Learning?” In: *IEEE Network* 34.6 (2020), pp. 310–317. DOI: 10.1109/MNET.011.2000286.

[60] C. Thapa et al. “Splitfed: When federated learning meets split learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 8. 2022, pp. 8485–8493.

[61] A. Brecko et al. “Federated Learning for Edge Computing: A Survey”. In: *Applied Sciences* 12.18 (2022). ISSN: 2076-3417. DOI: 10.3390/app12189124.

[62] L. Liu et al. “Client-edge-cloud hierarchical federated learning”. In: *ICC 2020-2020 IEEE international conference on communications (ICC)*. IEEE. 2020, pp. 1–6.

[63] A. Alromih et al. “Privacy-aware split learning based energy theft detection for smart grids”. In: *International Conference on Information and Communications Security*. Springer. 2022, pp. 281–300.

[64] X. Gao et al. “{PCAT}: Functionality and data stealing from split learning by {Pseudo-Client} attack”. In: *32nd USENIX Security Symposium (USENIX Security 23)*. 2023, pp. 5271–5288.

[65] N. Z. Gong et al. “You are who you know and how you behave: Attribute inference attacks via users’ social friends and behaviors”. In: *25th USENIX Security Symposium (USENIX Security 16)*. 2016, pp. 979–995.

[66] N. Z. Gong et al. “Attribute inference attacks in online social networks”. In: *ACM Transactions on Privacy and Security (TOPS)* 21.1 (2018), pp. 1–30.

[67] J. Jia et al. “{AttriGuard}: A practical defense against attribute inference attacks via adversarial machine learning”. In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018, pp. 513–529.

[68] M. Fredrikson et al. “Model inversion attacks that exploit confidence information and basic countermeasures”. In: *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 2015, pp. 1322–1333.

[69] E. Erdoğan et al. “Unsplit: Data-oblivious model inversion, model stealing, and label inference attacks against split learning”. In: *Proceedings of the 21st Workshop on Privacy in the Electronic Society*. 2022, pp. 115–124.

[70] D. Pasquini et al. “Unleashing the tiger: Inference attacks on split learning”. In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021, pp. 2113–2129.

[71] A. Roy Chowdhury et al. “Eiffel: Ensuring integrity for federated learning”. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2022, pp. 2535–2549.

[72] S. Prakash et al. “Mitigating byzantine attacks in federated learning”. In: *arXiv preprint arXiv:2010.07541* (2020).

[73] M. A. Belay et al. “Unsupervised anomaly detection for IoT-based multivariate time series: Existing solutions, performance analysis and future directions”. In: *Sensors* 23.5 (2023), p. 2844.

[74] G. Bovenzi et al. “Data poisoning attacks against autoencoder-based anomaly detection models: A robustness analysis”. In: *ICC 2022-IEEE International Conference on Communications*. IEEE. 2022, pp. 5427–5432.

[75] T. D. Luong et al. “FedLS: An Anti-poisoning Attack Mechanism for Federated Network Intrusion Detection Systems Using Autoencoder-Based Latent Space Representations”. In: *International Conference on Information Security Practice and Experience*. Springer. 2023, pp. 17–35.

[76] A. Gretton et al. “Measuring statistical dependence with Hilbert-Schmidt norms”. In: *International conference on algorithmic learning theory*. Springer. 2005, pp. 63–77.

[77] H. U. Manzoor et al. “Defending Federated Learning from Backdoor Attacks: Anomaly-Aware FedAVG with Layer-Based Aggregation”. In: *2023 IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. IEEE. 2023, pp. 1–6.

[78] Z. Zhang et al. “Fldetector: Defending federated learning against model poisoning attacks via detecting malicious clients”. In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2022, pp. 2545–2555.

[79] R. Tibshirani et al. “Estimating the number of clusters in a data set via the gap statistic”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63.2 (2001), pp. 411–423.

[80] Y. Meidan et al. “N-baiot—network-based detection of iot botnet attacks using deep autoencoders”. In: *IEEE Pervasive Computing* 17.3 (2018), pp. 12–22.

[81] K. Pillutla et al. “Robust aggregation for federated learning”. In: *IEEE Transactions on Signal Processing* 70 (2022), pp. 1142–1154.

[82] M. Sarhan et al. “Evaluating Standard Feature Sets Towards Increased Generalisability and Explainability of ML-Based Network Intrusion Detection”. In: *Big Data Research* 30 (nov. 2022), p. 100359. ISSN: 2214-5796. DOI: 10.1016/j.bdr.2022.100359.

[83] G. Draper-Gil. et al. “Characterization of Encrypted and VPN Traffic using Time-related Features”. In: *Proceedings of the 2nd International Conference on Information Systems Security and Privacy - ICISSP*. INSTICC. SciTePress, 2016, pp. 407–414. ISBN: 978-989-758-167-0. DOI: 10.5220/0005740704070414.

[84] A. H. Lashkari et al. “Characterization of tor traffic using time based features”. In: *International Conference on Information Systems Security and Privacy*. Vol. 2. SciTePress. 2017, pp. 253–262.

[85] D. Gong et al. “Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 1705–1714.

[86] F. Concone et al. “Twitter Analysis for Real-time Malware Discovery”. In: *2017 AEIT International Annual Conference (2017 AEIT)*. Cagliari, Italy, set. 2017.

[87] A. D. Paola et al. “A hybrid system for malware detection on big data”. In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. Apr. 2018, pp. 45–50. DOI: 10.1109/INFCOMW.2018.8406963.

[88] V. Agate et al. “A Behavior-Based Intrusion Detection System Using Ensemble Learning Techniques”. In: 2022.

[89] V. Agate et al. “Enhancing IoT Network Security with Concept Drift-Aware Unsupervised Threat Detection”. In: *2024 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2024.

[90] A. Augello et al. “NEP-IDS: a Network Intrusion Detection System Based on Entropy Prediction Error”. In: *2024 IEEE 49th Conference on Local Computer Networks (LCN)*. 2024, pp. 1–9. ISBN: 979-8-3503-8800-8. DOI: 10.1109/LCN60385.2024.10639755.

[91] V. Agate et al. “Adaptive Ensemble Learning for Intrusion Detection Systems”. In: *CEUR Workshop Proceedings*. Vol. 3762. 2024.

[92] A. Augello et al. “Tackling Selfish Clients in Federated Learning”. In: *27th European Conference on Artificial Intelligence (ECAI 2024)*. IOS Press, 2024, pp. 1888–1895. ISBN: 978-1-64368-548-9. DOI: 10.3233/FAIA240702.

[93] C. Zhou et al. “Anomaly detection with robust deep autoencoders”. In: *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 2017, pp. 665–674.

[94] L. Liu et al. “Client-edge-cloud hierarchical federated learning”. In: *ICC 2020-2020 IEEE international conference on communications (ICC)*. IEEE. 2020, pp. 1–6.

[95] S. Boyd et al. “Distributed optimization and statistical learning via the alternating direction method of multipliers”. In: *Foundations and Trends® in Machine learning* 3.1 (2011), pp. 1–122.

[96] A. Panda et al. “Sparsefed: Mitigating model poisoning attacks in federated learning with sparsification”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2022, pp. 7587–7624.

[97] J. Dip Das et al. “Encoder–Decoder Based LSTM and GRU Architectures for Stocks and Cryptocurrency Prediction”. In: *Journal of Risk and Financial Management* 17.5 (2024). ISSN: 1911-8074. DOI: 10.3390/jrfm17050200.

[98] M. Ahmed et al. “The k-means Algorithm: A Comprehensive Survey and Performance Evaluation”. In: *Electronics* 9.8 (2020). ISSN: 2079-9292. DOI: 10.3390/electronics9081295.

[99] I. Stellios et al. “A survey of iot-enabled cyberattacks: Assessing attack paths to critical infrastructures and services”. In: *IEEE Communications Surveys & Tutorials* 20.4 (2018), pp. 3453–3495.

[100] V. S. Sathyanarayan et al. “Signature generation and detection of malware families”. In: *Information Security and Privacy: 13th Australasian Conference, ACISP 2008, Wollongong, Australia, July 7-9, 2008. Proceedings 13*. Springer. 2008, pp. 336–349.

[101] J. Scott. “Signature based malware detection is dead”. In: *Institute for Critical Infrastructure Technology* (2017).

[102] A. A. Elhadi et al. “Malware detection based on hybrid signature behaviour application programming interface call graph”. In: *American Journal of Applied Sciences* 9.3 (2012), p. 283.

[103] T. Mikolov et al. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[104] A. Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[105] J. Hestness et al. *Deep Learning Scaling is Predictable, Empirically*. 2017.

[106] M. Rapp et al. “Distributed Learning on Heterogeneous Resource-Constrained Devices”. In: *CoRR* abs/2006.05403 (2020).

[107] Y. Meidan et al. “N-baiot—network-based detection of iot botnet attacks using deep autoencoders”. In: *IEEE Pervasive Computing* 17.3 (2018), pp. 12–22.

[108] V. Chandola et al. “Anomaly detection: A survey”. In: *ACM computing surveys (CSUR)* 41.3 (2009), pp. 1–58.

[109] J. Auskalnis et al. “Application of local outlier factor algorithm to detect anomalies in computer network”. In: *Elektronika ir Elektrotechnika* 24.3 (2018), pp. 96–99.

[110] J. White et al. “Unsupervised one-class learning for anomaly detection on home IoT network devices”. In: *2021 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*. IEEE. 2021, pp. 1–8.

[111] K. Sadaf et al. “Intrusion Detection Based on Autoencoder and Isolation Forest in Fog Computing”. In: *IEEE Access* 8 (2020), pp. 167059–167068. DOI: 10.1109/ACCESS.2020.3022855.

[112] G. Draper-Gil et al. “Characterization of encrypted and vpn traffic using time-related”. In: *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*. 2016, pp. 407–414.

[113] S. Boyd et al. “Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers”. In: *Found. Trends Mach. Learn.* 3.1 (gen. 2011), pp. 1–122. ISSN: 1935-8237. DOI: 10.1561/2200000016.

[114] M. Wu et al. “Split learning with differential privacy for integrated terrestrial and non-terrestrial networks”. In: *IEEE Wireless Communications* (2023).