UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA



Corso di Laurea Magistrale in
Ingegneria Informatica e dell'Automazione

Architettura Zero-Trust basata su Bidirectional Long Short-Term Memory e Deep Reinforcement Learning

Zero-Trust Architecture based on Bidirectional Long Short-Term Memory and Deep Reinforcement Learning

Relatore:
Prof. Spalazzi Luca
Correlatore:
Dott. Bonifazi Gianluca

Laureando:
Bocci Matteo

Anno Accademico 2025-2026

Abstract

La crescente distribuzione delle infrastrutture informatiche e il superamento dei tradizionali confini di rete hanno reso sempre più rilevante l'adozione del paradigma Zero Trust, basato sul principio di verifica continua delle richieste di accesso e sulla valutazione dinamica del rischio. In tale contesto, la presente tesi propone la progettazione e lo sviluppo di un'architettura di difesa automatizzata che integra tecniche di Deep Learning e Deep Reinforcement Learning per associare al rilevamento di attività sospette una risposta di sicurezza proporzionata al contesto osservato.
La soluzione realizzata è organizzata in una pipeline composta inizialmente dall'analisi mediante una rete Bidirectional Long Short-Term Memory (**BiLSTM**). Il modello svolge il compito di classificare il comportamento osservato tra attività legittime e sospette, producendo una probabilità associata alla presenza di un attacco. Tale informazione viene successivamente integrata con caratteristiche temporali e contestuali all'interno dello stato di un agente di Reinforcement Learning basato su Deep Q-Network (**DQN**), incaricato di selezionare automaticamente la strategia di mitigazione più appropriata tra diverse azioni di risposta.
La valutazione sperimentale, condotta sul dataset di eventi di sicurezza utilizzato nel progetto, ha mostrato prestazioni molto elevate nella capacità della BiLSTM di distinguere i comportamenti normali da quelli sospetti. Le analisi della componente DQN hanno inoltre evidenziato una capacità di selezionare in modo coerente differenti strategie di risposta. È stata inoltre analizzata la modalità online della pipeline, nella quale i nuovi eventi vengono valutati in funzione della storia disponibile. Quando non è presente una quantità sufficiente di informazioni temporali per costruire una sequenza valida, il sistema ricorre a una politica di cold-start basata sulle caratteristiche contestuali dell'evento.
Nel complesso, il lavoro mostra la fattibilità di un'architettura nella quale la fase di rilevazione e quella di risposta vengono mantenute distinte ma integrate all'interno dello stesso processo decisionale. La soluzione consente quindi di passare dall'individuazione di un possibile comportamento malevolo alla selezione automatica di una contromisura, in accordo con il contesto osservato. I risultati ottenuti devono tuttavia essere interpretati considerando i limiti del dataset sintetico e le semplificazioni introdotte rispetto a un ambiente operativo reale.

Indice

Abstract **3**

1 Introduzione **9**
1.1 Contesto di progetto 9
1.2 Struttura capitoli tesi 9
1.3 Il paradigma perimetrale 10
1.4 Zero Trust 11
1.4.1 Principi Zero Trust 12
1.4.2 Architettura Zero Trust 12
1.5 Obiettivo di progetto 14

2 Stato dell'arte **15**
2.1 Machine Learning nella cybersecurity 15
2.1.1 Sfide e limitazioni 17
2.1.2 Apprendimento supervisionato e classificazione 18
2.1.3 Feature engineering 19
2.1.4 Dati sequenziali e reti ricorrenti 19
2.2 LSTM 21
2.2.1 Problematiche delle LSTM 23
2.2.2 Bidirectional LSTM 23
2.3 Reinforcement Learning 25
2.3.1 Q-Learning 27
2.3.2 Problematiche del Q-learning 28
2.4 DQN 29

3 Dataset e strumenti software **33**
3.1 Dataset 33
3.1.1 Blazor:2.0.0.2:ApplicationAudit/AuthAudit 35
3.1.2 Captive Portal 38
3.1.3 p0f 39
3.2 Analisi esplorativa del dataset 40
3.2.1 Dimensione e composizione del dataset 40
3.2.2 Distribuzione di IP, utenti e sessioni 41

3.2.3 Analisi degli eventi sospetti per IP e utente 42
3.2.4 Concentrazione degli eventi sospetti 43
3.2.5 Analisi temporale 44
3.2.6 Analisi codici HTTP 46
3.2.7 Utilizzo dei certificati client 47
3.3 Considerazioni sull'analisi esplorativa 48
3.4 Strumenti operativi 49
3.4.1 TensorFlow / Keras 49
3.4.2 Scikit-learn 51
3.4.3 Gymnasium 52
3.4.4 Matplotlib 53

4 Progettazione architettura **55**
4.1 Architettura generale 55
4.2 Pipeline di preprocessing 56
4.2.1 Correlazione 56
4.2.2 Creazione log finali 57
4.2.3 Encoding variabili categoriche 59
4.2.4 Feature Engineering 59
4.2.5 Costruzione sequenze 62
4.3 Progettazione e implementazione BiLSTM 64
4.3.1 Architettura della rete 64
4.3.2 Configurazione dell'addestramento 67
4.3.3 Callbacks e gestione dell'addestramento 69
4.3.4 Metriche di valutazione 70
4.4 Progettazione e implementazione DQN 72
4.4.1 Definizione dello stato RL 72
4.4.2 Normalizzazione dello stato 73
4.4.3 Progettazione e implementazione dell'ambiente RL 74
4.4.4 Architettura della rete DQN 81
4.4.5 Configurazione e addestramento del DQN 82
4.5 Pipeline completa 84
4.5.1 Flusso completo di elaborazione 84
4.5.2 Gestione di un nuovo evento 87
4.5.3 Modalità Cold-Start 88

5 Risultati **91**
5.1 Valutazione BiLSTM 92
5.2 Analisi dello stato utilizzato dalla DQN 94
5.3 Valutazione DQN 95
5.3.1 Interazione BiLSTM–DQN 97
5.3.2 Differente scenario sperimentale 99
5.4 Valutazione modalità online 101

5.4.1 Caso COLD START nuova entità 101
5.4.2 Caso COLD START entità esistente 104
5.4.3 Caso FULL ANALYSIS 106

6 Conclusioni e sviluppi futuri **111**
6.1 Conclusioni 111
6.2 Sviluppi futuri 113

Bibliografia **115**

Elenco delle figure **119**

Elenco delle tabelle **122**

Capitolo 1

Introduzione

1.1 Contesto di progetto

Negli ultimi due decenni la sicurezza informatica ha subito una profonda evoluzione che ha completamente stravolto il modo di giudicare e affrontare le varie minacce digitali. Questo cambiamento è stato guidato dalla progressiva digitalizzazione di servizi e processi aziendali, dalla trasformazione delle infrastrutture IT, dall'adozione pesante del *cloud computing* e infine dall'aumento delle minacce informatiche. La maggiore distribuzione e complessità delle infrastrutture ha ovviamente portato vantaggi in termini di flessibilità e disponibilità di servizi, ma dall'altro lato ha reso più complicata la gestione della sicurezza. Con la possibilità di accesso ad applicazioni e dati anche da remoto, tramite dispositivi personali e aziendali, si rende necessario garantire che queste entità esterne non siano di natura malevola e non comportino danni per l'intero sistema.
In questo contesto, i modelli di sicurezza tradizionali, basati sulla protezione del perimetro di rete, si sono progressivamente rivelati insufficienti. Di conseguenza, si è affermato un nuovo approccio, noto come ***Zero Trust***, il quale verrà illustrato nel paragrafo 1.4 che ridefinisce il concetto stesso di fiducia, la cosiddetta ***Trustness***.

1.2 Struttura capitoli tesi

La seguente tesi è suddivisa in 6 capitoli. Il capitolo 1 introduce il contesto generale alla base del progetto e le motivazioni e obiettivo del lavoro. Nel capitolo 2 viene mostrato lo stato dell'arte attuale, facendo una panoramica relativa alle principali tecnologie utilizzate e sulle motivazioni di tali scelte. Il capitolo 3 è responsabile di illustrare i principali strumenti software utilizzati nello sviluppo dell'architettura, e nel presentare il dataset utilizzato, analizzandone la struttura ed eseguendo un'analisi esplorativa.

Il capitolo 4 rappresenta il cuore della tesi, dove viene descritta completamente la progettazione dell'architettura nelle sue diverse fasi e la sua implementazione, dalla fase di preprocessing e correlazione dei log alla definizione delle feature, alla costruzione delle sequenze e alla progettazione e addestramento dei modelli BiLSTM e DQN. Il capitolo 5 vede l'esposizione dei risultati ottenuti attraverso la valutazione sperimentale del sistema. Infine nel capitolo 6 vengono raccolte le conclusioni del progetto con le relative considerazioni su limitazioni e possibili sviluppi futuri.

1.3 Il paradigma perimetrale

Per moltissimo tempo, le principali strategie di difesa delle organizzazioni si sono basate sul paradigma perimetrale. Questo è un modello di protezione e sicurezza che si basa sull'esistenza di un confine netto ben preciso tra un qualcosa di "interno" come una rete interna, considerata fidata e sicura e un qualcosa di "esterno", considerato invece non affidabile e non sicuro, come Internet.
L'esempio classico di questo paradigma è il modello "*Castello e Fossato*", che così come il fossato con i coccodrilli che circonda il castello del re, difende solo i confini esterni considerando sicuro tutto quello che si trova all'interno e mettendo una barriera di sicurezza con ciò che proviene da fuori.
La protezione viene proprio concentrata sui punti di accesso alla rete, in modo da filtrare il traffico in ingresso e impedire accessi non autorizzati dall'esterno. Un aspetto rilevante di questo approccio è il fatto di avere una distinzione sostanziale tra utenti interni ed esterni; una volta aver superato il perimetro con i relativi controlli, all'utente viene assegnato un alto livello di fiducia che gli permette di muoversi liberamente all'interno della rete. Nel caso quindi in cui un utente malevolo fosse riuscito ad oltrepassare il perimetro avrebbe potuto operare all'interno della rete con relativa libertà, aumentando anche i suoi privilegi e rimanendo all'interno in maniera prolungata senza essere identificato.
La sicurezza di questo modello viene implementata attraverso *firewall*, capaci di filtrare il traffico in ingresso, sistemi di rilevamento e prevenzioni delle intrusioni *IDS/IPS*, *VPN* per accesso remoto sicuro e zone demilitarizzate (*DMZ*) per isolare i servizi esposti.
Come anticipato precedentemente, con il passare del tempo questo paradigma ha mostrato diversi limiti:

- difficoltà nella definizione di un perimetro chiaro negli ambienti distribuiti;
- esposizione a minacce interne relative ad utenti autorizzati;
- facilità di movimento laterale per gli attaccanti una volta superato il perimetro;
- aumento della superficie di attacco a causa dell'accesso remoto con VPN e connessioni esterne.

1.4 Zero Trust

Il momento storico della svolta in cui si è abbandonata la sicurezza perimetrale non è una data esatta ma due tappe fondamentali, ed è riconducibile al periodo temporale 2004-2010. Nel 2004 infatti un gruppo di esperti di sicurezza appartenente ad un consorzio internazionale noto come *Jericho Forum* [1] ha coniato il termine ***de-perimetrizzazione***, con l'obiettivo, successivamente raggiunto, di superare il tradizionale modello perimetrale, mettendo per primi in discussione i problemi relativi ai firewall e ai confini di rete aziendali che non erano più sufficienti a garantire protezione a causa dell'avvento del cloud. Si ritiene tutt'oggi che il lavoro del Jericho Forum abbia gettato le basi per la nascita del paradigma Zero Trust. Successivamente, nel 2010, *John Kindervag* oggi uno dei massimi esperti di sicurezza informatica, allora analista, ha utilizzato il termine "Zero Trust model" e formalizzato i principi e l'architettura di questo nuovo modello sancendo l'abbandono della difesa perimetrale. Un fondamentale contributo è stato dato anche da *Google* che sempre nel 2010, a seguito di un attacco informatico avvia l'iniziativa *BeyondCorp* con l'obiettivo di garantire accesso alle risorse in base all'identità e al contesto, permettendo di dimostrare al mondo come fosse possibile offrire sicurezza da qualsiasi rete ai propri dipendenti senza l'utilizzo di VPN tradizionali [2].
Il paradigma Zero Trust [3] si basa sul principio fondamentale *"Never trust, always verify"* ovvero non fidarti mai, verifica sempre.
Se nella cybersecurity classica valeva il celebre aforisma di *Eugene Spafford*, secondo cui «l'unico computer sicuro è un computer spento», oggigiorno con l'attuale operatività aziendale questo è impraticabile. Questo modello non cerca l'utopia del sistema isolato e spento, ma accetta la vulnerabilità intrinseca del sistema acceso, mitigandola attraverso questo principio.
In pratica non esiste nessuna fiducia intrinseca all'interno di una rete, per qualsiasi utente, dispositivo o connessione. Una richiesta di accesso non è considerata automaticamente affidabile solamente perché proviene da un utente, dispositivo o rete già autorizzata in passato. Questo, se così non fosse, provocherebbe molte minacce a causa di attaccanti in grado di scavalcare il perimetro di difesa. Ogni richiesta, ogni risorsa, utente o interfaccia, sia interna o esterna deve essere trattata con lo stesso livello di sospetto e valutata in modo continuo. Non effettuando una distinzione tra interno ed esterno, ogni richiesta viene valutata in base a diversi fattori come l'identità dell'utente e i suoi comportamenti, lo storico delle richieste effettuate, il dispositivo utilizzato, la posizione geografica, la risorsa richiesta e gli esiti passati. Tutti questi dati servono ad individuare il contesto completo della richiesta e permettere una decisione più accurata.

1.4.1 Principi Zero Trust

Vengono elencati i sette pilastri fondamentali del modello Zero Trust che rappresentano l'obiettivo ideale:

1. Tutte le fonti di dati e i servizi di elaborazione sono considerati risorse.
2. Tutte le comunicazioni sono protette, indipendentemente dalla posizione della rete.
3. L'accesso a singole risorse aziendali viene concesso in base alla sessione.
4. L'accesso alle risorse è determinato da criteri dinamici, tra cui lo stato osservabile dell'identità del client, dell'applicazione/servizio e dell'asset richiedente, e può includere altri attributi comportamentali e ambientali.
5. L'azienda monitora e misura l'integrità e il livello di sicurezza di tutti gli asset posseduti e associati.
6. Tutte le autenticazioni e le autorizzazioni delle risorse sono dinamiche e applicate rigorosamente prima che l'accesso sia consentito.
7. L'azienda raccoglie quante più informazioni possibili sullo stato attuale degli asset, dell'infrastruttura di rete e delle comunicazioni e le utilizza per migliorare il proprio livello di sicurezza.

1.4.2 Architettura Zero Trust

L'architettura Zero Trust [4] è caratterizzata da una serie di componenti interconnesse che collaborano per gestire e controllare l'accesso a risorse interne della rete privata. Nella figura 1.1 vengono illustrate le interazioni di base tra questi componenti. Si può notare come gli elementi concettuali inerenti la ZTA(*Zero Trust Architecture*) si trovino su un piano di controllo che risulta separato dal piano dati dove avviene invece lo scambio dei dati delle applicazioni.

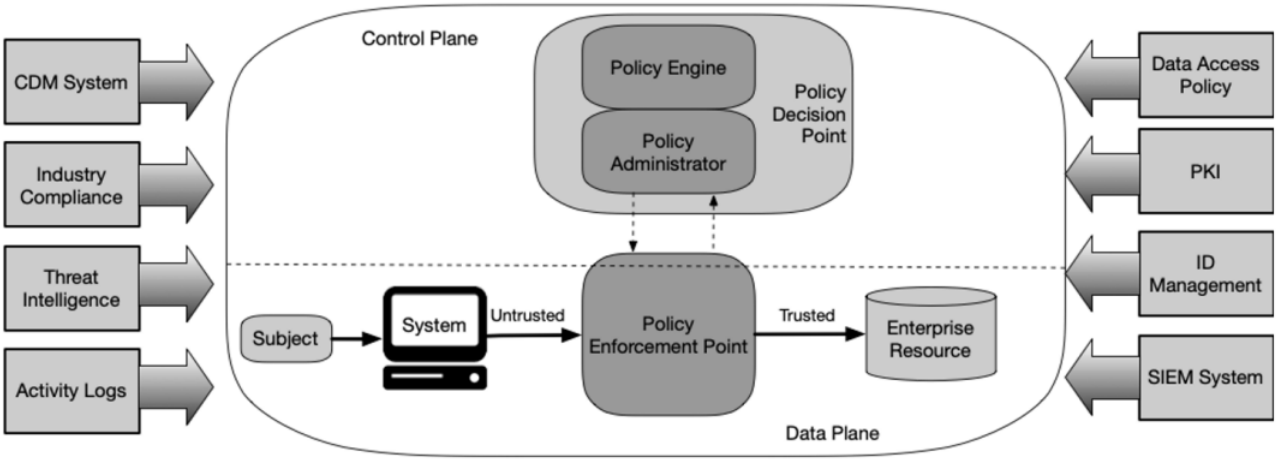


Figura 1.1: Componenti dell'architettura Zero Trust [3]

Di seguito vengono descritte le componenti che permettono il funzionamento dell'architettura.

- ***Policy Decision Point***: componente che funge da entità decisionale, delegata alla valutazione delle richieste di accesso alle risorse e decide se approvare o rifiutare in base alle politiche applicate. Come si può osservare dalla figura 1.1 viene diviso in due componenti logici:
  - ***Policy Engine***: componente incaricato della decisione. Determina se una richiesta di accesso può essere autorizzata o meno. Si basa sulle policy definite dall'organizzazione insieme a input esterni come informazioni provenienti da sistemi che monitorano lo stato dell'ambiente e la presenza di minacce. Una volta conclusa la valutazione, viene prodotta la decisione che può essere di autorizzazione, di rifiuto o anche della revoca di accesso precedentemente concesso.
  - ***Policy Administrator***: è responsabile di trasformare la decisione presa dal PE in un'azione concreta. Nel caso di richiesta autorizzata si occupa di gestire la comunicazione con la risorsa richiesta. Al contrario, nel caso di accesso negato, interrompe la comunicazione.
- ***Policy Enforcement Point***: punto di applicazione delle policy, attraverso il quale viene controllata la comunicazione tra soggetto e risorsa. È il primo componente che entra in gioco dell'architettura che intercetta la richiesta dell'utente impedendo che possa raggiungere direttamente la risorsa senza essere sottoposta a controllo. Si occupa quindi di instaurare la connessione autorizzata, del suo monitoraggio e della possibile terminazione.

Ad assistere il Policy Engine nell'assunzione delle decisioni di accesso ci sono varie sorgenti esterne che forniscono in input informazioni da diverse fonti, ottenendo una visione più completa dello stato dell'ambiente. Tra queste rientrano sistemi di gestione delle identità, informazioni relative ai dispositivi, dati prodotti dai sistemi di monitoraggio e informazioni su vulnerabilità e minacce, oltre alle politiche definite dall'organizzazione stessa.

1.5 Obiettivo di progetto

Individuare tempestivamente attività associabili a possibili minacce e attacchi in corso è fondamentale in un contesto di cybersecurity. I dati prodotti dai sistemi informatici relativi alle operazioni effettuate, agli accessi, alle richieste verso applicazioni e alle risposte generate dai servizi possono essere analizzati per individuare comportamenti che si discostano da quelli ritenuti normali.
Il semplice rilevamento dell'attività sospetta però non sempre basta, è necessario stabilire anche il comportamento da adottare. Una risposta troppo permissiva potrebbe lasciare proseguire un'attività dannosa, mentre una risposta eccessivamente restrittiva potrebbe impedire l'accesso a utenti legittimi. Oltre al rilevamento diventa quindi estremamente importante definire delle risposte che siano proporzionate al livello di rischio osservato.
In un'architettura Zero Trust non è quindi sufficiente decidere semplicemente se un accesso è consentito o meno, ma può essere utile valutare dinamicamente il rischio associato a una richiesta e adottare di conseguenza una risposta adeguata. La valutazione di una richiesta può richiedere l'analisi di molte informazioni relative in primis all'utente, alla sessione e al comportamento osservato. Questo rende interessante l'utilizzo di tecniche automatiche in grado di elaborare grandi quantità di eventi e individuare possibili situazioni di rischio.

L'obiettivo della tesi risulta quindi quello di progettare e sviluppare un'architettura di difesa capace di analizzare sequenze temporali di eventi di sicurezza, stimare la probabilità che il comportamento osservato sia associato ad un attacco e utilizzare questa informazione insieme ad altre caratteristiche contestuali, per selezionare automaticamente una risposta di sicurezza.
Nello specifico l'elaborato consiste in due fasi distinte che vengono considerate come parti successive dello stesso processo. La prima parte è dedicata all'analisi delle sequenze di eventi di log e al rilevamento di comportamenti sospetti mediante un modello di *Deep Learning*, un modello ***Bidirectional Long Short-Term Memory (BiLSTM)***, addestrato su eventi passati, che svolge il ruolo di classificatore, restituendo una probabilità associata alla presenza di un attacco.
La seconda parte invece relativa all'utilizzo di tale informazione insieme ad un contesto più ampio, all'interno di un agente di *Reinforcement Learning* basato su ***Deep Q-Network (DQN)***, che svolge il ruolo di *decision point*, per la selezione dell'azione di mitigazione ritenuta più appropriata tra diverse strategie di risposta.

Il seguente lavoro non si pone l'obiettivo di introdurre un nuovo algoritmo di Deep Learning o di Reinforcement Learning, ma di studiare e realizzare un'integrazione tra queste due tecniche all'interno di una stessa pipeline, orientata alla difesa contro eventi malevoli. La soluzione è così analizzata attraverso una valutazione sperimentale volta a verificare il comportamento dei due componenti, in particolare del contributo della fase di risposta rispetto ai risultati del rilevatore.

Capitolo 2

Stato dell'arte

In questo capitolo viene presentato lo stato attuale dell'arte relativo all'impiego di tecniche di Machine Learning e Deep Learning nell'ambito della cybersecurity e si esploreranno le principali tecnologie utilizzate alla base del presente lavoro.

2.1 Machine Learning nella cybersecurity

Con la diffusione dell'intelligenza artificiale è cambiato drasticamente lo scenario riguardante la difesa da attacchi informatici e, parallelamente, anche il modo in cui vengono concepite e affrontate le minacce. Gli attaccanti moderni sfruttano questa evoluzione utilizzando strumenti di intelligenza artificiale per progettare ed eseguire nuovi attacchi, con il conseguente aumento sia in termini di frequenza, che di complessità e diversificazione. Questo comporta un notevole aumento nella difficoltà di individuazione, prevenzione e mitigazione delle minacce da parte dei sistemi tradizionali. Da un lato, grazie al modello Zero Trust si ha una sicurezza efficace per via del principio fondamentale di negare la fiducia implicita, ma dall'altro bisogna restare al passo con i rapidi cambiamenti e le continue modifiche nelle minacce dovute all'AI affiancando strumenti in grado di analizzare una quantità crescente di informazioni e di adattarsi ai comportamenti osservati.
L'utilizzo dell'intelligenza artificiale e dell'apprendimento automatico (*Machine learning*) possono proprio fornire supporto in questa direzione, nel garantire maggiore sicurezza e nel rendere le architetture di sicurezza Zero Trust molto più robuste e resistenti grazie a modelli in grado di apprendere e rispondere autonomamente e in tempo reale. Un punto interessante è l'analisi comportamentale in grado di analizzare l'attività degli utenti e della rete per determinare comportamenti di riferimento. I sistemi monitorano continuamente le attività, apprendendo i modelli comportamentali appena citati con l'obiettivo di rispondere tempestivamente anticipando gli attacchi, con decisioni immediate che si adattano al contesto.

Per analizzare grandi quantità di eventi e individuare comportamenti che possono essere associati a una minaccia il Machine Learning svolge il ruolo fondamentale di permettere ad un sistema di apprendere da informazioni dei dati senza definire esplicitamente le regole necessarie al raggiungimento della decisione.

L'effettiva applicabilità di tecniche di Machine Learning e Deep Learning all'interno di architetture Zero Trust è stata analizzata anche in studi recenti. *Haider et al.* [5] hanno confrontato diversi algoritmi di apprendimento automatico e profondo per il rilevamento di attività malevole in ambito Zero Trust. Tra i modelli considerati nello studio erano presenti sia approcci di Machine Learning, tra cui *Naïve Bayes*, *Logistic Regression*, *Decision Tree* e *Random Forest*, sia modelli di Deep Learning, tra cui *MLP*, *CNN*, *GRU*, *LSTM* e *Bi-LSTM*.
I risultati ottenuti mostrano una differenza nelle prestazioni tra i diversi approcci considerati. I modelli di deep learning raggiungono costantemente valori superiori al 99% in tutte le metriche analizzate, mentre i modelli più tradizionali presentano in generale risultati inferiori, indicando una minore affidabilità nel rilevamento delle minacce. Lo studio considera inoltre la Bi-LSTM, evidenziando come le architetture ricorrenti possano essere utilizzate per sfruttare le relazioni presenti nei dati sequenziali.
Questi risultati evidenziano il potenziale delle tecniche di apprendimento automatico come componente del livello di rilevamento di un'architettura Zero Trust, nel quale l'analisi continua delle attività può affiancare i tradizionali meccanismi di autenticazione e autorizzazione. Allo stesso tempo, valori di accuratezza così elevati devono essere interpretati considerando le caratteristiche del dataset e le modalità con cui vengono effettuate le fasi di addestramento e valutazione. La qualità e la rappresentatività dei dati rimangono infatti elementi determinanti per valutare la reale capacità di generalizzazione di un modello in scenari differenti da quelli osservati durante l'addestramento.

2.1.1 Sfide e limitazioni

Non si discute sul fatto che l'impiego del Machine Learning possa fornire un supporto significativo ai sistemi di sicurezza, tuttavia, la sua integrazione all'interno di un'architettura Zero Trust introduce anche diverse problematiche da non sottovalutare. L'utilizzo di modelli di apprendimento per analizzare senza sosta le attività e supportare le decisioni di sicurezza richiede infatti di considerare vari aspetti, da quelli legati ai dati, alle prestazioni, all'affidabilità delle predizioni e infine alla capacità del sistema di adattarsi a scenari differenti. Tra le principali:

- **Privacy**: L'analisi del comportamento di utenti, dispositivi e reti richiede l'utilizzo di informazioni che possono contenere dati sensibili. La raccolta ed elaborazione di queste devono quindi tenere conto dei requisiti relativi alla protezione dei dati e alla privacy del GDPR. Una soluzione potrebbe essere l'apprendimento federato che permette l'addestramento attraverso dispositivi centralizzati che mantengono i dati senza effettivamente uno scambio degli stessi.
- **Disponibilità e qualità dei dati**: le prestazioni dei modelli dipendono dalla disponibilità di dati sufficientemente numerosi, completi e rappresentativi per poter apprendere caratteristiche rappresentative del comportamento. Dati incompleti, rumorosi o non aggiornati possono compromettere la capacità del modello di riconoscere correttamente i comportamenti osservati.
- **Costo computazionale**: l'analisi continua degli eventi e la valutazione dinamica real-time delle richieste richiedono risorse computazionali significative, soprattutto in ambienti caratterizzati da un elevato numero di eventi.
- **Falsi positivi/negativi**: uno sbilanciamento dei dati in funzione di eventi normali può provocare una classificazione errata e quindi la generazione di falsi positivi. Questo si verifica quando attività legittime vengono classificate come sospette e può generare interventi di sicurezza non necessari. Al contrario, un falso negativo si verifica quando un'attività malevola non viene correttamente rilevata.
- **Generalizzazione**: un modello addestrato su uno specifico insieme di dati può incontrare difficoltà nell'analizzare scenari differenti da quelli osservati durante l'addestramento. Cambiamenti nei comportamenti degli utenti, nell'ambiente o nelle modalità di attacco possono quindi influenzare le prestazioni del sistema.

2.1.2 Apprendimento supervisionato e classificazione

Il Machine Learning [6] è una parte dell'intelligenza artificiale che comprende varie tecniche e algoritmi per apprendere modelli dai dati di addestramento, senza specifiche regole, e fare previsioni su dati mai visti. Consente il riconoscimento di schemi, il miglioramento delle prestazioni e l'automazione delle attività basate sui dati. I paradigmi di apprendimento nel ML definiscono il modo in cui un algoritmo apprende dai dati, in base alla natura degli stessi e al loro obiettivo di addestramento. I principali sono :

- **apprendimento supervisionato**: addestra un modello per prevedere l'output corretto per un determinato input. Il modello viene addestrato su un set di dati di addestramento(*training set*) per cui la risposta è già nota. Queste istanze di dati sono etichettate, ovvero hanno una label, che rappresenta proprio il valore che il modello deve imparare a prevedere. Durante l'addestramento il modello cerca di imparare una relazione tra le varie caratteristiche dei dati e le relative etichette, in modo da poter successivamente fare previsioni sui dati nuovi non ancora utilizzati(*test set*).

- **apprendimento non supervisionato**: addestra un modello per individuare strutture, schemi, dipendenze e correlazioni nei dati. La differenza con l'apprendimento supervisionato risiede nel fatto che in questo scenario il modello analizza dati non etichettati, quindi senza una verità di base esterna con cui poter confrontare i suoi output.

- **apprendimento per rinforzo(*reinforcement learning*)**: addestra un modello per imparare a prendere decisioni autonomamente, interagendo con un ambiente tramite un susseguirsi di tentativi ed errori. Gli scenari non implicano una sola verità, ma si ha l'esistenza di azioni buone e cattive, il modello riceverà ricompense o penalità in base alle azioni compiute.

In questo lavoro sono stati utilizzati due dei paradigmi di apprendimento sopra citati, l'apprendimento supervisionato e l'apprendimento per rinforzo.
Nel caso specifico dell'apprendimento supervisionato, il compito per cui è stato utilizzato è la **classificazione**. L'obiettivo della classificazione è quello di assegnare ogni osservazione ad una o più classi definite in partenza. Questa prevede infatti valori discreti come la categoria o la classe a cui appartiene un punto; nella classificazione binaria vengono considerate solamente due possibili classi, in questo caso specifico una corrispondente al comportamento legittimo ed una al comportamento sospetto. Il modello tramite l'addestramento riuscirà ad apprendere come distinguere le due categorie sulla base dei dati a disposizione.

2.1.3 Feature engineering

L'efficacia del processo di classificazione dipende in modo significativo dalla qualità dei dati a disposizione utilizzati durante la fase di addestramento. I log grezzi possono contenere informazioni eterogenee, non rappresentative delle vere informazioni rilevanti e magari non utilizzabili dal modello. La **feature engineering** è proprio la fase iniziale dove si trasformano i dati a disposizione in una rappresentazione costituita da caratteristiche rilevanti per lo specifico problema.
Questa fase inizia dapprima con la selezione di quelle che possono essere le informazioni d'interesse, e solo successivamente nella loro trasformazione ove necessario. Le principali problematiche possono prevedere una natura categorica dei dati. Per caratteristiche numeriche, che rappresentano valori quantitativi come numero richieste nel tempo o il tempo trascorso tra due eventi, non si hanno problemi. Nel caso di attributi categorici invece, come il tipo di evento, il ruolo utente o la risorsa interessata, si rende necessaria una trasformazione in formato numerico per poter essere utilizzate dal modello.
Ulteriore operazione potrebbe essere quella di ricavare nuove caratteristiche dai dati disponibili, che permettono di rappresentare meglio degli aspetti utili del comportamento osservato. In questo modo le informazioni dei singoli eventi possono essere arricchite con nuove caratteristiche che ne descrivono il contesto. La selezione delle feature di interesse deve tenere inoltre in considerazione l'effettiva utilità, feature poco informative potrebbero solo peggiorare la classificazione introducendo rumore, mentre di contro feature eccessivamente correlate potrebbero portare il modello ad ottenere risultati elevati senza una reale capacità di generalizzare.

2.1.4 Dati sequenziali e reti ricorrenti

Nella maggior parte dei casi di classificazione, ci si trova di fronte ad eventi indipendenti che non hanno un legame gli uni con gli altri. Questa assunzione non risulta però valida in contesti dove il significato di un'osservazione può dipendere dalle informazioni delle osservazioni precedenti. Si parla in questo caso di **dati sequenziali**, ovvero dati che sono caratterizzati da una relazione temporale, in cui l'ordine degli elementi è fondamentale per comprenderne il significato, organizzati in un ordine preciso e consecutivo in cui ogni elemento è legato sia al precedente che al successivo. I log prodotti dai sistemi informatici che verranno utilizzati in questo lavoro rappresentano un esempio di dati sequenziali. Un log isolato dice pochissimo di per se, mentre la successione cronologica degli eventi può rivelare cosa sta realmente succedendo. Un banale esempio relativo al processo di autenticazione riguarda un log che rappresenta un "*Accesso eseguito correttamente*"; preso singolarmente questo log può risultare normale, ma se i tre log precedenti negli stessi istanti temporali dicessero "*Password errata*", questo diventerebbe il campanello di allarme di un attacco informatico. Ogni evento quindi è associato ad un determinato istante temporale, definito *timestamp*, formando una catena di record che non può essere modificata senza alterare la sequenza.

Le **Reti Neurali Ricorrenti** [7], *Recurrent Neural Networks(RNN)*, sono una tipologia di rete neurale progettata specificamente per elaborare dati sequenziali e quindi perfetta per analizzare log di sicurezza.
Una classica rete neurale *feed-forward* è la più semplice delle reti neurali artificiali in cui le informazioni si muovono in una singola direzione ovvero dall'ingresso verso l'uscita. A differenza delle reti neurali che appunto elaborano gli ingressi indipendentemente, mappando un singolo input su un singolo output, le RNN mappano una sequenza di input su un output, lavorando in un loop ricorrente in cui l'output di un passaggio viene utilizzato come input per il calcolo del passaggio successivo. Nella figura 2.1 possiamo notare la differenza nello schema di entrambe le tipologie di rete. Le RNN introducono il concetto fondamentale di "memoria" che permette di "ricordare" informazioni provenienti dagli elementi precedenti della sequenza per influenzare l'input e l'output correnti. Questa memoria interna, chiamata *stato nascosto* consente alle RNN di comprendere il contesto e l'ordine. L'informazione viene quindi trasferita progressivamente lungo la sequenza permettendo alla rete di tenere conto del contesto temporale.

Le semplici RNN hanno però un limite, presentano difficoltà ad apprendere dipendenze a lungo termine, cioè tendono a dimenticare le informazioni lette troppo tempo prima da eventi molto distanti nella sequenza. La propagazione delle informazioni attraverso molti passi temporali soffre del problema della scomparsa del gradiente [8].
Il gradiente è il valore matematico che indica come modificare i pesi della rete per ridurre gli errori durante l'addestramento. Per correggere gli errori le reti utilizzano un metodo chiamato *Backpropagation Through Time (BPTT)*, l'algoritmo in pratica ripercorre la sequenza temporale per aggiornare i pesi come se tornasse indietro nel tempo, e l'errore viene moltiplicato per gli stessi pesi ad ogni passaggio. Il gradiente rappresenta la direzione e l'intensità del cambiamento utile per migliorare la rete. Il problema della scomparsa del gradiente si verifica quando questi gradienti calcolati diventano molto piccoli, tendenti a zero, e l'aggiornamento dei pesi nella fase di training è quasi inesistente e quindi la rete non apprende. Questo problema è particolarmente rilevante quando per interpretare un evento è necessario considerare informazioni che si trovano a una distanza significativa nella sequenza.

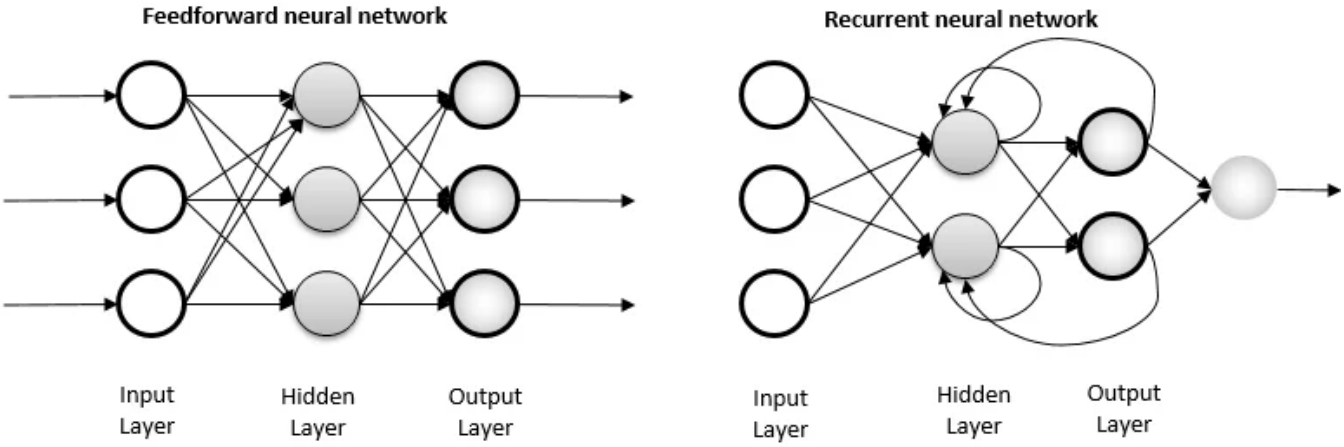


Figura 2.1: Differenza negli schemi di architettura delle reti neurali

2.2 LSTM

Per superare i limiti delle RNN tradizionali e il problema della scomparsa del gradiente, nel 1997 due scienziati informatici, *Sepp Hochreiter* e *Jürgen Schmidhuber*, pubblicano un articolo [9] dal titolo "*Long Short-Term Memory*" dove sviluppano i concetti fondamentali della nuova architettura che saranno alla base della struttura comunemente usata oggi. Nasce così la **LSTM**, una specifica rete neurale ricorrente progettata sempre per elaborare dati sequenziali e inoltre ricordare informazioni a lungo termine. Permettono quindi di mantenere le informazioni rilevanti per un periodo di tempo maggiore.
L'elemento caratteristico che fa la differenza nelle LSTM è la cella di memoria, attraverso la quale si mantengono e si aggiornano le informazioni durante l'elaborazione della sequenza. Queste celle di memoria lavorano insieme a delle porte (gate) per controllare il flusso delle informazioni all'interno della rete per decidere quali informazioni salvare, aggiornare o dimenticare. La cella di memoria può essere vista come un nastro trasportatore che attraversa l'intera catena di moduli, i gate intervengono su di essa aggiungendo o rimuovendo dati senza che il segnale venga perso. I gate della LSTM sono tre:

- **Forget gate**(Porta di oblio): risponde alla domanda "Cosa dobbiamo dimenticare?". Questo gate guarda l'input attuale (x_t) e lo stato nascosto precedente ($h_t - 1$) e decide quali informazioni nello stato della cella precedente scartare e quali mantenere, quindi quanta parte della vecchia memoria eliminare. Assume valori compresi tra 0 e 1, con valori arrotondati allo zero implica la cancellazione completa di quel ricordo, valori prossimi all'uno invece indicano la conservazione del ricordo.

- **Input gate**(Porta di input): risponde alla domanda "Cosa dobbiamo aggiungere?". Controlla l'inserimento di nuove informazioni nello stato corrente della cella. Come nella porta dell'oblio vale il sistema compreso tra 0 e 1, con un valore di zero blocca i dati mentre con un valore positivo li lascia passare. Una parte aggiuntiva è costituita da una *tangente iperbolica (tanh)* che crea un vettore di nuovi valori candidati che potrebbero essere aggiunti alla memoria. Successivamente i due risultati vengono moltiplicati e si sommano al nastro trasportatore aggiornando (c_t), eliminando informazioni irrilevanti e aggiungendone di nuove rilevanti.
- **Output gate**(Porta di output): risponde alla domanda "Cosa dobbiamo mostrare all'esterno?". Controlla quali informazioni nello stato corrente della cella emettere, quindi quale sarà il prossimo stato nascosto (h_t). La classica funzione *sigmoide* decide quali parti della cella di memoria vogliamo mostrare, lo stato della cella (c_t) viene fatto passare attraverso la *tanh* e infine i due elementi vengono moltiplicati per produrre l'output finale (h_t).

Rispetto alle classiche RNN tradizionali, la presenza della memoria e dei gate consente di mantenere le informazioni rilevanti in modo più efficace anche quando queste sono state osservate diversi passi temporali precedenti. Questa caratteristica rende le LSTM adatte a problemi nei quali il contesto temporale assume un ruolo significativo. Nel rilevamento di comportamenti sospetti e malevoli una rete LSTM può essere sfruttata per analizzare una successione di eventi e considerare il comportamento precedente nella valutazione degli eventi successivi. L'aggiunta delle informazioni delle attività già osservate può infatti contribuire a distinguere un singolo evento apparentemente normale, che inserito però in una determinata sequenza, potrebbe assumere un significato diametralmente opposto.

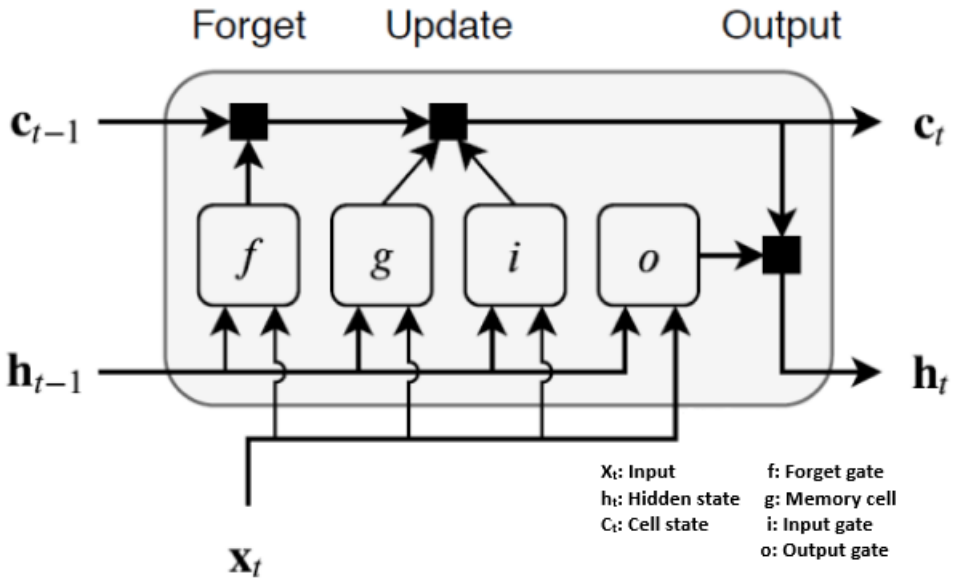


Figura 2.2: Flusso dati dei gate lstm

2.2.1 Problematiche delle LSTM

Le LSTM sono state sviluppate per migliorare la capacità delle reti neurali ricorrenti a gestire le dipendenze a lungo termine, con l'aggiunta della memoria, ciononostante presentano anch'esse alcune problematiche.

- Una prima problematica è legata al costo computazionale. La cella LSTM deve effettuare diverse operazioni di aggiornamento dello stato della memoria e dello stato nascosto ad ogni passaggio temporale. L'elaborazione di sequenze lunghe può quindi richiedere un numero elevato operazioni e un aumento delle risorse computazionali.

- Un'ulteriore aspetto riguarda la difficoltà di addestramento. Anche se il funzionamento delle LSTM riduca in modo significativo il problema della scomparsa del gradiente, non lo elimina completamente. La propagazione delle informazioni attraverso sequenze veramente molto lunghe può comunque rendere complesso l'apprendimento delle dipendenze temporali.

- Nel caso specifico di analisi di eventi di sicurezza, un'ulteriore limitazione è legata alla direzione con cui la sequenza viene elaborata. Una LSTM tradizionale infatti procede lungo la sequenza in un'unica direzione, da sinistra a destra, utilizzando le informazioni disponibili fino al passo temporale corrente. In determinati problemi però anche il contesto fornito dagli elementi successivi della sequenza può risultare utile per ottenere una rappresentazione più completa dell'attività osservata.

2.2.2 Bidirectional LSTM

Proprio per superare la limitazione legata all'elaborazione della sequenza in una sola direzione, sono state introdotte le ***Bidirectional Long Short-Term Memory (BiLSTM)***. Una BiLSTM è un tipo avanzato di architettura di RNN, un'estensione delle LSTM, progettata per comprendere meglio i dati sequenziali. Permette di elaborare una sequenza in entrambe le direzioni, sia in avanti che all'indietro. Mentre la classica LSTM analizza la sequenza procedendo dal primo elemento verso l'ultimo, una BiLSTM utilizza due componenti LSTM distinte, due reti separate in parallelo, una elabora gli elementi nell'ordine originale dall'inizio alla fine, mentre l'altra procede in direzione inversa dalla fine all'inizio. Le due componenti producono quindi due rappresentazioni della stessa sequenza, una ottenuta considerando il contesto precedente e l'altra considerando il contesto futuro. Le informazioni ottenute vengono successivamente combinate per costruire la rappresentazione utilizzata dalla rete ad ogni passo temporale. In questo modo, per ogni elemento della sequenza, il modello può disporre di informazioni provenienti da entrambe le direzioni, ottenendo una comprensione più completa della sequenza. Nella figura 2.3 è possibile vedere l'architettura appena descritta.

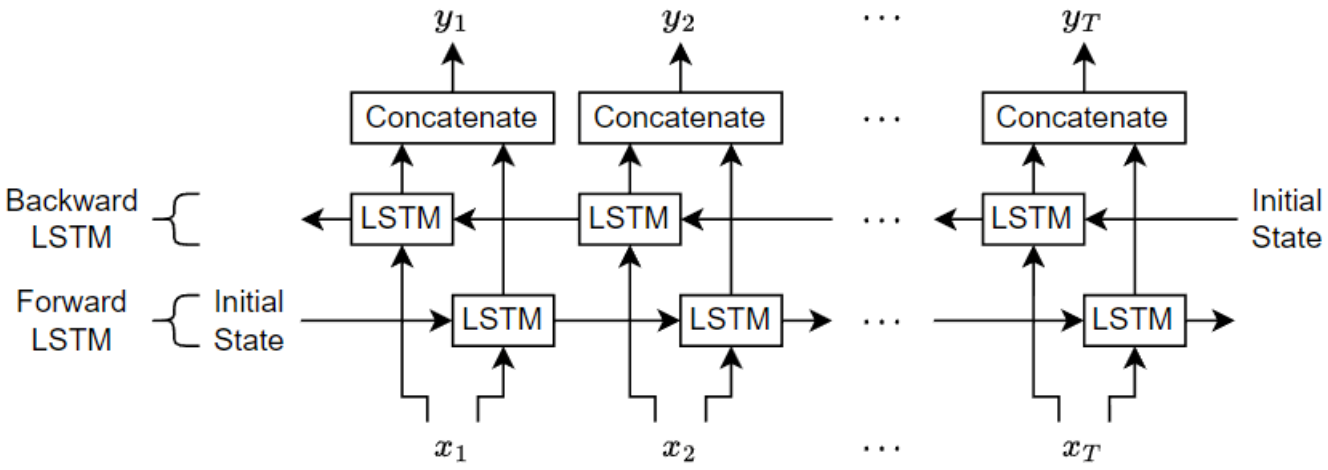


Figura 2.3: Architettura BiLSTM con più fasi temporali

Dal punto di vista architetturale una BiLSTM non costituisce quindi una nuova tipologia di cella rispetto alla LSTM, ma come già citato, combina semplicemente due reti che operano in direzioni opposte. Il risultato può essere utilizzato come input per gli strati successivi della rete, permettendo di costruire architetture più profonde e di effettuare successivamente operazioni di classificazione. Risulta particolarmente utile in problemi in cui il significato di un elemento dipende dal contesto complessivo della sequenza e in lavori dove i dati sequenziali possono variare in lunghezza. Un aspetto importante riguarda tuttavia il contesto di utilizzo di una rete bidirezionale; la possibilità di utilizzare informazioni successive all'interno della sequenza rende questa architettura particolarmente adatta a scenari nei quali l'intera sequenza è disponibile al momento dell'analisi.
In un contesto di cybersicurezza risultano efficaci nella classificazione di attacchi informatici come intrusioni di rete, DDoS, phishing e malware grazie proprio alla bidirezionalità, catturando il contesto completo delle minacce sequenziali.

Il vantaggio dell'elaborazione bidirezionale è stato analizzato già nel lavoro di *Schuster* e *Paliwal* [10], nel quale viene introdotta l'architettura ***Bidirectional Recurrent Neural Network (BRNN)***. Attraverso esperimenti di regressione e classificazione su dati artificiali, si dimostra che l'utilizzo simultaneo delle informazioni provenienti dalle due direzioni temporali può migliorare le prestazioni rispetto alle corrispondenti architetture unidirezionali. Un risultato analogo viene osservato nell'esperimento di classificazione dei fonemi sul dataset *TIMIT*: la struttura BRNN raggiunge un tasso di riconoscimento superiore a quello delle singole reti ricorrenti forward e backward considerate nello studio. Il risultato evidenzia come la conoscenza del contesto precedente e successivo fornisce informazioni aggiuntive utili al compito di classificazione di una sequenza.

Per la parte relativa alla classificazione la scelta del modello da utilizzare è ricaduta sulle BiLSTM in quanto, alla luce delle caratteristiche dei dati considerati nel presente lavoro, ritenute adatte all'analisi dei dati sequenziali dove l'ordine e soprattutto il contesto degli eventi assumono un ruolo significativo.
La principale motivazione della scelta della BiLSTM rispetto alla LSTM tradizionale è stata la possibilità di elaborare ogni sequenza in entrambe le direzioni, combinando le informazioni ottenute dalle due componenti ricorrenti e consentendo quindi al modello di costruire una rappresentazione più completa del contesto presente nella sequenza.
L'architettura BiLSTM costituisce quindi la componente di rilevamento della pipeline proposta, con il compito di effettuare una classificazione binaria tra eventi associati ad un comportamento legittimo ed eventi associati ad un comportamento sospetto.

2.3 Reinforcement Learning

Il ***Reinforcement Learning*** **(RL)**, o apprendimento per rinforzo [11], rappresenta il terzo pilastro del Machine Learning. In questa tecnica un agente autonomo impara a prendere le decisioni tramite l'interazione con un ambiente e ricevendo ricompense o penalità in base alle azioni compiute. L'agente imparerà quindi quali azioni eseguire sulla base delle conseguenze delle proprie scelte. I modelli di apprendimento per rinforzo vengono addestrati in modo olistico attraverso tentativi ed errori, ovvero considerando il sistema o l'individuo nella sua totalità, e non come una somma di parti separate, guardando a come tutte le abilità, il contesto e le reazioni interagiscono insieme per raggiungere un obiettivo finale.
Il processo di apprendimento avviene attraverso una sequenza di interazioni. Al posto delle coppie indipendenti di dati input-output classiche dell'apprendimento supervisionato, nel RL si utilizzano tuple di dati stato-azione-ricompensa interdipendenti. Qui l'obiettivo è quello di ottimizzare i parametri per massimizzare la ricompensa. L'agente osserva lo stato corrente dell'ambiente e, sulla base delle informazioni disponibili, seleziona un'azione. L'ambiente reagisce all'azione modificando il proprio stato e fornendo una ricompensa (reward), che rappresenta un'indicazione della qualità della scelta effettuata. L'agente utilizza queste informazioni per modificare progressivamente il proprio comportamento, cercando di imparare una strategia che permetta di ottenere ricompense elevate nel lungo periodo. Se una certa azione ottiene un segnale positivo dall'ambiente, l'agente sarà incoraggiato a ripetere quell'azione in stati futuri simili.

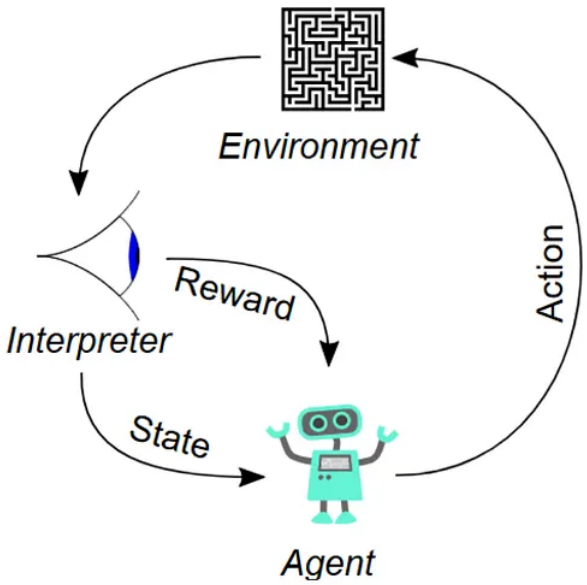


Figura 2.4: Schema funzionamento del Reinforcement Learning

Per modellare matematicamente questa sequenza di interazioni e formalizzare il problema dell'apprendimento per rinforzo, ci si affida al framework dei ***Processi Decisionali di Markov (MDP)***. I MDP sono quindi il modello matematico fondamentale che descrive l'ambiente nel RL. La regola principale alla base è la proprietà di *Markov* che afferma che il futuro dipende solo dallo stato attuale e non dalla storia passata. L'agente non deve ricordare tutti i passi fatti precedentemente, ma osservare la situazione presente e scegliere l'azione migliore. Gli elementi principali che descrivono un problema di RL sono:

- **Ambiente**: il sistema con cui l'agente interagisce e nel quale le azioni producono determinati effetti.
- **Agente**: il componente che prende le decisioni e seleziona le azioni da eseguire.
- **Stato**: la rappresentazione delle condizioni correnti dell'ambiente rilevanti per prendere una decisione. Lo spazio di stato contiene tutte le informazioni disponibili rilevanti per le decisioni che il modello potrebbe prendere e cambia solitamente con ogni azione eseguita dal modello.
- **Azione**: una delle possibili decisioni che l'agente può effettuare in un determinato stato. Lo spazio delle azioni contiene tutte le decisioni che il modello è autorizzato a prendere in un determinato momento.
- **Reward**: valore restituito dall'ambiente dopo l'azione, rappresenta il feedback, positivo o negativo, fornito all'agente come risultato di ogni azione.
- **Policy**: strategia che determina quale azione scegliere in funzione dello stato osservato; guida il comportamento dell'agente.

Esistono tre diversi classi di algoritmi di RL:

- **Metodi basati sul valore**: l'agente apprende una funzione di valore che calcola un punteggio in base alla qualità di ogni stato, quindi sceglie le azioni che portano a stati di valore più elevato.
- **Metodi basati sulla politica**: il modello apprende direttamente una politica, che associa gli stati alle azioni, ottimizzando i parametri della politica per massimizzare le ricompense attese.
- **Metodi actor-critic**: combinano i punti di forza degli approcci basati sul valore e sulla politica. L'actor aggiorna direttamente la politica, mentre il critic valuta le azioni stimando le funzioni di valore.

Un aspetto fondamentale riguarda il compromesso tra *esplorazione* ed *exploitation*. L'esplorazione consiste nel provare azioni differenti per acquisire nuove informazioni relative all'ambiente e verificare l'efficacia di strategie alternative. L'exploitation, invece, consiste nello sfruttare le conoscenze già acquisite, quindi, sulla base dell'esperienza passata, scegliere le azioni che risultano associate a ricompense maggiori. Un agente deve però trovare un equilibrio tra questi due comportamenti, evitando sia di utilizzare esclusivamente strategie già conosciute sia di continuare a esplorare senza sfruttare le informazioni apprese.

In ambito cybersecurity, questo approccio può essere utilizzato per supportare la scelta tra differenti strategie di risposta in funzioni delle condizioni osservate, permettendo al sistema di adattare progressivamente le proprie decisioni all'evoluzione dello scenario.

2.3.1 Q-Learning

Il ***Q-learning*** è un classico algoritmo di reinforcement learning basato sul valore, che permette ad un agente di apprendere quale azione eseguire in funzione dello stato corrente dell'ambiente. Viene introdotto da *Watkins* nel 1989 nella sua tesi di dottorato [12], la quale è riconosciuta come la pietra miliare che ha ufficialmente introdotto il Q-learning. Successivamente formalizzato e analizzato in modo approfondito sempre da *Watkins* e *Dayan* nel 1992 [13], dove vengono analizzate le proprietà dell'algoritmo e viene fornita una dimostrazione della sua convergenza verso i valori ottimali in specifiche condizioni di apprendimento. Introduce un approccio *model-free* attraverso il quale un agente può apprendere una strategia di comportamento senza conoscere a priori il modello di transizione dell'ambiente. Il principio alla base di tutto consiste nell'apprendere una funzione di valore Q, che associa ad ogni coppia stato-azione un valore numerico detto valore Q(***Q-value***). Questo Q-value rappresenta l'utilità attesa dell'esecuzione di quella determinata azione, però non solo considerando la ricompensa ottenibile nell'immediato, ma anche quelle future che derivano come conseguenza dell'azione, che potranno essere ottenute nelle fasi successive.

Durante l'interazione con l'ambiente, l'agente aggiorna progressivamente questi valori sulla base delle esperienze raccolte, avvicinandosi a una strategia in grado di selezionare le azioni più vantaggiose nei diversi stati.
Nel Q-learning tradizionale i valori della funzione Q vengono memorizzati in una specifica tabella chiamata *Q-table*, nella quale ogni possibile combinazione tra stato e azione è associata al relativo Q-value. Durante l'interazione con l'ambiente, l'agente osserva lo stato corrente, seleziona un'azione e riceve una ricompensa, raggiungendo successivamente un nuovo stato. Sulla base di questa esperienza, il valore associato alla coppia stato-azione viene aggiornato considerando sia la ricompensa immediatamente ricevuta sia la migliore ricompensa futura stimata, permettendo quindi di propagare progressivamente l'informazione relativa alle conseguenze delle azioni precedenti.
L'importanza che assume questo meccanismo si manifesta nei problemi caratterizzati da ricompense ritardate (*delayed rewards*). Può capitare in molti scenari che un'azione non produce un effetto immediatamente osservabile, ma che le sue conseguenze si manifestino solo dopo diversi passaggi. Il problema consiste quindi nell'attribuire correttamente la ricompensa finale alle azioni che hanno contribuito a determinarla. Con il Q-learning si risolve questo problema attraverso l'aggiornamento progressivo dei valori Q, permettendo che l'informazione relativa alle ricompense venga propagata verso le azioni precedentemente eseguite.
Il principale risultato dello studio condotto nel 1992 è il teorema di convergenza secondo il quale, in un ambiente Markoviano con rappresentazione discreta dei valori Q, l'algoritmo converge ai valori ottimali delle azioni con probabilità pari a uno, a condizione che tutte le azioni vengano campionate ripetutamente in tutti gli stati.

2.3.2 Problematiche del Q-learning

Nonostante i vantaggi riscontrati, anche in questo caso esistono degli svantaggi e limitazioni da tenere in considerazione. In riferimento al teorema di convergenza, la condizione relativa alla ripetuta esplorazione degli stati e delle azioni evidenzia un aspetto rilevante dell'algoritmo ossia che la convergenza teorica dipende dalle condizioni con cui l'ambiente viene esplorato. In secondo luogo, l'utilizzo di una Q-table presenta un limite significativo nel momento in cui lo spazio degli stati diventa particolarmente ampio. In questo caso la tabella dovrebbe contenere un valore per ogni possibile combinazione tra stato e azione e di conseguenza, all'aumentare della complessità dell'ambiente, il numero di combinazioni da tenere in memoria crescerebbe rapidamente. Quando si hanno spazi degli stati molto grandi la gestione della tabella risulta difficile e la memorizzazione e l'aggiornamento di ogni possibile coppia stato-azione diventano quindi poco pratici.

2.4 DQN

Il ***Deep Q-Network (DQN)*** nasce per superare un limite del Q-learning, il cosiddetto problema della *"maledizione della dimensionalità"*. Il DQN è un algoritmo di apprendimento per rinforzo basato sul Q-learning che però utilizza una rete neurale per approssimare la funzione Q. Viene sostituita la rappresentazione tabellare della funzione Q, la Q-table, con un approssimatore basato su una rete neurale, come si può vedere nella figura 2.5. Questo principio costituisce la base delle DQN, nelle quali i Q-value non vengono quindi più memorizzati direttamente in una tabella, ma vengono stimati da una rete neurale a partire dalla rappresentazione dello stato. La rete riceve quindi lo stato dell'ambiente come ingresso e restituisce i valori Q associati alle diverse azioni disponibili. L'agente può poi selezionare l'azione associata al valore Q più elevato, secondo la strategia di esplorazione adottata.
Nel corso degli anni molti ricercatori hanno provato ad unire reti neurali e apprendimento per rinforzo, con il risultato finale comune di modelli estremamente instabili che tendevano a fallire, divergere. *Sutton* e *Barto* [14] hanno dimostrato che quando si uniscono contemporaneamente tre elementi ("*la triade mortale*"), il sistema diventa matematicamente instabile per sua natura. I tre elementi sono: l'approssimazione della funzione, ovvero usare la rete neurale per stimare i valori al posto della tabella fissa; il bootstrapping, aggiornare le stime correnti basandosi su altre stime, come avviene nel Q-learning; infine la off-policy learning, cioè imparare da dati raccolti seguendo una strategia diversa da quella che si sta ottimizzando. Viene analizzato come l'unione di questi tre fattori provochi quasi sempre la divergenza dei parametri.
Anche Baird nel 1995 [15] dimostrò matematicamente che persino una semplicissima rete neurale lineare con pochissimi parametri, se combinata con il Q-Learning tradizionale e il bootstrapping, poteva divergere all'infinito anche in ambienti elementari con pochi stati. Questo mostrò come il problema non risiede nella complessità della rete, ma nella natura stessa dell'aggiornamento matematico del RL. Ancora tutt'oggi si può dire che il problema non è stato pienamente risolto, ne da un punto di vista teorico ne matematico, ma è stato "addomesticato" da un punto di vista ingegneristico. La divergenza nell'unione di RL e reti neurali accadeva perché i dati raccolti da un agente in movimento sono sequenziali e fortemente correlati tra loro.
Tra il 2013 e il 2015, *Mnih et al.*, ricercatori di *DeepMind*, grazie prima ad uno studio, e successivamente ad un articolo [16], danno vita all'algoritmo DQN. Il paper però conferma e descrive ampiamente l'estrema instabilità del sistema, ma più che risolvere il problema, questo viene aggirato attraverso due "trucchi" ingegneristici esterni alla teoria del RL. Per rendere stabile il processo di apprendimento vengono quindi introdotti due meccanismi, *experience replay* e *target network*.

La ***experience replay*** memorizza le esperienze dell'agente all'interno di una memoria, indicata come *replay buffer*. Ognuna di queste contiene lo stato corrente, l'azione eseguita, la ricompensa ricevuta e lo stato successivo. Successivamente, durante l'addestramento, il modello non utilizza soltanto l'esperienza più recente, ma utilizza campioni estratti casualmente dalle esperienze già memorizzate. Questo permette di riutilizzare le interazioni precedenti riducendo al contempo le correlazioni tra esempi consecutivi, rendendo più stabile l'aggiornamento della rete.
La ***target network*** invece consiste nell'utilizzare una "copia" della rete, una seconda rete neurale generalmente inizializzata con gli stessi parametri della rete principale. Quella principale, detta *online network* viene aggiornata durante l'addestramento, mentre la target network viene mantenuta temporaneamente fissa e aggiornata solo periodicamente copiando i parametri di quella principale. Con questo "trucchetto" i valori utilizzati per costruire il riferimento dell'aggiornamento cambiano meno frequentemente, contribuendo a limitare oscillazioni e instabilità durante l'apprendimento.

L'efficacia di questa combinazione è stata dimostrata sperimentalmente proprio da *Mnih et al.* attraverso una valutazione su 49 giochi Atari 2600. Il DQN ha ottenuto prestazioni superiori ai precedenti metodi di RL considerati in 43 dei 49 giochi, raggiungendo almeno il 75% della prestazione umana in ben 29 giochi.
Nonostante i risultati ottenuti, la principale problematica di una DQN resta la stabilità dell'addestramento, l'utilizzo di una rete neurale per approssimare la funzione Q può produrre oscillazioni o divergenza se le esperienze utilizzate durante l'apprendimento sono fortemente correlate o se i valori target cambiano rapidamente. L'experience replay e la target network sono state progettate proprio per ridurre tali problematiche, ma non eliminano la necessità di una corretta progettazione dell'ambiente e dei parametri di apprendimento.

Un ulteriore aspetto da tenere in considerazione riguarda la dipendenza del comportamento dell'agente dalla funzione di reward. Nel caso di un sistema di cybersecurity, la ricompensa deve rappresentare adeguatamente il compromesso tra efficacia dell'azione di mitigazione e l'impatto sull'attività legittima. Una funzione di reward non adeguatamente progettata potrebbe portare l'agente a privilegiare comportamenti che massimizzano la ricompensa numerica senza necessariamente corrispondere all'obiettivo di sicurezza desiderato. La progettazione dello stato, dello spazio delle azioni e della funzione di ricompensa rappresenta quindi una parte fondamentale del problema.

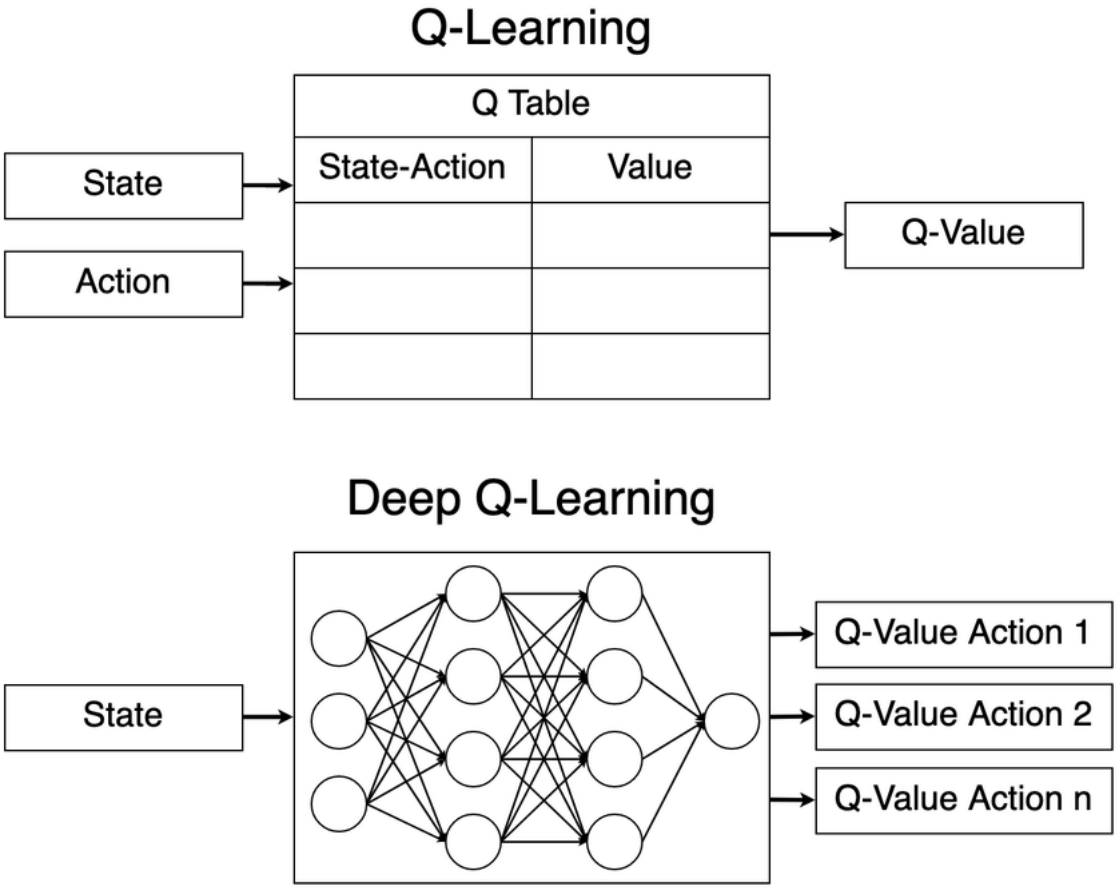


Figura 2.5: Differenza tra Q-learning e DQN. © 2021 IEEE.

Relativamente al presente progetto, per la realizzazione della componente decisionale della pipeline la scelta è ricaduta sulla DQN. La scelta è legata principalmente alla necessità di gestire uno stato dell'ambiente descritto da più caratteristiche relative al comportamento osservato e al contesto della richiesta. L'utilizzo di una Q-table risulterebbe infatti poco adatto alla rappresentazione di uno spazio degli stati di questo tipo, mentre l'approssimazione della funzione Q tramite una rete neurale permette di elaborare direttamente la rappresentazione dello stato e di stimare i valori associati alle diverse azioni di risposta.

La scelta dei due modelli risponde quindi a due esigenze differenti ma complementari. La BiLSTM viene utilizzata per analizzare le sequenze temporali di eventi e classificare il comportamento osservato, mentre la DQN utilizza le informazioni prodotte dal rilevatore e il contesto corrente per apprendere una strategia di risposta. In questo modo, la prima componente si concentra sull'identificazione del possibile attacco, mentre la seconda supporta la selezione della contromisura da applicare.

Capitolo 3

Dataset e strumenti software

In questo capitolo si possono trovare una descrizione e un'analisi dei dati a disposizione e degli strumenti software utilizzati per la realizzazione del progetto. Nella prima parte viene quindi presentato il dataset utilizzato, analizzandone la struttura con le sue principali caratteristiche e informazioni contenute al suo interno, successivamente vengono descritti gli strumenti software e le principali librerie utilizzate per l'implementazione dell'intera architettura.

3.1 Dataset

Il dataset utilizzato nel presente lavoro raccoglie eventi di audit, autenticazione e monitoraggio generati all'interno di un ambiente di rete controllato. La raccolta dei log è stata progettata con l'obiettivo di fornire una rappresentazione sufficientemente dettagliata delle attività svolte all'interno del sistema, includendo sia operazioni applicative ordinarie sia eventi caratterizzati da comportamenti potenzialmente anomali o rilevanti dal punto di vista della sicurezza.
I dati derivano da un'infrastruttura di monitoraggio realizzata nell'ambito di un progetto complementare [17], finalizzato alla costruzione di un ambiente di rete completo nel quale differenti componenti applicative e strumenti di sicurezza producono eventi osservabili e successivamente analizzabili. Tale impostazione consente di ottenere informazioni eterogenee relative agli utenti, ai dispositivi, alle richieste HTTP, alle risorse applicative coinvolte e, più in generale, al contesto nel quale ciascun evento è stato generato.

L'infrastruttura citata è stata realizzata ad-hoc per rappresentare un dataset etichettato per uno scenario Zero Trust. Questa architettura, basata su *Microsoft Hyper-V*, è organizzata in tre aree di sicurezza isolate e separate da un firewall di nuova generazione. Una sonda passiva raccoglie eventi di rete tramite diversi strumenti open source; questi dati confluiscono in *Splunk* insieme ai record di audit prodotti da un'applicazione *Blazor WebAssembly*.

Quest'ultima, grazie a middleware che verificano JWT e sessione lato server, genera eventi più ricchi rispetto alla sola rete, perché combina identità, dispositivo e comportamento. Ogni accesso viene poi arricchito e associato a un campo Suspicious e a una descrizione categorica.
Il traffico legittimo è prodotto da un bot Python che autentica l'utente ed esegue sequenze *CRUD* casuali, con label *Suspicious=false*.
Il traffico malevolo è invece generato tramite un nodo *Kali Linux* dual-homed, dotato di una suite modulare di penetration testing, che copre l'intera kill chain. Questi eventi sono etichettati come *Suspicious=true*.

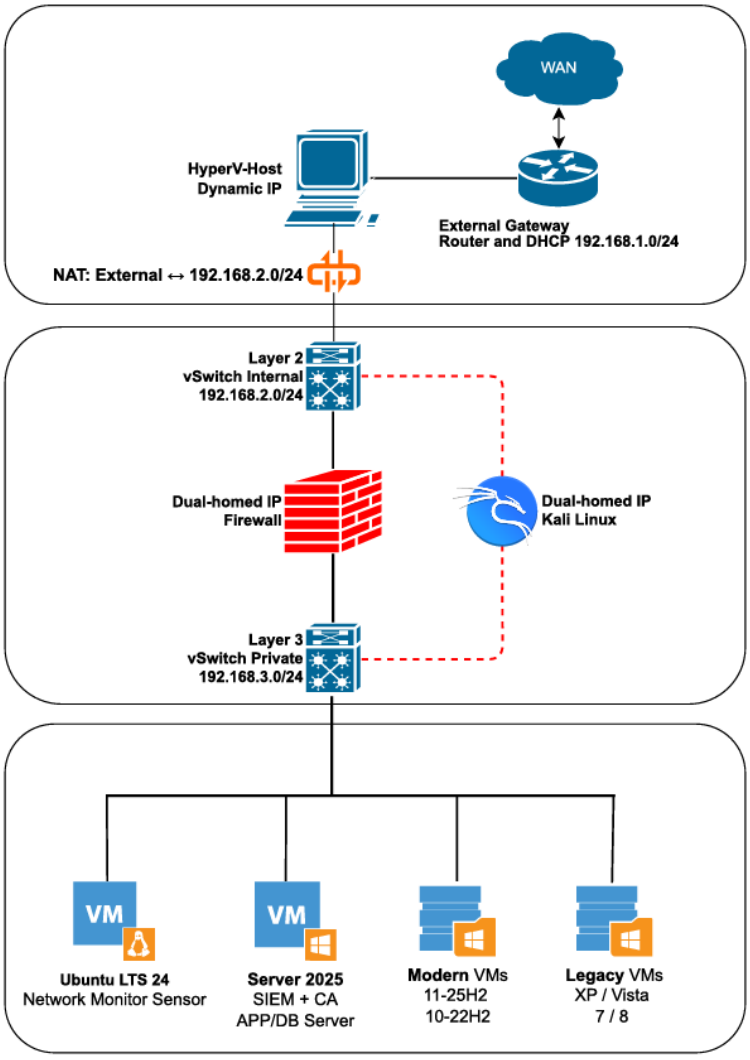


Figura 3.1: Topologia cyber-range Zero Trust a tre livelli

Ciascun elemento del dataset rappresenta un singolo evento e presenta una struttura gerarchica composta da un insieme di attributi generali, comuni alla maggior parte delle sorgenti, e da un campo *message* contenente le informazioni specifiche dell'evento.

Campo	Descrizione
`time`	Timestamp associato all'evento, indicante la data e l'ora in cui l'azione è stata registrata dal sistema.
`host`	Identificativo della macchina o del sistema dal quale è stato generato il log.
`source`	Identificativo dell'applicazione o della componente software responsabile della produzione dell'evento.
`sourcetype`	Categoria dell'evento, utilizzata per distinguere differenti tipologie di log prodotte dalla medesima sorgente.
`message`	Corpo principale dell'evento. Contiene le informazioni specifiche relative all'operazione registrata e può includere strutture annidate differenti in funzione della sorgente e del tipo di evento.

Tabella 3.1: Descrizione dei principali campi del log

La presenza di un campo message strutturato rappresenta un elemento particolarmente importante ai fini dell'analisi; permette infatti di associare all'informazione temporale e alla sorgente dell'evento ulteriori attributi relativi, ad esempio, all'indirizzo IP del client, all'utente coinvolto, alla sessione applicativa, alla richiesta HTTP, alla risorsa interessata e all'eventuale classificazione dell'evento come sospetto.
Considerata poi l'elevata quantità di informazioni disponibili, non tutti gli attributi presenti nei log assumono la stessa rilevanza. Per questo motivo, l'analisi è stata concentrata sui campi caratterizzati da un maggiore contenuto informativo e maggiormente utili ai fini della caratterizzazione del comportamento degli utenti e dell'individuazione di fenomeni anomali.

3.1.1 Blazor:2.0.0.2:ApplicationAudit/AuthAudit

Una parte rilevante del dataset è costituita dagli eventi generati dalle componenti di audit dell'applicazione web, identificate attraverso sorgenti riconducibili ad *ApplicationAudit* e *AuthAudit*.
Tali componenti hanno il compito di registrare le principali operazioni effettuate dagli utenti e le richieste considerate significative dal punto di vista applicativo o della sicurezza. Tra gli eventi osservabili rientrano, ad esempio, operazioni di autenticazione, accesso alle funzionalità dell'applicazione, creazione o modifica di risorse, richieste non valide ed eventi classificati dal sistema come sospetti.

Gli attributi del campo *message* rilevanti per il presente lavoro sono riportati di seguito.

Campo	Descrizione
`Id`	Identificativo dell'evento di audit.
`Description`	Descrizione testuale dell'operazione registrata.
`HashCheck`	Valore hash associato all'evento, utilizzabile per verificare l'integrità delle informazioni registrate.
`IP`	Indirizzo IP del client associato all'evento.
`User`	Identificativo dell'utente che ha eseguito l'operazione, quando disponibile.
`UserAgent`	Stringa User-Agent del browser o del client che ha originato la richiesta.
`EventType`	Tipologia dell'evento applicativo registrato.
`Level`	Livello informativo associato all'evento, ad esempio informazione o warning.
`Suspicious`	Indicatore booleano utilizzato dal sistema per identificare eventi considerati potenzialmente sospetti.
`SuspiciousDescription`	Descrizione delle motivazioni che hanno determinato la classificazione dell'evento come sospetto.
`Route`	Endpoint applicativo coinvolto nella richiesta.
`HTTPStatusCode`	Codice di stato HTTP restituito dal server.
`SessionId`	Identificativo della sessione associata all'utente o alla richiesta.
`ResourceContext`	Informazioni relative alla risorsa coinvolta nell'evento, come database e tabella.
`RequestMetadata`	Metadati della richiesta HTTP, comprendenti metodo, protocollo, URL, endpoint locale e remoto e altre informazioni di contesto.
`Fingerprint`	Insieme di attributi utilizzati per descrivere le caratteristiche del client, quali browser, sistema operativo, risoluzione, memoria, GPU e impostazioni del browser.
`FingerprintHMAC`	Valore HMAC associato al fingerprint del client.
`ClientCertificate`	Certificato client associato alla richiesta, quando presente.
`UserContext`	Informazioni sul contesto applicativo dell'utente, tra cui i ruoli assegnati.

Tabella 3.2: Principali campi degli eventi di audit applicativo

La ricchezza di tali informazioni consente di analizzare un evento non soltanto come una singola richiesta applicativa, ma come parte di un contesto più ampio. È infatti possibile correlare un indirizzo IP con uno o più utenti, ricostruire l'attività di una sessione, osservare la distribuzione temporale delle richieste e valutare il coinvolgimento delle differenti risorse applicative.

Particolare rilevanza assume il campo ***Suspicious***. Esso fornisce una prima classificazione degli eventi e permette di distinguere il traffico considerato normale da quello che ha soddisfatto una o più condizioni di anomalia definite dal sistema di monitoraggio. Questa informazione è stata utilizzata nella successiva analisi esplorativa come riferimento per studiare la distribuzione e la concentrazione degli eventi sospetti.
Anche ***HTTPStatusCode*** risulta significativo. Codici di risposta differenti possono infatti rappresentare tipologie di comportamento molto diverse: una richiesta completata correttamente produce generalmente una risposta appartenente alla classe 2xx, mentre un numero elevato di risposte 4xx può essere associato a richieste non valide, errori di autenticazione o tentativi automatizzati.
Il campo ***UserContext*** permette inoltre di distinguere gli eventi in funzione del ruolo applicativo dell'utente. Nel dataset sono presenti ruoli quali *User* e *Administrator*. Tale distinzione consente di verificare se la distribuzione degli eventi, e in particolare degli eventi sospetti, varia in funzione del livello di privilegio posseduto dall'utente.

3.1.1.1 Informazioni relative alle risorse applicative

Gli eventi di audit possono contenere un oggetto ***ResourceContext***, utilizzato per descrivere la risorsa interessata dall'operazione. Tra le informazioni maggiormente rilevanti sono presenti il database e la tabella coinvolti. Tali attributi consentono di passare da un'analisi esclusivamente orientata al client a una caratterizzazione dell'attività anche dal punto di vista delle risorse accedute. In questo modo è possibile individuare quali componenti del sistema risultino maggiormente coinvolte nelle operazioni registrate e verificare se specifiche risorse presentino una maggiore frequenza di accesso.

3.1.1.2 Informazioni relative all'identità e alle sessioni

Per caratterizzare l'origine degli eventi sono stati utilizzati principalmente gli attributi ***IP***, ***User*** e ***SessionId***. L'indirizzo *IP* rappresenta il punto di osservazione principale per individuare il client responsabile di una determinata attività. Il campo *User*, quando valorizzato, consente invece di associare l'evento a un'identità applicativa.
È importante sottolineare che la relazione tra indirizzi IP e utenti non è necessariamente biunivoca. Uno stesso indirizzo IP può essere associato a utenti differenti, così come uno stesso utente può generare eventi provenienti da indirizzi IP diversi. Per tale motivo, oltre al numero di IP e utenti distinti, sono state analizzate anche le combinazioni uniche IP–utente. La disponibilità di *SessionId* permette infine di aggregare più eventi all'interno della medesima sessione applicativa. Tale informazione è particolarmente utile nell'analisi temporale, poiché consente di calcolare la durata dell'attività associata a ciascuna sessione osservando il tempo trascorso tra il primo e l'ultimo evento registrato.

3.1.1.3 Certificati client e ruoli applicativi

Tra le ulteriori informazioni presenti negli eventi sono stati considerati i certificati client e i ruoli applicativi.
Il campo ***ClientCertificate*** permette di verificare se una determinata richiesta sia stata accompagnata da un certificato client. Nel dataset tale attributo può essere valorizzato oppure risultare nullo. Per la caratterizzazione generale dei dati è stata quindi calcolata la percentuale di eventi nei quali il certificato risulta presente.
Il campo ***UserContext***, invece, contiene informazioni relative all'identità applicativa dell'utente. In particolare, la lista dei ruoli permette di distinguere gli eventi generati da differenti categorie di utenti: *Administrator* e *User*.
La distribuzione dei ruoli è stata analizzata sia in termini assoluti sia rispetto alla classificazione di sicurezza dell'evento, allo scopo di verificare se la percentuale di eventi sospetti presenti differenze significative tra le varie categorie di utenti.

3.1.2 Captive Portal

La tabella 3.3 illustra la struttura dei log generati dal *captive portal*, ossia l'interfaccia web attraverso la quale un utente può ottenere l'accesso alla rete. Tale componente introduce un ulteriore livello di controllo e identificazione, permettendo di raccogliere informazioni relative al client fin dalle prime fasi della connessione e di associare alla sessione dati provenienti sia dal lato server sia dal browser dell'utente.

Campo	Descrizione
`ip`	Indirizzo IP del client che ha avviato una sessione accedendo alla pagina del captive portal.
`ipBehindProxy`	Indirizzo IP del client individuato dietro un eventuale proxy, qualora tale informazione sia disponibile.
`FingerprintHMAC`	Codice HMAC calcolato sui dati di fingerprinting, utilizzato per verificarne l'autenticità e l'integrità.
`userAgent_php`	Stringa User-Agent del client acquisita lato server attraverso PHP.
`userAgent_js`	Stringa User-Agent rilevata direttamente dal browser mediante JavaScript.
`referrer_js`	Indirizzo della pagina di provenienza rilevato lato browser tramite JavaScript; il campo può risultare vuoto qualora l'informazione non sia disponibile.
`SHA256`	Valore hash SHA-256 calcolato sui dati dell'evento, utilizzato per verificare l'integrità delle informazioni registrate.
`EventType`	Tipologia dell'evento registrato. Nel caso considerato assume il valore `captive_portal.opened`, indicando l'apertura della pagina del captive portal da parte del client.

Tabella 3.3: Campi dell'evento del captive portal

3.1.3 p0f

La seguente tabella descrive la struttura dei log prodotti da *p0f*, uno strumento di passive OS fingerprinting utilizzato per individuare e caratterizzare i dispositivi presenti all'interno della rete. A differenza delle tecniche di scansione attiva, p0f analizza il traffico di rete osservato senza instaurare comunicazioni dirette con gli host, consentendo di ricavare informazioni quali il sistema operativo presumibilmente utilizzato, gli indirizzi di rete e gli intervalli temporali di attività dei dispositivi.

Campo	**Descrizione**
`session_id`	Identificativo univoco associato alla sessione di acquisizione effettuata da p0f. Include informazioni temporali relative all'avvio della raccolta dei dati.
`device_count`	Numero complessivo di dispositivi individuati durante la sessione di osservazione.
`<IP>`	Campo associato a ciascun indirizzo IP rilevato. Contiene una struttura con le principali informazioni relative al dispositivo osservato.
`mac`	Indirizzo MAC associato al dispositivo rilevato.
`os`	Sistema operativo stimato da p0f attraverso l'analisi passiva del traffico di rete.
`first_seen`	Timestamp relativo alla prima osservazione del dispositivo durante la sessione di acquisizione.
`last_seen`	Timestamp corrispondente all'ultima osservazione del dispositivo registrata nella sessione.
`hostname`	Nome host associato al dispositivo, qualora sia stato possibile rilevarlo.
`SuspiciousDescription`	Descrizione testuale relativa agli indirizzi IP o ai dispositivi classificati come potenzialmente sospetti.
`devicesHash`	Valore hash calcolato sull'insieme dei dispositivi rilevati, utilizzato per verificare l'integrità dei dati associati alla sessione.
`devicesHMAC`	Codice HMAC calcolato sui dati relativi ai dispositivi, impiegato per verificarne l'autenticità e rilevare eventuali alterazioni.

Tabella 3.4: Campi dell'evento p0f

3.2 Analisi esplorativa del dataset

Prima di procedere con le successive fasi di elaborazione è stata eseguita un'analisi esplorativa del dataset. Tale attività ha avuto una duplice finalità: verificare le caratteristiche generali dei dati disponibili e individuare eventuali pattern significativi dal punto di vista temporale, applicativo e della sicurezza.
Le metriche considerate sono state selezionate cercando di evitare indicatori ridondanti e privilegiando quelli maggiormente interpretabili.

3.2.1 Dimensione e composizione del dataset

La prima fase dell'analisi riguarda la dimensione complessiva del dataset e la ripartizione degli eventi sulla base del campo *Suspicious*.

Record totali	283.399
Record sospetti	140.637 (49,63%)
Record normali	141.153 (49,81%)

Il dataset analizzato contiene complessivamente 283.399 eventi, dei quali:

- 140.637 classificati come sospetti, pari al 49.63% del totale;
- 141.153 classificati come normali, pari al 49.81%;
- 1.609 record hanno *Suspicious* mancante/non interpretabile.

Questa prima distinzione permette di valutare il bilanciamento complessivo del dataset. La percentuale relativa agli eventi sospetti rappresenta infatti un elemento importante sia per comprendere la natura dei dati sia, in una fase successiva, per valutare eventuali problematiche di sbilanciamento tra le classi.

3.2.2 Distribuzione di IP, utenti e sessioni

Successivamente è stata analizzata la cardinalità delle principali entità presenti nei log. Sono stati individuati:

- 19 indirizzi IP distinti;
- 10 utenti distinti;
- 136 sessioni applicative;
- 35 combinazioni uniche tra indirizzo IP e utente.

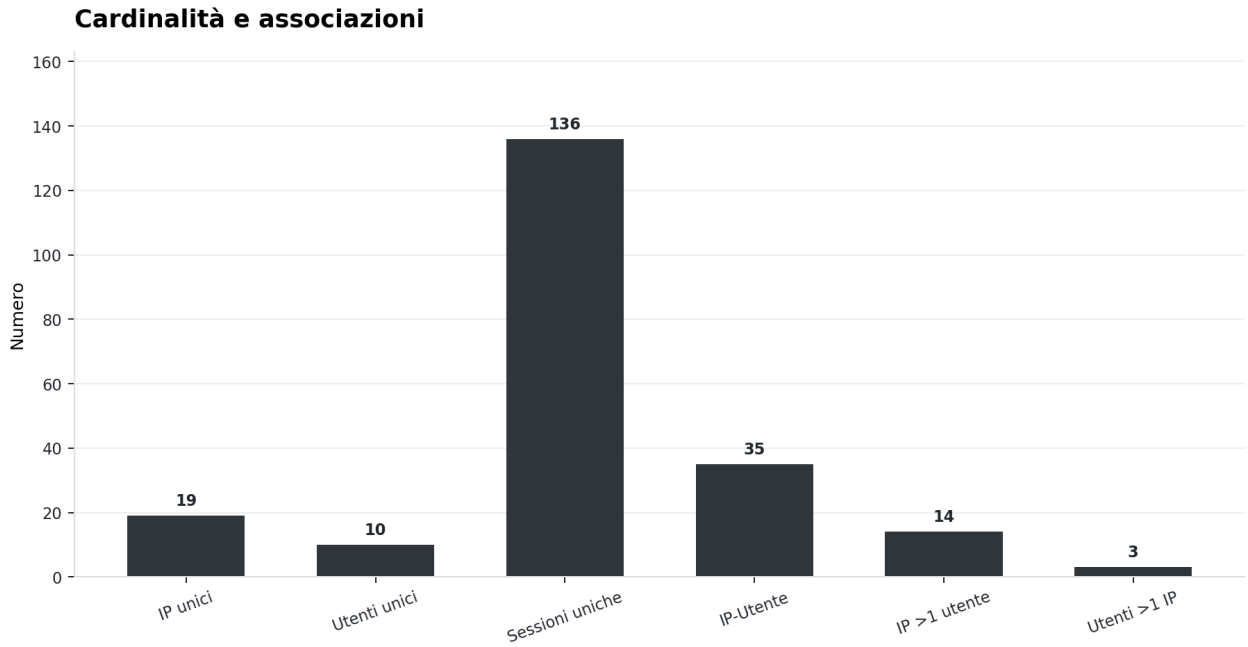


Figura 3.2: Distribuzione IP, utenti e sessioni

Oltre alle cardinalità semplici, è stata analizzata la relazione tra IP e identità applicativa. In particolare, 14 indirizzi IP risultano associati a più di un utente, mentre 3 utenti sono stati osservati da più indirizzi IP differenti.
Queste informazioni consentono di descrivere la complessità delle relazioni presenti nel dataset. La semplice associazione di un utente a un indirizzo IP non può infatti essere considerata sufficiente per identificare univocamente un soggetto, soprattutto in presenza di postazioni condivise, sistemi intermedi, variazioni di rete o accessi effettuati da dispositivi differenti.

3.2.3 Analisi degli eventi sospetti per IP e utente

Per valutare il comportamento delle singole entità non è sufficiente considerare esclusivamente il numero assoluto di eventi sospetti. Un indirizzo IP che produce un numero elevato di eventi potrebbe infatti risultare molto attivo anche in condizioni normali.
Per questo motivo è stato calcolato, per ciascun IP e utente, il tasso di sospetto, definito come il rapporto tra il numero di eventi classificati come sospetti e il numero totale di eventi associati alla medesima entità.

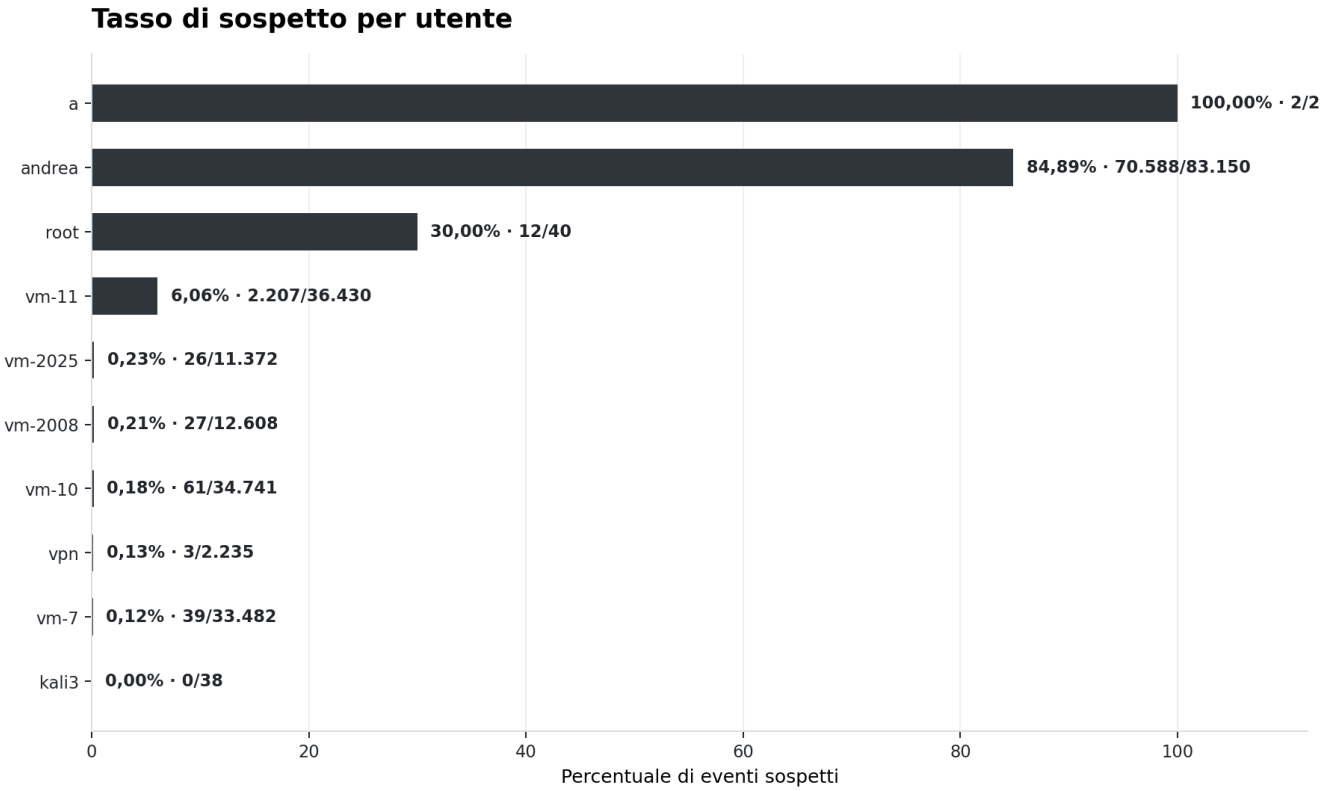


Figura 3.3: Tasso di sospetto per utenti

Tale rappresentazione permette di distinguere soggetti caratterizzati da un elevato volume di attività ma da una bassa incidenza di anomalie da soggetti per i quali la quasi totalità delle operazioni viene classificata come sospetta.

Un'analisi analoga è stata effettuata sui ruoli applicativi, confrontando ad esempio gli eventi associati agli utenti con ruolo *Administrator* con quelli appartenenti alla categoria *User*.
Per la categoria Administrator sono stati rilevati 72.901 eventi sospetti su un totale di 177.122, equivalenti al 41.16%. Per User invece solamente 64 sospetti su un totale di 36.976, ovvero un trascurabile 0.17%.

3.2.4 Concentrazione degli eventi sospetti

Oltre al tasso di sospetto individuale è stata analizzata la concentrazione complessiva degli eventi sospetti.
L'obiettivo è verificare se le anomalie risultino distribuite uniformemente tra indirizzi IP e utenti oppure se siano riconducibili principalmente a un numero ristretto di soggetti.
A tale scopo gli IP sono stati ordinati in base al numero di eventi sospetti generati ed è stata calcolata la quota cumulativa attribuibile ai primi soggetti della distribuzione. La medesima analisi è stata effettuata per gli utenti.
Nel dataset analizzato, i primi 5 indirizzi IP rappresentano complessivamente il 53,14% degli eventi sospetti. Analogamente, il primo utente rappresenta da solo il 96,74% del totale.

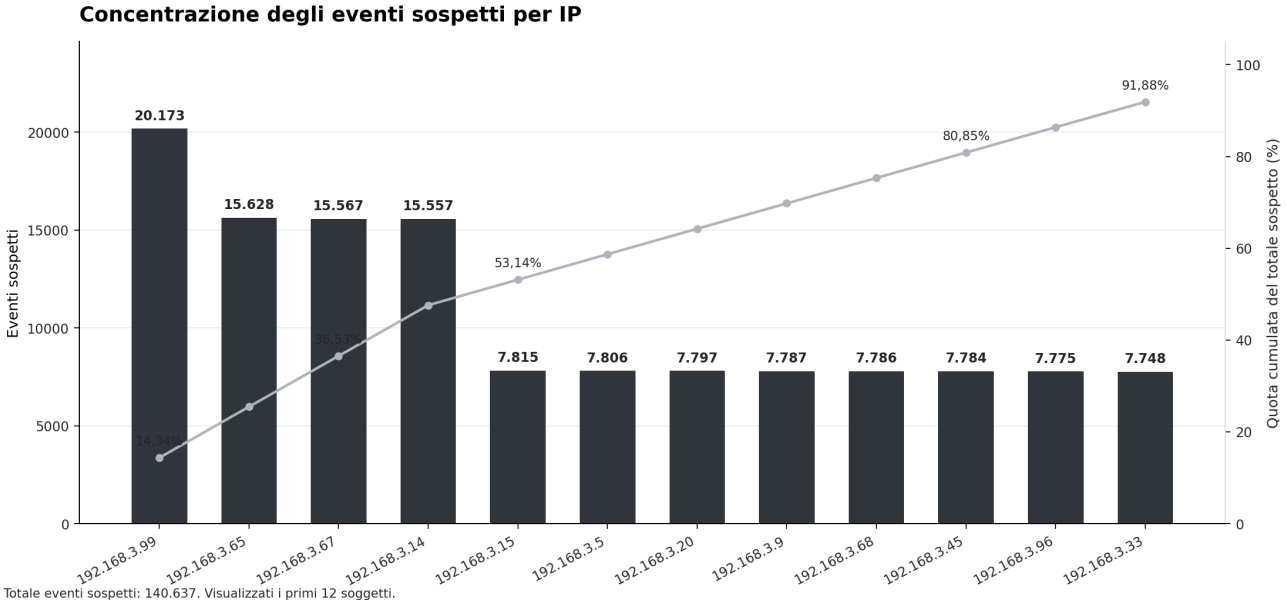


Figura 3.4: Concentrazione eventi sospetti per ip

3.2.5 Analisi temporale

La componente temporale costituisce un ulteriore elemento fondamentale nella caratterizzazione del dataset.
Sono stati determinati il timestamp del primo e dell'ultimo evento, ricavando così l'intervallo temporale complessivamente coperto dai dati:

Primo evento	21/12/2025 18:55:36
Ultimo evento	22/12/2025 01:06:31
Durata totale	6.182 h

Sulla base dell'intervallo di osservazione sono stati calcolati il numero medio di eventi al secondo e al minuto che risultano rispettivamente circa 12 e 764.
Sono stati inoltre individuati i picchi di attività, determinando:

- il massimo numero di eventi registrati all'interno dello stesso secondo: 222;
- il massimo numero di eventi registrati all'interno dello stesso minuto: 5.247.

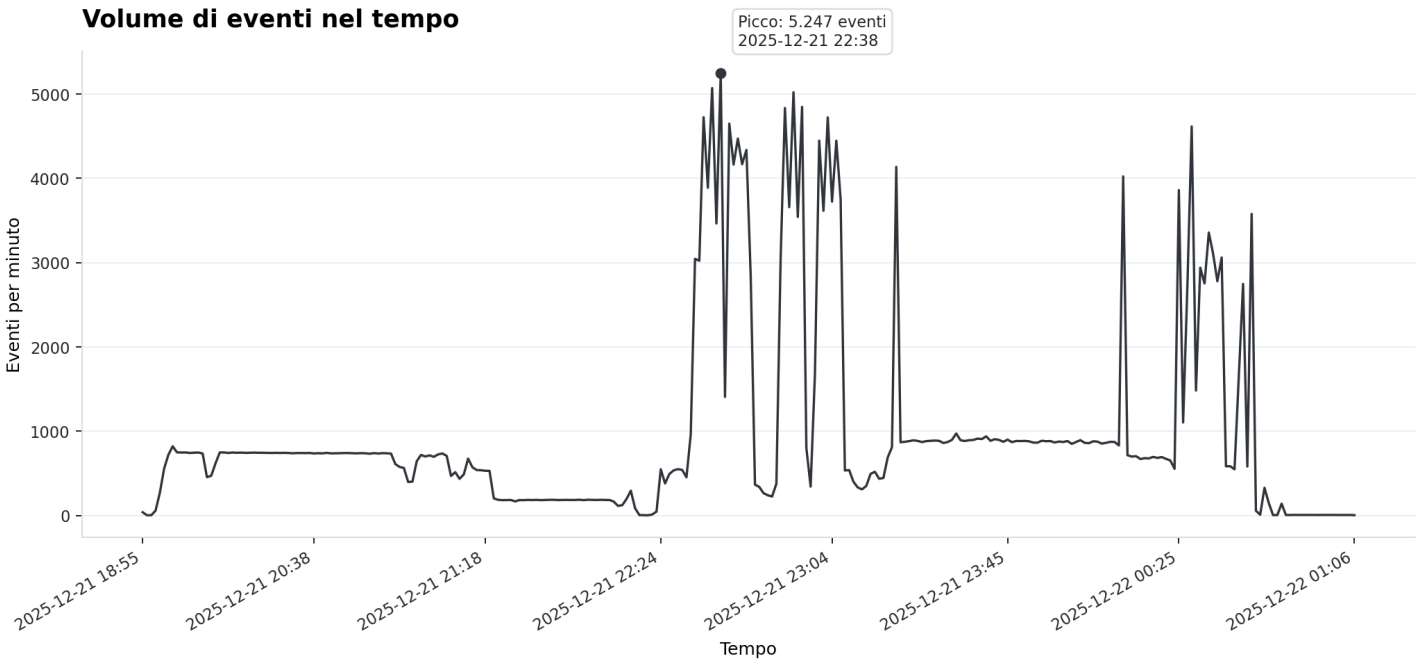


Figura 3.5: Volume eventi nel corso del tempo

Tali metriche consentono di evidenziare la presenza di burst di traffico, particolarmente rilevanti nel caso di richieste automatizzate o comportamenti ad alta frequenza.

Per approfondire ulteriormente questa caratteristica, gli eventi normali e quelli sospetti sono stati aggregati temporalmente e rappresentati separatamente. Il confronto permette di verificare se i picchi complessivi siano determinati principalmente dal traffico ordinario oppure dalla presenza di attività anomala.
La figura 3.6 mostra l'andamento del numero di eventi normali e sospetti nel tempo.

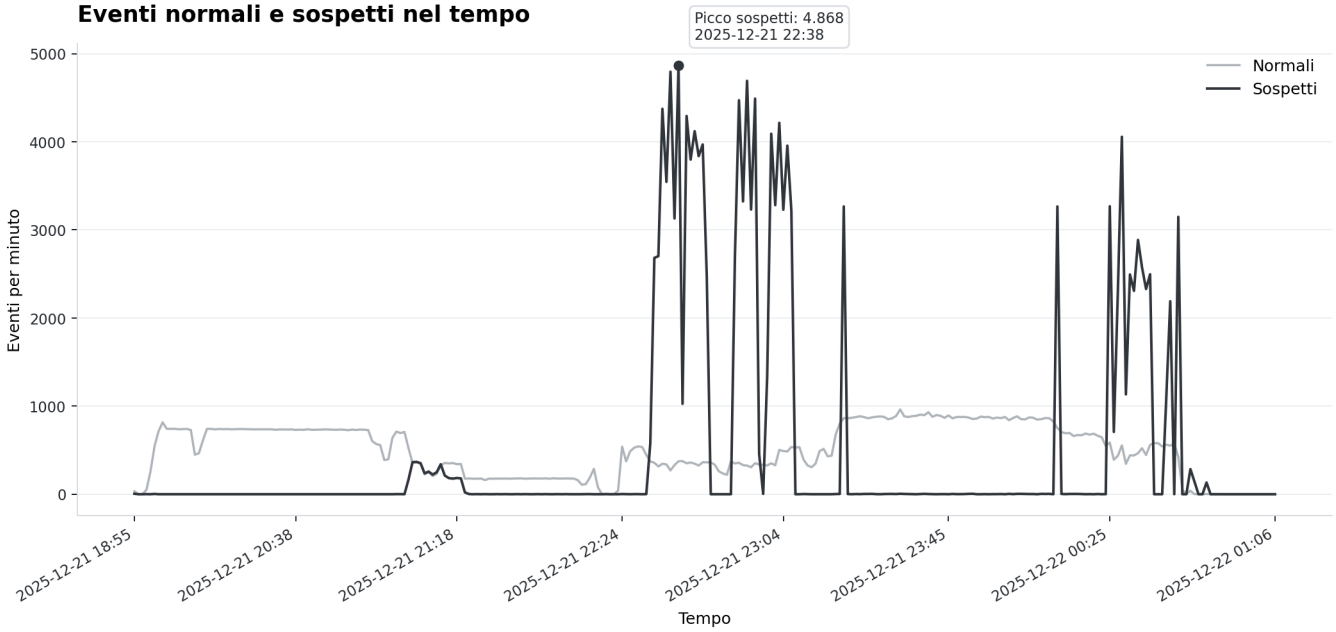


Figura 3.6: Volume eventi normali/sospetti nel corso del tempo

Una crescita improvvisa della componente sospetta, soprattutto se concentrata in un intervallo temporale ristretto, può essere indicativa della presenza di attività automatizzata o di una sequenza particolarmente intensa di richieste anomale.

Nel caso in questione bisogna tenere conto che essendo un dataset sintetico prodotto artificialmente, i vari eventi, in particolar modo gli attacchi, sono stati concentrati tutti in due periodi ristretti.

3.2.6 Analisi codici HTTP

I codici di stato HTTP sono stati analizzati per comprendere l'esito delle richieste registrate nel dataset.
Per ciascun codice è stato calcolato il numero complessivo di occorrenze e la relativa percentuale sul totale. È stato inoltre effettuato un confronto tra eventi normali e sospetti.
Questo secondo passaggio consente di valutare, ad esempio, se determinate classi di errore risultino maggiormente associate agli eventi classificati come sospetti, il che è stato proprio quello che è stato riscontrato.

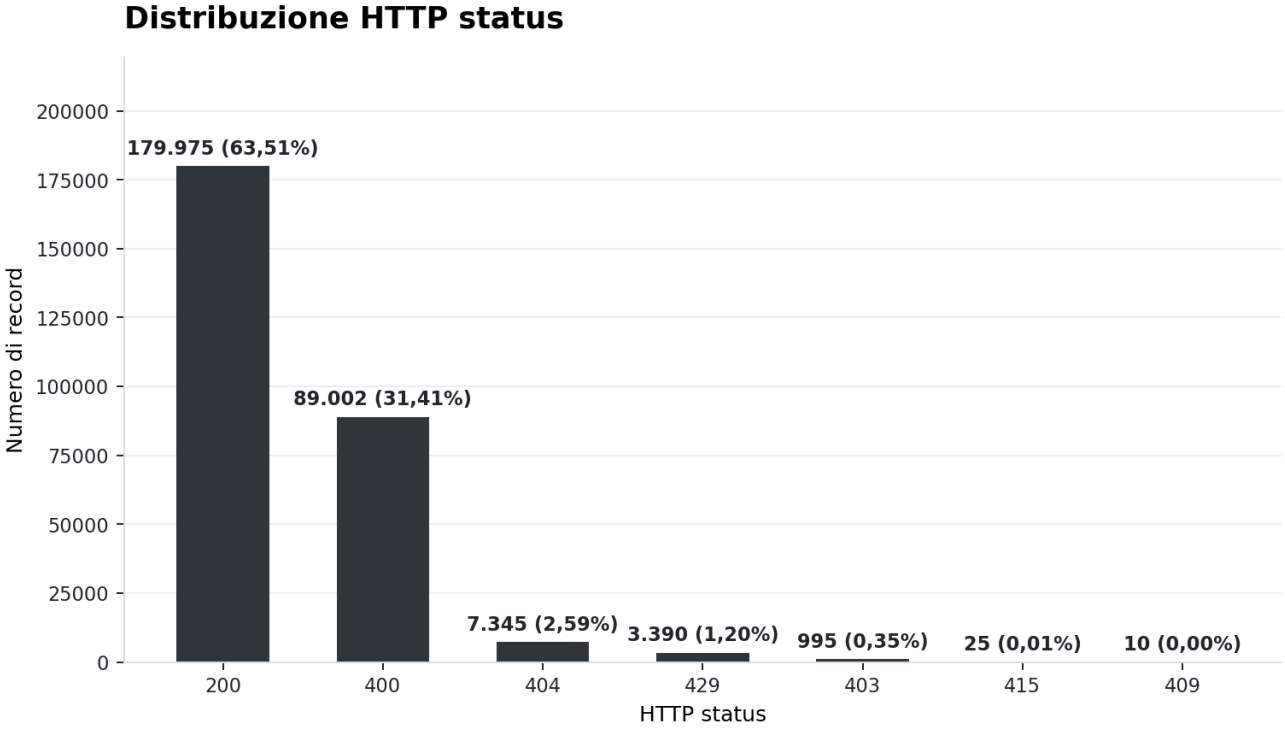


Figura 3.7: Distribuzione degli stati HTTP

L'analisi congiunta del codice HTTP e della classificazione di sicurezza risulta più informativa rispetto alla sola frequenza del codice, in quanto consente di verificare l'eventuale relazione tra errori applicativi e comportamenti anomali.

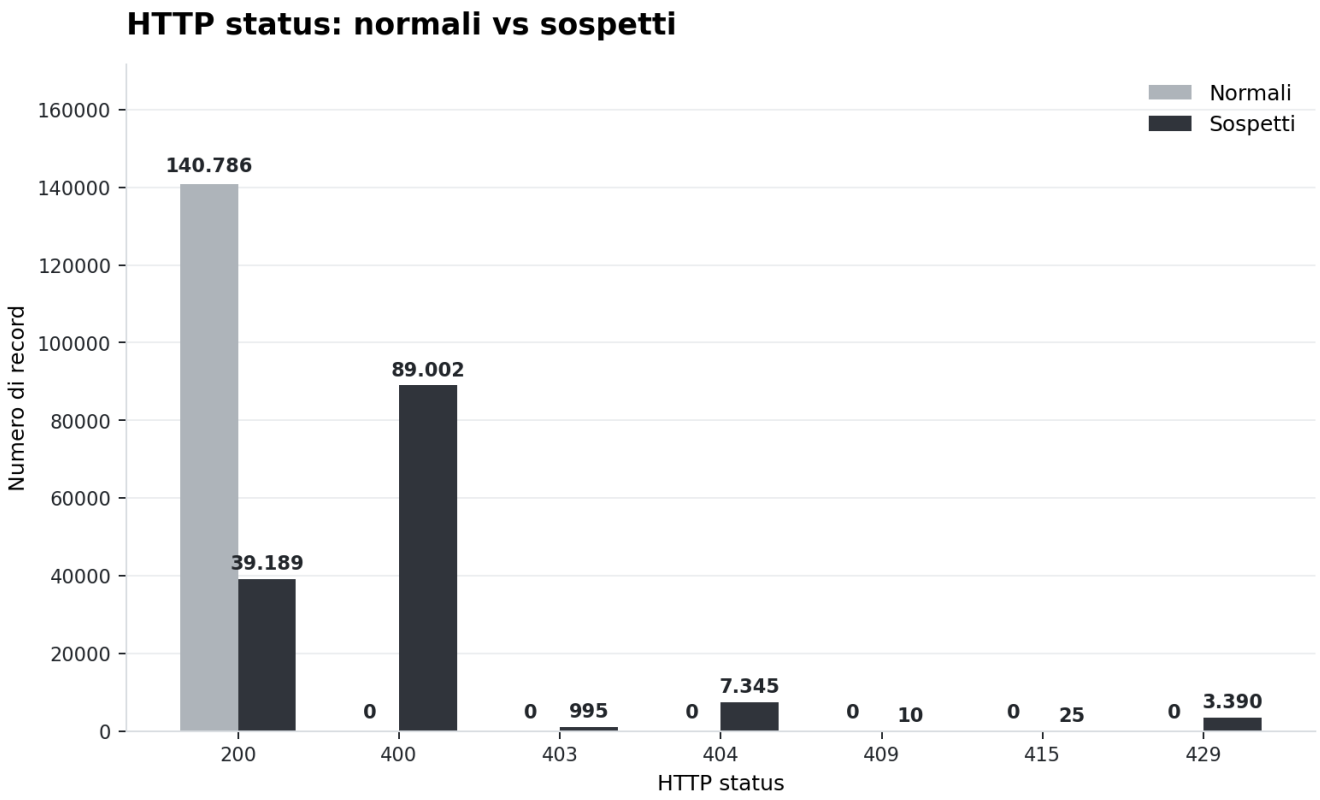


Figura 3.8: Distribuzione degli stati HTTP normali e sospetti

3.2.7 Utilizzo dei certificati client

Per quanto riguarda i meccanismi di autenticazione, è stata analizzata la presenza del campo *ClientCertificate*.
Nel dataset sono stati osservati:

- 25.593 eventi con certificato client valorizzato;
- 257.806 eventi con certificato nullo o assente.

La percentuale di eventi nei quali è presente un certificato client risulta quindi pari al 9.03%.
Questa informazione fornisce una descrizione quantitativa dell'utilizzo di meccanismi di autenticazione basati su certificato all'interno dell'ambiente osservato.

3.3 Considerazioni sull'analisi esplorativa

L'analisi descrittiva svolta sul dataset ha permesso di ottenere una prima caratterizzazione delle informazioni disponibili e di evidenziare gli aspetti maggiormente rilevanti per le successive fasi del lavoro.
In particolare, l'analisi non si è limitata alla determinazione della numerosità complessiva degli eventi, ma ha considerato congiuntamente la dimensione temporale, l'identità dei soggetti coinvolti, le sessioni applicative, le risorse accedute e la classificazione degli eventi dal punto di vista della sicurezza.
Tale caratterizzazione risulta utile non soltanto per comprendere la struttura dei dati, ma anche per individuare eventuali proprietà del dataset che possono influenzare le successive fasi di modellazione e valutazione.

È tuttavia necessario considerare un aspetto fondamentale nell'interpretazione sia dei risultati dell'analisi esplorativa sia, soprattutto, di quelli che verranno presentati nelle fasi successive. Il dataset utilizzato è stato prodotto all'interno di un ambiente controllato e mediante attività generate artificialmente, con l'obiettivo di riprodurre differenti comportamenti applicativi e scenari di sicurezza.
Di conseguenza, pur mantenendo una struttura rappresentativa di un sistema reale e raccogliendo eventi attraverso i medesimi meccanismi di logging e monitoraggio, le distribuzioni statistiche osservate non devono essere considerate direttamente equivalenti a quelle che caratterizzerebbero una rete operativa reale.
In un contesto reale, infatti, il comportamento degli utenti e dei sistemi sarebbe generalmente caratterizzato da una maggiore eterogeneità. Il volume e la frequenza delle richieste potrebbero variare in funzione dell'orario, del giorno, del numero di utenti attivi e delle differenti modalità di utilizzo dell'applicazione; le sessioni potrebbero presentare durate maggiormente variabili; uno stesso utente potrebbe accedere da dispositivi e reti differenti; potrebbero inoltre essere presenti errori occasionali, configurazioni non uniformi, attività automatizzate legittime e numerosi altri fattori difficilmente riproducibili integralmente in un ambiente sperimentale.
Anche la componente relativa agli eventi sospetti risente della modalità con cui il dataset è stato costruito. In uno scenario controllato, determinate attività anomale possono essere generate secondo sequenze, frequenze o condizioni prestabilite, producendo pattern più regolari o maggiormente separabili rispetto a quelli osservabili nel traffico reale.

Questo aspetto verrà ripreso nell'interpretazione dei risultati sperimentali e nelle considerazioni conclusive. Una naturale estensione del presente lavoro consisterebbe infatti nella validazione della metodologia su dati provenienti da un'infrastruttura reale, o su dataset caratterizzati da una maggiore variabilità e complessità, al fine di valutare la robustezza del modello rispetto a condizioni operative meno controllate e la sua effettiva capacità di generalizzazione.

3.4 Strumenti operativi

La progettazione e lo sviluppo dell'architettura sono stati realizzati interamente utilizzando strumenti software, sfruttando le numerose librerie messe a disposizione da *Python*, all'interno di *Visual Studio Code*.
Questa combinazione di strumenti ha permesso di sviluppare e integrare in un unico ambiente le diverse fasi del progetto, dalla gestione e preparazione dei dati fino alla realizzazione e alla valutazione dei modelli. In particolare, ha consentito di effettuare il caricamento e la pulizia del dataset, il preprocessing e la trasformazione delle feature, l'addestramento e l'utilizzo dei modelli *BiLSTM* e *DQN*, nonché la gestione dei risultati.
Nelle sezioni successive si vedranno nel dettaglio le principali librerie utilizzate, così da offrire una panoramica chiara degli strumenti scelti e del ruolo che ognuno di essi ha avuto nel flusso di lavoro.

3.4.1 TensorFlow / Keras



Figura 3.9: Loghi TensorFlow e Keras

TensorFlow è una piattaforma open-source per il calcolo numerico e il machine learning, mentre *Keras* [18] è l'API di alto livello, che semplifica la definizione, l'addestramento e la gestione delle reti neurali.
Costituisce il principale framework utilizzato per la realizzazione dei modelli di deep learning, impiegato sia per la realizzazione della rete BiLSTM, destinata alla classificazione delle sequenze di log e alla stima della probabilità di attacco, sia per la realizzazione della rete DQN, utilizzata per l'apprendimento della politica di risposta.
Nello specifico nell'architettura sono stati utilizzati i seguenti parametri:

- ***Sequential***: modello che permette di costruire una rete neurale organizzando i diversi livelli della rete in sequenza, uno dopo l'altro, in modo che l'output di un livello diventi l'input del livello successivo;

- ***Input***: definisce la forma dei dati forniti in ingresso al modello. Avendo un modello sequenziale specifica quindi la struttura della sequenza che il modello deve ricevere, indicando il numero di elementi della sequenza e il numero di caratteristiche associate a ciascun elemento;
- ***LSTM***: implementa una rete neurale ricorrente adatta all'elaborazione di dati sequenziali e temporali;
- ***Bidirectional***: permette di elaborare una sequenza in entrambe le direzioni, combinando le informazioni provenienti dai due versi;
- ***Dense***: rappresenta un livello completamente connesso, nel quale ogni unità riceve informazioni da tutte le unità del livello precedente. Viene generalmente utilizzato per elaborare le rappresentazioni apprese dai livelli precedenti e produrre l'output finale del modello;
- ***Dropout***: è una tecnica di regolarizzazione che disattiva casualmente una parte delle unità durante l'addestramento, riducendo quindi la dipendenza del modello da specifiche unità e contribuendo a ridurre il rischio di overfitting;
- ***LayerNormalization***: esegue una normalizzazione delle attivazioni all'interno dei livelli della rete. Questo permette di mantenere i valori delle rappresentazioni su scale più stabili durante l'addestramento, contribuendo a rendere il processo di apprendimento più stabile.

Per la gestione dell'addestramento sono state inoltre utilizzate alcune *callback*, cioè funzioni che hanno permesso di intervenire automaticamente durante il training:

- ***EarlyStopping***: interrompe l'addestramento quando le prestazioni monitorate non migliorano più, evitando un training non necessario;
- ***ReduceLROnPlateau***: riduce automaticamente il *learning rate* quando le prestazioni del modello smettono di migliorare;
- ***ModelCheckpoint***: permette di salvare automaticamente il modello durante l'addestramento, conservando la configurazione associata alle migliori prestazioni raggiunte.

3.4.2 Scikit-learn



Figura 3.10: Logo Scikit-Learn

Scikit-learn [19] è una libreria di machine learning open source per Python.
Mette a disposizione strumenti e moduli per algoritmi di apprendimento supervisionato e non, quali classificazione, regressione e clustering, strumenti per il pre-processing dei dati, la trasformazione delle feature e infine la valutazione delle prestazioni dei modelli.
Nello specifico è stata utilizzata principalmente per preparare le variabili numeriche prima del loro ingresso nel modello BiLSTM e successivamente per valutare le prestazioni ottenute dal modello.
Le componenti esatte utilizzate sono:

- **sklearn.preprocessing.StandardScaler**: utilizzato per normalizzare le (*feature*) numeriche utilizzate dal modello. Molti algoritmi di Machine Learning sono sensibili alla scala dei dati, questo evita che una feature domini sulle altre. Trasforma i dati in modo che la media di ogni colonna diventi 0 e la deviazione standard diventi 1.
  Lo StandardScaler porta quindi le feature sulla stessa scala geometrica senza però perdere le proporzioni o i pattern interni.

- **sklearn.utils.class_weight**: modulo usato per la determinazione dei pesi delle classi utilizzati durante l'addestramento per tenere conto della distribuzione delle classi. Generalmente risolve il problema dei dataset fortemente sbilanciati, cosa che rappresenta uno scenario abbastanza tipico in progetti di detection.

- **sklearn.metrics**: contiene varie metriche aggiuntive con cui si possono valutare le prestazioni del classificatore. Queste permettono di analizzare il comportamento del modello da differenti prospettive e di verificare la capacità di distinguere gli eventi normali da quelli associati ad attività sospette. Le metriche sono descritte di seguito e derivano da *confusion_matrix*, *classification_report* e *roc_auc_score*.

- **accuracy**: misura la proporzione di previsioni corrette rispetto alle previsioni totali;
- **precision**: misura la qualità delle previsioni positive indicando quanti degli elementi previsti come positivi sono davvero corretti (evita i falsi allarmi);
- **recall**: noto anche come sensibilità, valuta la capacità del modello di trovare tutti i casi positivi reali presenti nei dati, quindi la proporzione di predizioni positive corrette sul totale di positivi reali (evita i falsi negativi);
- **F1-score**: la media armonica tra precision e recall, fornisce un equilibrio tra queste due metriche, utile quando una classe domina l'altra;
- **AUC-ROC**: è l'indicatore che misura la capacità del modello di non fare confusione tra due classi. Questa metrica valuta il comportamento del modello a qualsiasi livello di soglia. Rappresenta visivamente il compromesso tra il tasso di veri positivi (TPR) e il tasso di falsi positivi (FPR).

3.4.3 Gymnasium



Figura 3.11: Logo Gymnasium

Gymnasium [20] è una libreria open-source per la definizione e la gestione di ambienti di Reinforcement Learning, attraverso i quali un agente può interagire con un ambiente, osservare il relativo stato, eseguire un'azione e ricevere un feedback sotto forma di ricompensa.
Il concetto fondamentale è proprio quello dell'Environment, ovvero l'ambiente di RL che rappresenta il mondo con cui l'agente interagisce. In questo progetto l'ambiente creato rappresenta una situazione di cyber defense, nella quale un agente deve prendere decisioni dinamiche per mitigare un possibile attacco.
Ad ogni richiesta analizzata, il sistema costruisce uno stato contenente le informazioni rilevanti per la decisione, poi l'agente utilizza tale stato per determinare quale strategia applicare, ricevendo successivamente una ricompensa che consente di valutare la qualità della decisione effettuata.

Nello specifico ci sono 4 concetti fondamentali utilizzati per la connessione al DQN:

- **Observation**: l'observation space definisce ciò che l'agente può osservare dell'ambiente. Nel caso specifico sono state definite 9 feature che verranno poi discusse nel dettaglio in seguito;
- **Action**: l'action space definisce invece quali decisioni può prendere l'agente. Anche in questo caso le diverse opzioni verranno discusse nei prossimi capitoli;
- **Reward**: è il meccanismo attraverso cui l'agente riceve un feedback. Gymnasium prevede che dopo l'azione l'ambiente restituisca una ricompensa, quindi la reward permette al DQN di capire quanto è stata efficace la decisione presa.
- **Episode**: l'episodio è una sequenza completa di interazioni, impostata in questo caso ad un massimo di 5 step per episodio. Lo step rappresenta invece una singola interazione tra agente e ambiente quindi una decisione di difesa.
  Questo permette all'agente di non prendere una sola decisione isolata, ma di poter effettuare una sequenza di decisioni mentre la situazione evolve.

3.4.4 Matplotlib

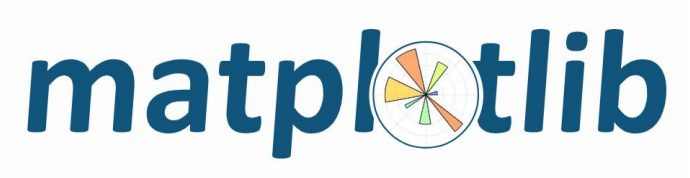


Figura 3.12: Logo Matplotlib

Matplotlib [21] è una libreria completa open source per la creazione di grafici e visualizzazioni di dati in Python.
Nel progetto è stata utilizzata principalmente durante le fasi di analisi esplorativa del dataset e valutazione sperimentale dei modelli, consentendo di rappresentare graficamente le principali caratteristiche dei dati e di analizzare in modo più immediato i risultati ottenuti.

Capitolo 4

Progettazione architettura

4.1 Architettura generale

In questo capitolo, che rappresenta il cuore del progetto, vengono illustrati nello specifico i passaggi e le fasi necessari per il funzionamento dell'intero processo.[22] L'architettura sviluppata è costituita da una pipeline composta da due componenti principali, tra loro consecutive. La prima è la rilevazione del comportamento sospetto attraverso la BiLSTM, e successivamente, l'utilizzo di queste informazioni ottenute dal rilevatore per determinare una possibile azione di risposta mediante l'agente DQN.
L'elaborazione prevede come step zero una fase di preprocessing, dove si lavora sui log grezzi prodotti dai diversi componenti poiché tali informazioni non sono direttamente utilizzabili dai modelli. Gli eventi vengono correlati sulla base dell'utente e dell'indirizzo *IP*, trasformati in una rappresentazione strutturata e infine arricchiti attraverso la fase di feature engineering. Gli eventi risultanti vengono poi ordinati temporalmente e organizzati in sequenze, rappresentando l'effettivo ingresso della componente di detection.
La BiLSTM analizza le sequenze e produce, per l'evento posto al termine della sequenza una probabilità associata alla classe sospetta. Questo valore chiamato *attack probability*, rappresenta il risultato della prima componente che viene successivamente utilizzato anche per la costruzione dello stato dell'ambiente di RL. Questo stato della componente decisionale non si basa però esclusivamente sulla probabilità di attacco ottenuta dalla BiLSTM, ma ad essa vengono aggiunte ulteriori informazioni relative al contesto dell'attività osservata. In questo modo il DQN dispone di una rappresentazione più completa ed è in grado di stimare il valore delle possibili azioni di risposta, selezionando quella con il valore Q più elevato secondo la policy appresa.
Un'ulteriore aspetto dell'architettura riguarda la possibilità di analizzare nuovi eventi durante l'esecuzione del sistema.

L'evento ricevuto in input segue le stesse fasi precedentemente descritte, viene sottoposto alle trasformazioni della fase di preprocessing, viene integrato nella sequenza utilizzata dal detector, si produce la relativa *attack probability* e infine si ottiene una decisione dal DQN. Questo consente di utilizzare i modelli addestrati all'interno di una modalità di analisi progressiva, nella quale il contesto viene aggiornato man mano che vengono osservati nuovi eventi.
Le sezioni successive sono dedicate all'approfondimento di questi processi.

4.2 Pipeline di preprocessing

Per poter essere utilizzati dai modelli proposti, i log acquisiti, ancora grezzi, devono essere sottoposti ad alcune operazioni utili a trasformarli in una rappresentazione strutturata e coerente. La primissima operazione nella fase di preprocessing è stata proprio la conversione del file *.csv* in un formato strutturato adatto che permetta un accesso più agevole ai vari campi e alle informazioni interne al *message* di ogni log relativo alle sorgenti. Questa operazione è eseguita tramite lo script *csv_to_json.py*, e l'output risultante è la creazione del file *logsprova.json*, contenente il dataset precedentemente accennato con la struttura chiave valore adatto alle fasi di correlazione. Quest'ultimo file è proprio l'input della pipeline di preprocessing. Ogni record rappresenta un evento prodotto da uno dei componenti monitorati e contiene le informazioni comuni viste precedentemente nella sezione 3.1 e in particolare nella tabella 3.1. L'obiettivo è quello di predisporre i log per le successive operazioni di correlazione e trasformazione. Il contenuto del campo *message* viene interpretato in funzione della sorgente del log, permettendo di recuperare le informazioni relative all'utente, all'indirizzo IP, al tipo di evento, al codice di risposta HTTP, alla risorsa richiesta, al ruolo dell'utente e alla presenza del certificato.
Un'ulteriore operazione preliminare è stata quella inerente la gestione dei timestamp. Non tutti i valori relativi a date e tempo avevano un formato comune. Tramite questo processo le date e gli orari presenti nel log vengono convertiti in un formato uniforme, cosi da permettere un successivo ordinamento cronologico degli eventi e il calcolo delle distanze temporali tra eventi differenti.

4.2.1 Correlazione

La presenza di più sorgenti di logging ha reso necessario correlare gli eventi appartenenti allo stesso contesto. L'obiettivo è quello di ricondurre eventi che appartengono alla stessa attività ad un unico contesto, evitando cosi di analizzare separatamente informazioni che invece descrivono la medesima interazione. Lo script interessato di questo procedimento è *correlation.py*. Questa correlazione viene effettuata considerando la coppia utente-indirizzo IP, utilizzata quindi come identificativo dell'entità osservata, e la prossimità temporale degli eventi.

Vengono dapprima separati gli eventi in base alla sorgente, poi utilizzando i log applicativi si costruiscono delle sessioni. Gli eventi appartenenti alla stessa entità, quindi con stesso utente/IP, sono quindi organizzati in sessioni considerando un timeout di 30 minuti. La sessione viene quindi mantenuta attiva per un massimo di 30 minuti dall'ultimo evento associato all'entità. Se trascorso questo tempo viene rilevato un nuovo evento, viene considerata una nuova sessione. Per ogni sessione si individuano gli eventi delle altre sorgenti relative allo stesso IP che si trovano entro una finestra temporale di correlazione di 5 minuti prima dell'inizio o dopo la fine della sessione. Si associano così informazioni provenienti da sorgenti differenti all'evento applicativo di riferimento. Facendo in questo modo, le informazioni che non appartengono direttamente al singolo log applicativo come quelle relative a dispositivo rilevate tramite *p0f* o all'autenticazione tramite *captive portal*, possono essere associate al contesto dell'evento se temporalmente compatibili. Si parte quindi dalla presenza di molti log separati e si cerca di ricostruire chi stava usando un determinato IP, in quale intervallo di tempo e quali informazioni sul dispositivo o sull'accesso alla rete possono essere associate a quella sessione. La correlazione rappresenta un passaggio fondamentale perché permette di passare dalla semplice sequenza di log indipendenti ad una rappresentazione organizzata per entità e sessioni.

4.2.2 Creazione log finali

Dopo la fase di correlazione, le informazioni associate ai diversi eventi vengono trasformate in una struttura più semplice e uniforme attraverso una fase di flattening (appiattimento). L'obiettivo di questa fase è quello di ottenere, per ogni evento, un singolo record contenente le informazioni necessarie alle successive operazioni di encoding e feature engineering, indipendentemente dalla sorgente da cui queste informazioni derivano. La funzione adibita a ciò è *create_logs.py* che estrae dai dati correlati i principali attributi dell'evento. Nella tabella 4.1 vengono illustrati i campi considerati, relativi all'identità dell'entità, al tipo di evento, alla sorgente, al ruolo dell'utente, alla risorsa interessata al codice di risposta HTTP, al dispositivo e al contesto applicativo. Vengono estratti il ruolo dell'utente e le eventuali informazioni relative ai privilegi, in particolare, determinati ruoli come *Admin*, *Administrator*, *SuperAdmin* e *SecurityAdmin* sono quelli identificati come privilegiati. Questa informazione viene mantenuta nel log finale attraverso un apposito campo booleano. Quando sono presenti più ruoli si utilizza l'ultimo valore disponibile, mentre in assenza dell'informazione viene assegnato di default il ruolo generico *User*. Viene identificata anche la risorsa associata alla richiesta e viene utilizzato il nome della tabella quando disponibile, ricorrendo al nome del database come informazione alternativa qualora il primo non sia presente. In questo modo viene mantenuto un riferimento alla risorsa coinvolta anche in presenza di informazioni parziali. Viene inoltre verificato se si tratta di una risorsa critica o meno, confrontandola con un elenco definito con alcune categorie,

tra cui quelle relative ad autenticazione, credenziali, sicurezza e amministrazione. Per ogni sessione vengono anche recuperate le informazioni sul dispositivo associate ai dati *p0f*, come sistema operativo, indirizzo MAC e hostname. Questi dati vengono associati a ciascun evento della sessione, permettendo di mantenere nel log finale anche il contesto relativo al dispositivo dal quale l'azione è stata effettuata. Infine, per ogni evento vengono raccolte le principali informazioni applicative e di sicurezza, tra cui tipo di evento, sorgente, codice HTTP, presenza di attività sospette, certificato client e route utilizzata. A queste informazioni è stato aggiunto un valore iniziale di rischio associato all'evento, *user_risk*. Questo viene calcolato sulla base di alcuni indicatori semplici, la presenza di un'attività sospetta aumenta il rischio, così come gli errori HTTP (*status_code*>=400), mentre il codice 429 (troppe richieste) introduce un ulteriore incremento. Il valore viene infine limitato a un intervallo compreso tra 0 e 1. Il risultato è una lista di log normalizzati e arricchiti, in cui ogni record rappresenta un singolo evento e contiene tutte le informazioni utili per le fasi successive.

Campo	**Descrizione**
`user`	Identificativo dell'utente associato all'evento.
`ip`	Indirizzo IP sorgente associato all'evento.
`event_time`	Timestamp dell'evento.
`event_type`	Tipologia dell'evento.
`source`	Sorgente dell'evento.
`status_code`	Codice di stato HTTP associato all'evento.
`suspicious`	Indicatore booleano relativo al comportamento sospetto, ovvero la label per la classificazione.
`role`	Ruolo dell'utente. Se non presente, di default: "*User*".
`privileged`	Indicatore booleano relativo al ruolo dell'utente se privilegiato o meno. Un utente è definito privilegiato quando il suo ruolo è uno tra *Admin*, *Administrator*, *SuperAdmin* o *SecurityAdmin*.
`resource`	Risorsa coinvolta nell'evento.
`critical_resource`	Indicatore booleano relativo alla criticità della risorsa. Sono state definite critiche le seguenti risorse: *AspNetUsers*, *Users*, *Identity*, *Authentication*, *Credentials*, *Security*, *Admin*.
`certificate`	Certificato client associato all'evento, se presente.
`os`	Sistema operativo associato all'evento.
`mac`	Indirizzo MAC associato al dispositivo rilevato.
`hostname`	Nome host associato al dispositivo, qualora sia stato possibile rilevarlo.
`route`	Endpoint applicativo coinvolto nella richiesta.
`user_risk`	Punteggio iniziale di rischio associato all'evento, tra 0 e 1. Calcolato sulla base della label e del codice di stato HTTP.

Tabella 4.1: Descrizione dei campi del log finale

4.2.3 Encoding variabili categoriche

Le informazioni ottenute dalle fasi precedenti comprendono diverse variabili di natura categorica, espresse sotto forma di stringhe. Quelle su cui si è lavorato, in funzione dell'utilizzo nel vettore della BiLSTM sono il tipo di evento, la sorgente del log, il ruolo dell'utente e la risorsa interessata. Questi valori testuali non possono essere utilizzati direttamente dalla rete neurale, devono prima essere convertiti in una rappresentazione numerica. Viene quindi applicata una codifica delle categorie, dove, per ognuno dei quattro campi, viene creato un encoder che assegna a ciascun valore distinto un identificativo numerico. La codifica viene anche progettata per la gestione dei valori non presenti durante la fase di addestramento, associandoli a una categoria "*UNKNOWN*". Questa scelta permette di rendere il preprocessing più robusto anche quando il sistema opera su eventi contenenti categorie mai osservate.
Si è scelto poi di salvare gli encoder per poi poter essere ricaricati, garantendo che le stesse categorie mantengano le stesse codifiche numeriche al successivo riutilizzo del modello. Le variabili categoriche codificate vengono successivamente mantenute distinte dalle caratteristiche numeriche. Tale separazione consente di applicare la normalizzazione esclusivamente alle variabili per le quali essa è appropriata, evitando di interpretare gli identificativi categorici come valori caratterizzati da un ordinamento numerico.
Si precisa che l'operazione di encoding va distinta dalla selezione delle feature utilizzate dalla BiLSTM. La struttura *flat* può contenere infatti un numero di informazioni superiore rispetto a quelle che invece saranno effettivamente utilizzate dal modello. L'encoding rende alcune di queste informazioni disponibili in forma numerica, mentre la successiva fase di feature engineering determina quali variabili utilizzare e come combinarle per costruire il vettore di input definitivo.

4.2.4 Feature Engineering

La fase di *feature engineering* è quella incaricata di trasformare le informazioni disponibili ottenute nelle fasi precedenti in un insieme di caratteristiche adatte all'analisi automatica del comportamento. Vengono trasformate in un insieme di variabili in grado di rappresentare non soltanto le caratteristiche del singolo evento, ma anche alcuni aspetti temporali e comportamentali dell'attività osservata. In altre parole, consiste nella selezione e nella trasformazione delle informazioni dei log in variabili numeriche capaci di rappresentare gli aspetti più rilevanti di ciascun evento.
La progettazione delle feature è stata effettuata considerando come entità di riferimento la coppia (User, IP), in coerenza con la successiva costruzione delle sequenze temporali. Si precisa che non tutte le informazioni disponibili nel flat log vengono però utilizzate direttamente come input della rete.

Alcuni attributi vengono trasformati in feature numeriche, altri vengono codificati come variabili categoriche, mentre ulteriori informazioni vengono mantenute nella struttura di contesto dell'evento e utilizzate nelle successive fasi della pipeline o non utilizzate affatto. Per ogni evento viene costruito un vettore composto da 15 feature, mantenute in ordine fisso, che sarà il vettore finale destinato alla BiLSTM.

4.2.4.1 Feature categoriche

Le prime quattro variabili sono di tipo categorico e vengono convertite in identificativi numerici mediante gli encoder definiti precedentemente. Nel modello vengono utilizzate quindi quattro caratteristiche di questo tipo. La prima è ***event_id***, che rappresenta la codifica del tipo di evento osservato. Permette di distinguere le diverse operazioni registrate dal sistema e di fornire alla rete informazioni sulla natura dell'attività effettuata. La seconda è ***source_id***, relativa alla sorgente che ha prodotto il log, consente di distinguere eventi provenienti da differenti componenti del sistema di monitoraggio. La terza caratteristica è ***role_id***, ottenuta dalla codifica del ruolo associato all'utente. Infine, come ultima feature categorica, ***resource_id***, che identifica in forma numerica la risorsa interessata dall'evento.

4.2.4.2 Feature temporali

La prima informazione temporale considerata è ***delta***, rappresentante il tempo trascorso, espresso in secondi, tra l'evento corrente e quello precedente appartenente alla stessa entità. Per il primo evento non essendo disponibile un evento precedente, il valore viene posto a zero. Poiché gli intervalli temporali possono assumere valori molto differenti, viene inoltre calcolata ***delta_log***, ottenuta applicando la funzione *log1p* a delta. Gli attacchi informatici basati su script o bot generano eventi a distanze di tempo infinitesimali, questa funzione mantiene una precisione microscopica nel distinguere la differenza tra eventi separati da 1 millisecondo rispetto a eventi separati da 1 microsecondo, un segnale cruciale per identificare anomalie automatizzate. Inoltre, l'utilizzo della trasformazione logaritmica permette di ridurre l'effetto di valori temporali molto elevati e di rappresentare in modo più compatto intervalli caratterizzati da scale differenti.
Vengono poi estratti ***hour***, che rappresenta l'ora del giorno in cui si verifica l'evento, e ***weekday***, corrispondente al giorno della settimana. Queste informazioni permettono di fornire al modello una collocazione temporale dell'attività osservata. A queste viene aggiunta infine ***is_night***, una variabile binaria che identifica l'attività svolta nelle fasce orarie considerate notturne. Nel progetto si è scelto di considerare orari notturni quelli precedenti alle ore 06:00 oppure successivi alle ore 22:00. Quando l'evento ricade in questa fascia temporale, il corrispondente valore sarà 1, mentre assume valore 0 negli altri casi.

Nel loro insieme, queste caratteristiche consentono di rappresentare sia la distanza temporale tra eventi consecutivi sia il momento della giornata e della settimana nel quale l'attività viene effettuata.

4.2.4.3 Feature comportamentali

Per descrivere il comportamento recente dell'entità vengono utilizzate due caratteristiche. La prima è ***failed***, variabile binaria che indica se la richiesta associata all'evento è terminata con un errore. Viene calcolata a partire dal codice di risposta HTTP ed è impostata ad 1 quando quest'ultimo è maggiore o uguale a 400 e a zero altrimenti. Poi viene calcolato ***requests_5m*** che indica il numero di richieste osservate nella finestra temporale mobile di cinque minuti relativa alla stessa entità user,IP. Per ottenere questo valore viene mantenuta una struttura contenente i timestamp degli eventi dell'utente e vengono rimossi progressivamente quelli che si trovano al di fuori della finestra considerata. In questo modo la feature fornisce un'indicazione della pressione di richieste generata dall'entità e di affiancare alla valutazione del singolo evento una misura della sua attività recente.

4.2.4.4 Feature di sicurezza

Il quarto gruppo è relativo alle caratteristiche direttamente legate al contesto di sicurezza della richiesta. La feature ***has_certificate*** assume un valore binario e indica la presenza o meno del certificato client associato all'evento. ***Privileged*** identifica invece se l'utente associato all'evento appartiene ad un ruolo considerato privilegiato, anch'essa assume valore binario. Infine, ***critical_resource*** identifica se la risorsa richiesta appartiene all'insieme delle risorse che si è scelto di considerare come critiche. Tra le risorse critiche sono comprese *AspNetUsers*, *Users*, *Identity*, *Authentication*, *Credentials*, *Security* e *Admin*.
Queste feature permettono di distinguere il livello di esposizione associato alla richiesta in funzione dell'identità dell'utente e della risorsa coinvolta.

4.2.4.5 Feature di rischio

L'ultima caratteristica del vettore è ***historical_risk***, progettata per rappresentare il rischio accumulato dall'entità sulla base del comportamento osservato in precedenza. Rappresenta quindi la persistenza nel tempo di un comportamento potenzialmente rischioso associato alla stessa entità. Il valore associato all'evento corrente deriva esclusivamente dallo storico precedente e viene successivamente aggiornato in funzione del comportamento osservato. In particolare, il rischio precedente viene attenuato mediante un fattore di decadimento pari a 0,95 e può essere incrementato quando vengono osservati determinati segnali comportamentali. Il valore viene limitato nell'intervallo da 0 a 1.

In presenza di una richiesta fallita viene aggiunto un incremento di 0,10, mentre quando il numero di richieste nella finestra di cinque minuti supera una determinata soglia impostata, viene aggiunto un ulteriore incremento di 0,15. La soglia è stata pensata per rappresentare il 95° percentile osservato relativo alle richieste nei cinque minuti esclusivamente degli eventi etichettati normali.
La caratteristica non rappresenta quindi esclusivamente il rischio del singolo evento, ma conserva una memoria sintetica del comportamento precedente dell'entità. In questo modo la valutazione di un evento può tenere conto anche delle osservazioni antecedenti, oltre alle informazioni immediatamente associate alla richiesta. Un aspetto importante di questa implementazione riguarda la prevenzione del target leakage. La label dell'evento corrente non viene utilizzata per costruire il valore di *historical_risk* fornito allo stesso evento: l'informazione relativa al comportamento corrente viene utilizzata soltanto per aggiornare il rischio destinato agli eventi successivi. In questo modo la feature rappresenta esclusivamente informazioni disponibili prima della classificazione dell'evento considerato.

Ogni evento dispone cosi di una rappresentazione numerica composta dalle 15 feature definite. La sua struttura combina informazioni statiche relative all'evento con variabili dinamiche che descrivono il comportamento recente e la sua evoluzione temporale.

4.2.5 Costruzione sequenze

I singoli eventi devono ora essere organizzati in sequenze temporali affinché possano essere elaborati dalla rete BiLSTM. Questa fase è necessaria appunto perché il modello non opera su un singolo evento isolato, ma utilizza la successione degli eventi per l'individuazione di comportamenti sospetti. La costruzione delle sequenze mantiene la stessa definizione di entità utilizzata nelle fasi precedenti, identificata dalla coppia (User,IP). Gli eventi appartenenti alla stessa entità vengono quindi raggruppati e ordinati cronologicamente in base al timestamp. In questo modo la sequenza rappresenta l'evoluzione temporale dell'attività associata a uno specifico utente e indirizzo IP. Viene mantenuta una finestra temporale della durata di 5 minuti per ogni entità. Durante l'elaborazione gli eventi che si trovano al di fuori di questo intervallo vengono progressivamente rimossi dalla finestra, mentre quelli che ne fanno parte vengono aggiunti. Per evitare la costruzione di sequenze troppo brevi e garantire un contesto minimo, vengono considerate valide solamente le finestre temporali contenenti almeno cinque eventi. La finestra viene fatta poi avanzare e per evitare la costruzione questa volta di un numero eccessivamente elevato di sequenze fortemente sovrapposte, viene utilizzato uno stride (passo) di cinque eventi. Inoltre, quando all'interno della finestra siano presenti più di 50 eventi, vengono considerati solo quelli più recenti, imponendo una lunghezza massima alla sequenza e mantenendo il contesto temporale più vicino all'evento di interesse.

Le sequenze poi non avranno necessariamente tutte la stessa lunghezza, ognuna può avere un numero di eventi diverso dalle altre. Una rete neurale però normalmente vuole un tensore con dimensione regolare. Una funzione di *padding* si occupa proprio di questo, ovvero rendere tutte le sequenze della stessa lunghezza. Il massimo di eventi per una sequenza come detto è impostato a 50, quindi alle sequenze con numero di eventi inferiori alle 50 unità vengono aggiunti degli zero a tutte le feature per ogni sequenza vuota aggiunta. Un aspetto particolarmente importante riguarda la definizione dell'etichetta associata alla sequenza. La label non viene assegnata all'intera sequenza sulla base di una maggioranza delle etichette presenti, ma viene determinata utilizzando l'ultimo evento della sequenza. Considerando una sequenza composta dagli eventi: $[e_1, e_2, \ldots, e_{n-1}, e_n]$ gli eventi fino a (e_{n-1}) costituiscono l'input della BiLSTM, mentre l'etichetta associata all'ultimo evento (e_n) viene utilizzata come target. Questa impostazione permette di formulare il problema come una previsione relativa all'evento successivo rispetto al contesto osservato. La scelta consente inoltre di mantenere separati gli input dal relativo target, evitando di fornire direttamente alla rete l'informazione che si intende prevedere. Il modello riceve quindi una sequenza di vettori di feature e deve determinare la probabilità che l'evento successivo, sulla base del comportamento osservato nella finestra temporale e rappresentato dalla label associata all'ultimo elemento della finestra, appartiene alla classe sospetta.

Prima dell'ingresso nella rete viene anche eseguita la normalizzazione delle feature numeriche. Questa operazione è necessaria per rendere confrontabili variabili caratterizzate da scale differenti, evitando che quelle con valori numericamente più elevati assumano un peso eccessivo durante l'addestramento. I quattro identificativi categoriali non vengono invece normalizzati, poiché rappresentano codici discreti e non quantità continue. La normalizzazione viene quindi applicata alle restanti feature numeriche della sequenza. Viene usato lo *StandardScaler* di *scikit-learn*, dove il valore normalizzato z equivale a:

$$z = \frac{x - \mu}{\sigma}$$

e x è il valore originale, μ la media della feature e σ la deviazione standard della feature.
I parametri utilizzati per la normalizzazione vengono determinati sui dati di addestramento e successivamente riutilizzati sul test set, evitando di ricalcolarli sui dati da classificare. Viene utilizzata la funzione *fit_ transform()* sui dati del train set, e *trasform()* sui dati del test set. La prima funzione calcola media e deviazione standard del train e poi usa quelle statistiche per normalizzare. L'uso invece della sola seconda funzione significa che il test viene normalizzato usando le statistiche relative ai dati di training. Questo viene fatto per evitare *data leakage*.

Al termine di queste operazioni, ogni sequenza è rappresentata da un insieme ordinato di vettori contenenti le 15 feature definite. La struttura risultante costituisce quindi l'input della rete BiLSTM. I log grezzi sono stati progressivamente trasformati, correlati, appiattiti, codificati e arricchiti con feature temporali e comportamentali, fino a ottenere sequenze utilizzabili dal modello di detection.

4.3 Progettazione e implementazione BiLSTM

Le sequenze ottenute vengono fornite alla rete BiLSTM sotto forma di tensore tridimensionale la cui struttura è rappresentata come (N,T,15), dove N indica il numero complessivo di sequenze utilizzate, T rappresenta il numero di timestep della sequenza, quindi 50 come impostato anche con il padding, e 15 corrisponde al numero di feature utilizzate per rappresentare ciascun evento. Ogni timestep contiene quindi il vettore delle feature relative a un singolo evento, mentre la dimensione temporale consente alla rete di analizzare la successione degli eventi all'interno della finestra considerata. La BiLSTM riceve pertanto una rappresentazione nella quale le informazioni relative alle caratteristiche del singolo evento sono associate alla loro evoluzione temporale.

4.3.1 Architettura della rete

Ogni singola sequenza è costituita dall'insieme ordinato di eventi relativi alla stessa entità e ogni evento è rappresentato dal vettore di feature. Formalmente la singola sequenza è indicata come $[X = (x_1, x_2, \ldots, x_T)]$ con T che rappresenta il numero di eventi contenuti nella sequenza e ogni vettore contiene poi le feature associate all'evento al tempo t. Di conseguenza, l'input della rete ha dimensionalità (T,F) per ogni campione, mentre durante l'elaborazione a batch TensorFlow considera anche la dimensione relativa al numero di sequenze elaborate come detto precedentemente. Nel codice, tale struttura è definita attraverso il livello

$$Input(shape = (sequence_length, feature_number))$$

Il parametro *sequence_length* rappresenta quindi il numero massimo di elementi temporali della sequenza, mentre *feature_number* indica il numero di caratteristiche utilizzate per descrivere ciascun evento. Il livello *Input* non effettua una trasformazione delle informazioni, ma definisce formalmente la struttura dei dati che la rete deve ricevere.

L'architettura implementata è composta dai seguenti blocchi:

$$[\text{Input} \rightarrow \text{BiLSTM}(128) \rightarrow \text{LayerNorm} \rightarrow \text{Dropout} \rightarrow \text{BiLSTM}(64) \rightarrow \text{LayerNorm} \rightarrow \text{Dropout} \rightarrow \text{Dense}(64, \text{ReLU}) \rightarrow \text{Dropout} \rightarrow \text{Dense}(1, \text{Sigmoid})]$$

L'architettura è composta da più livelli organizzati in modo sequenziale, con l'obiettivo di estrarre progressivamente le caratteristiche temporali presenti nei dati e trasformarle in una rappresentazione utilizzabile per la classificazione finale.
La rete può essere interpretata come una successione di tre fasi principali. La prima e la seconda sono dedicate all'elaborazione temporale mediante gli strati ricorrenti, successivamente una componente completamente connessa elabora la rappresentazione ottenuta e, infine, un singolo neurone produce la probabilità associata alla classe sospetta.
La prima componente della rete è costituita da uno strato ***Bidirectional LSTM***, formato da un livello LSTM con **128 unità**. Una unità LSTM rappresenta un'istanza della cella ricorrente capace di mantenere e aggiornare una propria rappresentazione dello stato della sequenza. Il numero di unità determina quindi la dimensionalità dello stato nascosto prodotto dalla singola direzione della LSTM. Come ampiamente detto l'utilizzo della configurazione bidirezionale permette alla rete di elaborare la sequenza sia in ordine temporale diretto sia inverso, combinando le due rappresentazioni ottenute. In questo modo, ogni elemento della sequenza può essere interpretato considerando le relazioni con gli altri eventi presenti nella stessa finestra temporale. Possedendo 128 unità entrambe le LSTM, l'output combinato della BiLSTM ha dimensionalità doppia ovvero di 256 unità. L'uso del parametro *return_sequences=True* consente poi al primo livello di restituire una rappresentazione per ciascun elemento della sequenza, invece del solo output finale, necessaria per fornire l'output al successivo livello ricorrente. All'interno dello strato LSTM sono poi presenti altri due parametri, *dropout=0.2* e *recurrent_dropout=0.1*. Viene quindi applicato un dropout sia sulle connessioni di ingresso sia sulle connessioni ricorrenti. È importante distinguerli però dal layer Dropout presente successivamente nella rete. Il primo parametro applica una regolarizzazione alle trasformazioni effettuate sugli input della cella LSTM, mentre il secondo interviene sulle trasformazioni associate allo stato ricorrente. In entrambi i casi la riduzione delle connessioni attive viene effettuata durante l'addestramento e non durante la fase di inferenza. In altri termini, durante il training la rete non può fare affidamento sistematicamente sulle stesse connessioni. Questo rende più difficile l'apprendimento di rappresentazioni eccessivamente specializzate sui dati di addestramento e contribuisce quindi a ridurre il fenomeno di overfitting, ossia la situazione in cui il modello si adatta molto bene ai dati utilizzati per l'apprendimento ma generalizza peggio su dati non osservati.
Successivamente, l'output del primo livello BiLSTM viene elaborato dallo strato ***LayerNormalization***. Questa normalizza le attivazioni per ciascun esempio, anziché calcolare le statistiche sull'intero batch come avviene nella *BatchNormalization*. Nel modello sviluppato, la normalizzazione ha la funzione di rendere più controllata la distribuzione delle attivazioni prodotte dai livelli ricorrenti prima che queste vengano sottoposte alle elaborazioni successive. Non modifica il contenuto concettuale della sequenza, ma ne riorganizza numericamente le rappresentazioni.

Dopo la normalizzazione viene applicato un ulteriore ***Dropout***, questa volta non interno alla cella LSTM, ma che agisce direttamente sull'output prodotto dal livello precedente. Questo esclude casualmente dall'elaborazione il 30% delle unità durante l'addestramento, visto che è impostato a 0.3, prevenendo l'overfitting.
La seconda componente ricorrente è costituita da un ulteriore ***Bidirectional LSTM***, questa volta con **64 unità**. La principale differenza rispetto al primo livello è rappresentata dal parametro *return_ sequences=False*. Questo strato non restituisce l'intera sequenza di rappresentazioni, ma restituisce solamente la rappresentazione finale della sequenza elaborata dal livello ricorrente. In questo modo l'insieme degli eventi della finestra viene progressivamente trasformato in un singolo vettore di caratteristiche, utilizzabile dai livelli completamente connessi successivi. Il passaggio può quindi essere rappresentato schematicamente come:

$$[(T, 256) \rightarrow (128)]$$

La funzione del secondo BiLSTM può essere interpretata come una fase di sintesi della sequenza. Il primo livello produce rappresentazioni temporali dettagliate per i singoli eventi; il secondo livello riceve tali rappresentazioni e costruisce una rappresentazione più compatta del comportamento complessivamente osservato nella finestra. L'output viene successivamente sottoposto a un secondo livello *LayerNormalization* e a un *Dropout(0.3)*. La loro funzione è analoga a quella descritta per il primo blocco: stabilizzare la rappresentazione prodotta e introdurre una ulteriore forma di regolarizzazione.
La presenza di due livelli ricorrenti permette quindi di effettuare una progressiva trasformazione delle informazioni temporali, passando dalla rappresentazione dei singoli elementi della sequenza a una rappresentazione complessiva utilizzabile per la classificazione.
Dopo l'elaborazione ricorrente, l'output viene passato a uno strato ***Dense*** composto da 64 neuroni con funzione di attivazione *ReLu*. Un livello Dense o *fully connected layer*, è appunto un livello completamente connesso che collega ciascuna unità di ingresso a tutte le unità del livello successivo. La funzione ReLu (*Rectified Linear Unit*) $ReLU(z) = \max(0, z)$ mantiene i valori positivi e porta a zero quelli negativi. Il suo utilizzo introduce una non linearità nella rete, permettendo al livello Dense di trasformare la rappresentazione ottenuta dalle LSTM in uno spazio più adatto alla successiva classificazione. Viene quindi ridotta la rappresentazione da 128 a 64 dimensioni e successivamente è applicato un nuovo *Dropout(0.25)*, che introduce un'ulteriore regolarizzazione, questa volta sulla rappresentazione prodotta dal classificatore intermedio. Alla fine la rete termina con uno strato Dense costituito da un singolo neurone con funzione di attivazione *sigmoide*. Questo è coerente con la natura binaria del problema, nel quale l'evento deve essere associato a una delle due classi previste: comportamento normale oppure comportamento sospetto.

Il valore prodotto prima dell'attivazione è trasformato dalla funzione sigmoide $[\sigma(z) = \frac{1}{1+e^{-z}}]$ che restituisce sempre un valore compreso nell'intervallo [0,1].
È importante sottolineare che il modello non produce direttamente una contromisura. Nel modello tale valore viene interpretato come una stima della probabilità associata alla classe sospetta e costituisce l'***attack probability***, che verrà poi utilizzata in seguito come stato del RL e per determinare l'azione di mitigazione.

4.3.2 Configurazione dell'addestramento

Dopo aver definito l'architettura della rete, è necessario configurare i parametri utilizzati durante la fase di addestramento. In particolare, vengono definiti la funzione di perdita, l'algoritmo di ottimizzazione, il learning rate, la gestione dello sbilanciamento tra le classi, la dimensione dei batch e il numero massimo di epoche. A questi elementi si aggiungono alcuni meccanismi di controllo dell'addestramento, utilizzati per evitare un eccessivo adattamento ai dati e per conservare la configurazione della rete che fornisce le prestazioni migliori.
Poiché il problema affrontato è una classificazione binaria, l'ottimizzazione della rete viene effettuata utilizzando la **BCE** (*Binary Cross-Entropy*) come funzione di perdita, appropriata proprio per problemi di classificazione binaria nei quali l'output della rete rappresenta una probabilità. Indicando con y_i il valore reale della classe e con p_i la probabilità stimata dalla rete, la funzione di perdita può essere espressa come:

$$Loss = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1-y_i)\log(1-p_i)]$$

La funzione penalizza quindi le predizioni che si discostano dalle etichette reali. Quando il modello assegna una probabilità elevata alla classe corretta, il contributo alla perdita è ridotto; al contrario, una predizione molto sicura ma errata produce una penalizzazione elevata.

Per l'aggiornamento dei pesi della rete viene usato l'ottimizzatore **Adam** (*Adaptive Moment Estimation*), un metodo di ottimizzazione basato sulla discesa del gradiente che utilizza stime adattive del primo e del secondo momento dei gradienti, configurato con un learning rate pari a 0.0005. L'ottimizzatore ha il compito di modificare progressivamente i parametri della rete sulla base dell'errore calcolato durante l'addestramento, mentre il learning rate determina l'entità degli aggiornamenti applicati ai pesi a ogni iterazione.
Un valore più elevato consente variazioni più rapide dei pesi, mentre un valore più contenuto rende l'aggiornamento progressivo e controllato.

Durante l'addestramento viene inoltre considerata la distribuzione delle classi presenti nei dati. Per ridurre l'effetto di eventuali differenze nella loro frequenza, vengono calcolati automaticamente dei pesi di classe bilanciati a partire dalle etichette del training set. Tali pesi vengono successivamente utilizzati nella funzione di perdita durante l'addestramento, facendo sì che gli errori commessi sulle classi meno rappresentate abbiano un'incidenza proporzionalmente maggiore.

Il processo di addestramento è stato impostato con un massimo di 50 epoche e una dimensione del batch pari a 32 campioni. Un'epoca corrisponde a una completa elaborazione dei campioni utilizzati per l'addestramento, mentre il *batch size* indica il numero di sequenze elaborate prima di effettuare un aggiornamento dei pesi. L'utilizzo di batch relativamente contenuti consente di eseguire più aggiornamenti durante ogni epoca, mantenendo al tempo stesso un carico computazionale gestibile.

La suddivisione dei dati viene effettuata mantenendo l'ordine temporale degli eventi. Dopo aver ordinato i record in base al timestamp, il dataset viene suddiviso in tre parti: 80% per il training, 10% per la validation e 10% per il test. La prima porzione viene utilizzata per l'aggiornamento dei pesi della rete, mentre la validation viene fornita al modello durante l'addestramento attraverso il parametro *validation_ data*. Quest'ultima non viene quindi utilizzata per modificare direttamente i pesi, ma per monitorare l'andamento delle prestazioni su dati non impiegati nell'aggiornamento e supportare i meccanismi di gestione del training descritti nella sezione successiva. La porzione di test rimane invece separata e viene utilizzata soltanto al termine dell'addestramento, attraverso la funzione *model.evaluate*, per ottenere la valutazione finale del modello.
La scelta di effettuare la suddivisione in ordine cronologico permette di mantenere una distinzione temporale tra le diverse fasi del processo. In particolare, il modello non viene addestrato mescolando casualmente eventi appartenenti a periodi differenti, ma apprende dai dati più vecchi e viene successivamente verificato su eventi collocati temporalmente dopo quelli utilizzati per il training.
Questa impostazione risulta coerente con il funzionamento previsto della pipeline, nella quale il detector deve analizzare eventi successivi rispetto a quelli osservati durante l'apprendimento.

Le percentuali indicate vengono applicate agli eventi prima della costruzione delle sequenze. Poiché la generazione delle sequenze tiene conto dei vincoli definiti dal sequence builder, come il numero minimo di eventi e la finestra temporale considerata, il numero finale di sequenze nei tre insiemi può non coincidere esattamente con le proporzioni iniziali. La suddivisione temporale viene quindi mantenuta anche durante la trasformazione degli eventi in sequenze. Anche la fase di normalizzazione viene eseguita nel rispetto della separazione tra i tre insiemi.

Lo *StandardScaler* viene adattato esclusivamente sulle feature numeriche del training set e successivamente applicato, senza un nuovo adattamento, alla validation e al test. In questo modo le informazioni statistiche provenienti dai dati successivi non vengono utilizzate durante la fase di apprendimento.

4.3.3 Callbacks e gestione dell'addestramento

Per gestire in modo automatico il processo di addestramento della BiLSTM, vengono utilizzati tre meccanismi distinti, delle callback messe a disposizione da Keras. Il loro impiego permette di controllare l'evoluzione del training e di evitare che l'addestramento prosegua in condizioni non più vantaggiose per il modello.

- ***EarlyStopping***: monitora il valore di validation AUC e consente di interrompere automaticamente il training quando le prestazioni monitorate non mostrano ulteriori miglioramenti. Sono state impostate 8 epoche, per cui se non ci sono miglioramenti dopo questo numero di epoche, l'addestramento termina. In questo modo si evita di continuare l'addestramento per un numero di epoche non necessario, riducendo anche il rischio di overfitting. L'interruzione non comporta quindi semplicemente una riduzione del numero massimo di epoche previste, ma permette al processo di terminare in funzione dell'andamento effettivo dell'apprendimento. È inoltre configurato per ripristinare i pesi corrispondenti alla migliore prestazione raggiunta durante l'addestramento.

- ***ReduceLROnPlateau***: monitora invece la validation loss, ed è impiegata per modificare dinamicamente il learning rate durante il training. Quando le prestazioni del modello raggiungono una fase di stallo, quindi la loss non migliora per quattro epoche consecutive, il learning rate viene ridotto della metà, consentendo di effettuare aggiornamenti dei pesi più contenuti e di proseguire l'ottimizzazione in modo più graduale. Questa strategia è utile soprattutto nelle fasi avanzate dell'addestramento, quando variazioni troppo ampie dei pesi possono impedire ulteriori miglioramenti.

- ***ModelCheckpoint***: permette di salvare automaticamente il modello durante il training, mantenendo la configurazione associata alle migliori prestazioni raggiunte. Viene salvato quello corrispondente alla migliore *validation AUC* osservata durante l'addestramento. Il modello risultante dalla fase di addestramento può quindi essere successivamente riutilizzato per la fase di inferenza senza dover necessariamente utilizzare l'ultima configurazione ottenuta al termine delle epoche.

Nel loro insieme, questi meccanismi permettono di rendere più controllata la fase di addestramento della BiLSTM e costituiscono il collegamento tra la configurazione del modello descritta nelle sezioni precedenti e il modello effettivamente salvato e utilizzato successivamente nella pipeline.

4.3.4 Metriche di valutazione

Per valutare le prestazioni del classificatore sono state considerate diverse metriche, selezionate in modo da analizzare sia la capacità complessiva di classificazione sia il comportamento del modello rispetto alla classe sospetta. La sola accuratezza, infatti, non è sufficiente a descrivere completamente le prestazioni di un classificatore binario applicato a eventi di sicurezza, poiché è importante distinguere anche tra falsi positivi e falsi negativi e valutare la capacità del modello di identificare correttamente gli eventi sospetti.

La classificazione viene effettuata a partire dalla probabilità (p) restituita dal livello di output della rete. Nel processo di valutazione, tale probabilità viene trasformata in una classe binaria mediante una soglia pari a 0.50:

$$\hat{y} = \begin{cases} 1 & \text{se } p \geq 0.5 \\ 0 & \text{se } p < 0.5 \end{cases}$$

dove $\hat{y}$ rappresenta la classe predetta. Quindi valori inferiori alla soglia vengono associati alla classe **NORMAL**, mentre valori maggiori o uguali a 0.50 vengono associati alla classe **ATTACK**. È comunque importante mantenere il valore probabilistico prodotto dalla rete, anziché considerare esclusivamente la classificazione binaria.

La prima metrica considerata è l'**accuracy**, che misura la proporzione di predizioni corrette rispetto al numero complessivo di campioni.

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

dove TP e TN indicano rispettivamente i veri positivi e i veri negativi, mentre FP e FN rappresentano i falsi positivi e i falsi negativi.
L'Accuracy fornisce quindi una misura generale della correttezza delle classificazioni effettuate dal modello.

La **Precision** misura invece, tra tutti gli eventi classificati come sospetti, quanti risultano effettivamente appartenenti alla classe positiva:

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

Questa metrica è particolarmente utile per valutare la presenza di falsi allarmi, poiché una diminuzione della precisione corrisponde a un aumento della frazione di eventi normali classificati erroneamente come sospetti.

La **Recall**, o sensibilità, misura la capacità del modello di individuare gli eventi realmente appartenenti alla classe positiva:

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

In questo caso l'attenzione è quindi rivolta ai falsi negativi, ovvero agli eventi sospetti che non vengono riconosciuti dal classificatore.

L' **F1-score** unisce precision e recall in un unico valore, ideale quando i dati non sono bilanciati, cioè quando una classe domina nettamente sull'altra.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

A supporto di queste metriche viene analizzata la matrice di confusione, che rappresenta il numero di campioni distribuiti nelle quattro possibili combinazioni tra classe reale e classe predetta:

$$\begin{bmatrix} TN & FP \\ FN & TP \end{bmatrix}$$

La matrice consente di osservare direttamente la distribuzione degli errori del modello e di distinguere tra falsi positivi e falsi negativi, fornendo quindi un'informazione complementare rispetto ai valori precedenti.

Infine, viene utilizzata la ***ROC-AUC*** (*Receiver Operating Characteristic – Area Under the Curve*). A differenza delle metriche calcolate sulla classificazione binaria ottenuta con soglia (0.5), la ROC-AUC considera le probabilità prodotte dal modello e valuta la capacità complessiva del classificatore di separare le due classi al variare della soglia decisionale. Un valore di AUC maggiore indica una migliore capacità discriminativa tra eventi normali e sospetti.

Le metriche descritte vengono utilizzate congiuntamente per fornire una valutazione completa del comportamento della rete. I relativi valori ottenuti durante gli esperimenti e l'analisi della matrice di confusione vengono riportati nel capitolo 5, dedicato ai risultati sperimentali.

4.4 Progettazione e implementazione DQN

La seconda componente dell'architettura proposta è costituita da un agente di Deep Reinforcement Learning basato su Deep Q-Network (DQN). Il suo compito è determinare, a partire dalle informazioni prodotte dalla fase di detection e dal contesto associato all'evento, quale azione di risposta applicare.
A differenza della BiLSTM, che si occupa della rilevazione e produce una probabilità di attacco, la DQN è responsabile della fase decisionale. A tale scopo, le informazioni disponibili vengono organizzate in uno stato dell'ambiente, sul quale l'agente valuta le possibili azioni attraverso la funzione Q appresa durante l'addestramento.

4.4.1 Definizione dello stato RL

Una volta ottenuta la probabilità di attacco dalla BiLSTM, il risultato viene utilizzato per costruire lo stato osservato dall'agente di Reinforcement Learning. L'obiettivo di questa fase è fornire al DQN una rappresentazione più completa della situazione corrente, non limitata alla sola predizione del detector, ma arricchita con informazioni relative al comportamento osservato e al contesto della richiesta, come quelle già viste proprio nella fase di classificazione.
Lo stato è rappresentato tramite un vettore composto da nove variabili, selezionate con l'obiettivo di combinare l'informazione prodotta dal modello di detection con indicatori temporali, comportamentali e di sicurezza. In questo modo il DQN può associare situazioni differenti a possibili risposte differenti, sulla base non soltanto della probabilità di attacco, ma anche del contesto nel quale l'evento si verifica.
La prima componente di questo stato è proprio ***attack_probability***, ottenuta direttamente dall'output della BiLSTM, compresa tra 0 e 1, rappresenta la stima del modello sulla probabilità che il comportamento osservato appartenga alla classe di attacco. A differenza della semplice etichetta binaria ottenuta applicando una soglia, la probabilità mantiene l'informazione relativa al livello di confidenza della predizione e costituisce quindi l'elemento principale che collega il modulo di detection a quello decisionale. La seconda componente è ***failure_ratio***, che rappresenta la proporzione di eventi falliti all'interno della sequenza considerata. Viene calcolata come rapporto tra il numero di eventi con esito negativo e il numero complessivo di eventi della sequenza. In questo modo non viene utilizzato semplicemente il numero assoluto di errori, ma la loro incidenza rispetto alla dimensione della sequenza. La terza variabile è ***requests_5m***, rappresentante il numero di richieste osservate nella finestra di cinque minuti associata all'evento corrente. Successivamente si trovano le componenti ***hour*** e ***is_night*** per il contesto temporale, permettono al DQN di considerare il momento in cui viene effettuata la richiesta.

Le successive tre variabili descrivono gli aspetti direttamente collegati al contesto di sicurezza. ***Privileged*** per indicare se l'utente associato alla richiesta possiede un ruolo privilegiato, ***critical_ resource*** segnala se la richiesta riguarda una risorsa classificata come critica e ***has_ certificate*** indica invece la presenza di un certificato client associato all'evento. Queste informazioni permettono di distinguere situazioni caratterizzate da differenti livelli di esposizione o criticità, senza considerarle singolarmente come una prova automatica di comportamento malevolo. L'ultima componente è ***historical_ risk*** e rappresenta il rischio storico associato all'entità osservata. Il valore deriva dalla memoria comportamentale costruita durante il feature engineering e tiene conto degli eventi precedenti, con un meccanismo di decadimento nel tempo. In questo modo lo stato del DQN può incorporare anche una componente relativa al comportamento passato dell'entità, oltre alle informazioni disponibili nell'evento e nella sequenza corrente.
È inoltre importante distinguere l'origine delle diverse componenti dello stato. La *attack_ probability* viene prodotta come detto direttamente dalla BiLSTM, mentre la maggior parte delle altre variabili viene ricavata dalle informazioni relative all'ultimo evento della sequenza. Il *failure_ ratio* costituisce un'eccezione, poiché viene calcolato considerando l'intera sequenza osservata. In questo modo lo stato combina una valutazione predittiva del comportamento con informazioni puntuali e aggregate.

Nel complesso, il vettore di stato può quindi essere interpretato come una descrizione sintetica della situazione corrente secondo tre dimensioni principali: rischio stimato dal detector, comportamento osservato e contesto di sicurezza. Tale rappresentazione viene successivamente fornita al DQN, che sulla base dello stato e della politica appresa determina l'azione da applicare. La fase successiva riguarda la definizione dell'ambiente nel quale queste osservazioni vengono utilizzate e nel quale l'agente può scegliere tra le diverse contromisure disponibili.

4.4.2 Normalizzazione dello stato

Prima di essere fornito alla rete DQN lo stato ha bisogno di una fase di normalizzazione, cosi come successo nella BiLSTM, finalizzata a rendere più omogenea la scala delle diverse componenti. Questa operazione è particolarmente importante perché il vettore contiene variabili con domini differenti, alcune sono già espresse nell'intervallo `[0, 1]`, per costruzione oppure perché rappresentate tramite valori binari, mentre altre, come *requests_ 5m* e *hour*, assumono valori numerici più elevati. Riguardo ad *hour*, rappresentando l'ora del giorno e assumendo valori tra 0 e 23, la normalizzazione viene applicata rispetto al valore massimo.
Per la variabile *requests_ 5m* invece c'è stato bisogno di una trasformazione specifica, poiché rappresenta un conteggio e può assumere valori sensibilmente superiori rispetto alle altre componenti dello stato.

Viene utilizzato un valore determinato a partire dalle statistiche osservate sulla distribuzione delle richieste nei 5 minuti nel training set, corrispondente al 90° percentile. Questa scelta permette di ridurre l'influenza dei valori particolarmente elevati senza utilizzare direttamente il massimo osservato nella distribuzione. La trasformazione complessiva permette quindi di fornire alla DQN un vettore composto da variabili rappresentate su una scala omogenea. È importante distinguere questa operazione dallo *StandardScaler* utilizzato nella fase precedente per le feature numeriche della BiLSTM, in quanto, in questo caso, non viene effettuata una standardizzazione basata su media e deviazione standard, ma una normalizzazione progettata specificamente per le variabili che costituiscono lo stato dell'agente

4.4.3 Progettazione e implementazione dell'ambiente RL

Per permettere all'agente di apprendere una politica di risposta è stato necessario definire un ambiente di RL dove poter simulare differenti condizioni di sicurezza e valutare gli effetti delle azioni disponibili. Questo ambiente è stato realizzato in forma sintetica, con l'obiettivo di riprodurre differenti situazioni caratterizzate da livelli di rischio e comportamenti differenti. In questo contesto l'agente osserva uno stato, seleziona un'azione e riceve una ricompensa in funzione della scelta effettuata. Il ciclo di interazione è :

$$\textit{stato} \rightarrow \textit{azione} \rightarrow \textit{ricompensa} \rightarrow \textit{nuovo stato}$$

Lo stato corrisponde al vettore a nove dimensioni definito nella sezione precedente, contenente la probabilità di attacco prodotta dalla BiLSTM e le informazioni comportamentali e contestuali associate all'evento. A partire da tale osservazione, l'agente DQN sceglie una delle contromisure disponibili. L'ambiente valuta quindi l'azione sulla base dello scenario corrente e restituisce una ricompensa, fornendo al modello il segnale necessario per aggiornare progressivamente la politica appresa.
Per rendere l'addestramento differenziato rispetto al tipo di situazione osservata, l'ambiente genera diversi scenari comportamentali. Tra quelli considerati rientrano condizioni di traffico normale (NORMAL), attacchi di tipo brute force (BRUTE_FORCE), comportamento automatizzato o riconducibile a bot (BOT), attività di scansione delle risorse (RESOURCE_SCAN), attacchi associati a utenti privilegiati (PRIVILEGED_ATTACK) e situazioni intermedie di attività sospetta (SUSPICIOUS_ACTIVITY). Ogni scenario viene associato a differenti distribuzioni dei valori che compongono lo stato, così da rappresentare condizioni di rischio non equivalenti. Questi valori vengono generati in modo randomico tenendo però in considerazione l'intervallo numerico nel quale ogni stato può trovarsi, anche sulla base dei valori riscontrati nella fase di analisi del dataset.

La presenza di scenari differenti è importante perché l'obiettivo dell'apprendimento non è associare una singola azione a una determinata classe, ma apprendere una relazione tra stato osservato, azione e conseguenza dell'azione. Uno stato caratterizzato da una minaccia elevata può quindi richiedere una risposta differente rispetto a uno stato con indicatori più deboli o con caratteristiche prevalentemente normali.
L'ambiente tiene inoltre conto dell'impatto delle diverse contromisure. Le azioni non hanno infatti tutte lo stesso costo operativo, ma interventi più restrittivi comportano un costo maggiore rispetto a semplici operazioni di monitoraggio o di autorizzazione. La funzione di *reward* utilizza questa differenza per introdurre nel processo di apprendimento un compromesso tra la necessità di ridurre la minaccia e l'impatto prodotto dalla risposta scelta. Nelle due sezioni successive vengono approfondite proprio le definizioni delle azioni e della funzione di reward.
Un elemento rilevante di questa progettazione è la natura sintetica dell'ambiente. Gli scenari e le relative transizioni sono stati costruiti per fornire al DQN un contesto controllato nel quale apprendere la politica di risposta e non derivano dall'interazione diretta con una rete reale. Di conseguenza, l'ambiente costituisce una rappresentazione sperimentale del problema e non una riproduzione completa di tutte le condizioni che possono verificarsi in un'infrastruttura reale.

4.4.3.1 Spazio delle azioni

Lo spazio delle azioni definisce le possibili contromisure che l'agente DQN può selezionare e applicare in risposta allo stato osservato. Nell'ambiente sviluppato, lo spazio delle azioni è discreto ed è costituito da sei possibili azioni, ciascuna associata a un differente livello crescente di restrizione nei confronti dell'entità considerata. Questa scelta consente al DQN di formulare una decisione di difesa tra un insieme finito di contromisure, anziché limitarsi a determinare esclusivamente se una richiesta debba essere consentita o bloccata. Le varie azioni disponibili sono:

- **ALLOW**: consente la richiesta senza applicare nessuna contromisura restrittiva;
- **MONITOR**: consente la prosecuzione della sessione mantenendola tuttavia sotto osservazione;
- **MFA**: richiede un'ulteriore verifica dell'identità dell'utente attraverso un'autenticazione multifattore;
- **BLOCK_SESSION**: blocca la sessione associata all'attività, impedendo la prosecuzione delle richieste associate alla sessione;
- **BLOCK_IP**: blocca l'indirizzo IP associato alla richiesta;
- **QUARANTINE**: applica la contromisura più restrittiva prevista dall'ambiente, portando il livello di attività dell'entità verso una condizione fortemente limitata.

Le azioni rappresentano quindi differenti livelli di risposta al rischio. ALLOW costituisce l'intervento meno invasivo, mentre le azioni di blocco e quarantena introducono conseguenze progressivamente più rilevanti sull'attività dell'entità. MONITOR rappresenta invece una risposta intermedia che permette di mantenere l'attività sotto osservazione senza interromperla, mentre MFA introduce un ulteriore controllo sull'identità.
Le sei azioni sono rappresentate internamente mediante valori interi compresi tra 0 e 5 e vengono definite attraverso uno spazio Discrete(6) di Gymnasium. La rappresentazione numerica non esprime quindi un ordinamento quantitativo delle azioni, ma costituisce semplicemente l'identificativo utilizzato dall'algoritmo DQN per distinguere le diverse decisioni disponibili. Questa articolazione permette di evitare una decisione puramente binaria tra autorizzazione e blocco. La progettazione dello spazio delle azioni tiene conto del fatto che una risposta di sicurezza non deve necessariamente essere immediatamente e completamente restrittiva. In presenza di un'attività a basso rischio, l'agente può infatti scegliere di consentire la richiesta oppure di monitorarla, mentre, al crescere del rischio sono invece disponibili contromisure progressivamente più incisive.
Un aspetto importante dell'ambiente è che l'effetto di un'azione non è determinato esclusivamente dall'azione stessa, ma anche dallo scenario nel quale viene applicata. Questo permette di rappresentare situazioni nelle quali una stessa contromisura può produrre conseguenze differenti. Ad esempio, consentire una richiesta in presenza di un attacco brute-force determina un incremento della probabilità di attacco, del rapporto di richieste fallite e del volume di richieste, mentre l'applicazione di un blocco riduce tali indicatori.
L'azione viene quindi applicata allo stato corrente attraverso una funzione di transizione dell'ambiente. In funzione dello scenario, tale transizione modifica principalmente la probabilità di attacco, il rapporto di richieste fallite e il numero di richieste osservate nella finestra temporale considerata. Successivamente viene aggiornato anche il rischio storico dell'entità. Questo meccanismo permette di simulare la conseguenza delle decisioni dell'agente e costituisce il collegamento fondamentale tra una scelta effettuata al tempo corrente e le condizioni che l'agente incontrerà negli step successivi.
L'ambiente non tratta inoltre tutti gli episodi nello stesso modo. Gli scenari normali hanno una durata massima inferiore, mentre gli episodi di attacco possono terminare anticipatamente quando l'attività malevola viene considerata neutralizzata, ossia quando la probabilità di attacco scende al di sotto di una determinata soglia. In ogni caso viene imposto un limite massimo di cinque step per episodio. Questa struttura permette di rappresentare una sequenza di decisioni successive e non una singola scelta isolata, caratteristica necessaria per formulare il problema come processo decisionale sequenziale. Lo spazio delle azioni è quindi progettato per permettere al DQN di apprendere non soltanto quale azione sia associata a un determinato livello di rischio, ma quale intervento risulti più appropriato considerando anche l'evoluzione attesa della sessione.

L'obiettivo dell'agente è pertanto individuare una politica capace di bilanciare la necessità di neutralizzare l'attività sospetta con quella di evitare contromisure eccessivamente restrittive nei confronti di attività legittime.

4.4.3.2 Funzione di reward

La funzione di reward definisce il criterio attraverso il quale l'agente valuta l'efficacia delle azioni intraprese all'interno dell'ambiente. A differenza dell'approccio supervisionato utilizzato per la componente BiLSTM, nel quale il modello apprende la distinzione tra eventi normali e sospetti a partire dalle etichette disponibili, il DQN deve apprendere quale contromisura applicare sulla base delle conseguenze prodotte dalle proprie decisioni. La funzione di reward traduce quindi tali conseguenze in un segnale numerico che permette all'agente di rafforzare le strategie più appropriate e di penalizzare quelle che producono risultati indesiderati. Nel sistema proposto, questa progettazione mira a rappresentare il compromesso tra riduzione della minaccia e costo operativo della contromisura, ed è orientata a bilanciare due esigenze: da un lato, la capacità di contrastare efficacemente le attività malevole e, dall'altro, la necessità di limitare interventi eccessivamente restrittivi nei confronti delle attività legittime. L'obiettivo non è quindi associare semplicemente una ricompensa positiva alle azioni più restrittive, ma permettere all'agente di apprendere quale risposta sia maggiormente adeguata rispetto allo stato osservato. Il reward viene calcolato a ogni transizione confrontando lo stato precedente con quello risultante dall'applicazione dell'azione. Durante l'esecuzione di uno step vengono infatti conservate una copia dello stato precedente, viene applicata l'azione scelta dal DQN e viene successivamente calcolata la ricompensa sulla base dello stato risultante, dell'azione eseguita e dello scenario corrente. In questo modo il reward non dipende solamente dalla tipologia di azione, ma anche dall'effetto che tale azione ha avuto sull'evoluzione della situazione.
Per questo motivo a ciascuna azione viene associato un costo operativo crescente in funzione del livello di invasività dell'intervento.

Azione	**Costo**
ALLOW	0
MONITOR	1
MFA	4
BLOCK_SESSION	12
BLOCK_IP	20
QUARANTINE	30

Tabella 4.2: Costi associati alle azioni di sicurezza

Questi valori rappresentano una modellazione sperimentale dell'impatto delle diverse contromisure e non devono essere interpretati come costi misurati direttamente in un ambiente reale.

Come si può vedere l'azione ALLOW rappresenta l'intervento meno restrittivo, mentre MONITOR introduce un costo limitato associato all'attività di controllo. MFA comporta un costo maggiore, poiché richiede un'ulteriore verifica dell'identità dell'utente. Le azioni di blocco e quarantena invece hanno invece costi progressivamente più elevati, poiché impediscono in misura crescente il proseguo dell'attività. Questa differenziazione permette di introdurre nel processo di apprendimento il principio secondo cui una contromisura più restrittiva deve essere giustificata da un livello di rischio sufficientemente elevato. Di conseguenza, in uno scenario normale, una politica che ricorra sistematicamente a blocchi o quarantene riceve un segnale meno favorevole rispetto a una politica che mantenga l'accesso quando le condizioni non evidenziano una minaccia significativa. Al contrario, durante un attacco, il costo associato a una contromisura deve essere valutato insieme al beneficio derivante dalla riduzione del rischio e dalla neutralizzazione dell'attività malevola.
Un altro elemento considerato è la severità della situazione. Un'attività caratterizzata da elevata probabilità di attacco, numerose richieste fallite, elevata pressione delle richieste o presenza di condizioni particolarmente sensibili deve produrre ovviamente un incentivo maggiore verso una risposta di contenimento rispetto a una situazione nella quale gli stessi indicatori presentano valori ridotti. In questo modo il reward contribuisce a evitare una corrispondenza rigida tra determinati valori dello stato e una singola azione, lasciando all'agente la possibilità di apprendere una strategia condizionata dal contesto.

La funzione è implementata attraverso una classe dedicata, *RewardCalculator*, che riceve in ingresso lo stato precedente, l'azione selezionata, lo stato successivo e lo scenario corrente. La ricompensa viene quindi calcolata al termine di ogni transizione dell'ambiente, tenendo conto sia della decisione dell'agente sia degli effetti che questa ha prodotto. In particolare, la logica adottata distingue due condizioni, ovvero, gli scenari normali nei quali è nota l'assenza di un attacco, e gli scenari di attacco, nei quali la ricompensa è determinata dall'efficacia della contromisura. Per gli scenari normali viene adottata una ricompensa specifica, assegnata sulla base di una tabella di valori predefiniti, finalizzata a favorire il mantenimento del normale funzionamento del sistema evitando risposte eccessivamente invasive. In questo caso, non viene valutata la riduzione della minaccia, poiché la simulazione rappresenta un'attività legittima e non esiste un attacco da neutralizzare.
Come si può notare dalla tabella 4.3, questa configurazione favorisce il mantenimento della continuità operativa, attribuendo la ricompensa maggiore all'azione ALLOW, seguita da MONITOR. Al contrario, le azioni che richiedono verifiche aggiuntive o che interrompono l'attività comportano ricompense negative, proporzionali al livello di restrizione che comportano, incentivando quindi a non applicare contromisure drastiche quando il contesto non presenta evidenze sufficienti di un attacco.

L'obiettivo è evitare che l'agente sviluppi una strategia troppo conservativa, nella quale il blocco sistematico delle sessioni venga preferito indipendentemente dal contesto. La presenza di penalità elevate per le azioni più invasive consente quindi di rappresentare il costo dei falsi positivi e di incentivare decisioni proporzionate quando l'attività osservata è legittima.

Azione	**Reward**
ALLOW	+20
MONITOR	+8
MFA	-10
BLOCK_SESSION	-60
BLOCK_IP	-90
QUARANTINE	-120

Tabella 4.3: Reward associati alle azioni nello scenario NORMAL

Negli scenari di attacco, la ricompensa non è definita attraverso una tabella statica, ma viene calcolata dinamicamente in funzione dell'evoluzione della minaccia. A questo scopo, viene introdotto un indicatore sintetico, denominato ***threat score***, che riassume le principali informazioni relative al livello di rischio della sessione.
Il punteggio viene calcolato a partire da cinque variabili dello stato: la probabilità di attacco stimata, il rapporto di richieste fallite, l'intensità delle richieste, la criticità della risorsa interessata e il rischio storico dell'entità. Ciascuna variabile viene ricondotta all'intervallo [0,1] e combinata mediante una somma pesata:

$$\begin{aligned}\text{ThreatScore} = {} & 0.35 \cdot \text{attack_probability} \\ & + 0.25 \cdot \text{failure_ratio} \\ & + 0.15 \cdot \text{request_intensity} \\ & + 0.15 \cdot \text{critical_resource} \\ & + 0.10 \cdot \text{historical_risk}\end{aligned}$$

Il risultato è un valore compreso tra zero e uno, che costituisce una misura sintetica della minaccia e permette di confrontare lo stato precedente e quello successivo all'applicazione di una contromisura. Una volta determinato il *threat score*, la ricompensa viene quindi determinata considerando principalmente la variazione della minaccia prima e dopo l'applicazione dell'azione.
La funzione di reward negli scenari di attacco è costruita attraverso più componenti:

$$
\begin{aligned}
R = & 120\Delta T \\
& - 20T(s_{t+1}) \\
& - C(a) \\
& - 25I(a = \text{ALLOW}) \\
& + 20I(P_{\text{attack},t+1} \leq 0.05) \\
& + 40\Delta T I(\Delta T < 0)
\end{aligned}
$$

Dove C(a) rappresenta il costo operativo dell'azione, mentre I(·) è una funzione indicatrice che assume valore uno quando la condizione indicata è verificata e zero altrimenti. La prima componente, 120ΔT, premia la riduzione della minaccia e penalizza il suo incremento. Il coefficiente moltiplicativo determina l'entità della ricompensa associata all'efficacia dell'intervento: ad esempio, una riduzione di 0,50 del threat score produce un contributo positivo pari a 60 punti. Alla variazione del rischio si affianca una penalizzazione proporzionale alla minaccia residua, attraverso il termine -20T(s_{t+1}). Questa componente incentiva l'agente non soltanto a migliorare le condizioni rispetto allo stato iniziale, ma anche a raggiungere uno stato successivo caratterizzato da un livello di rischio contenuto. In questo modo, due azioni che producono la stessa riduzione della minaccia possono ricevere ricompense differenti qualora conducano a livelli di rischio residuo diversi. Una terza componente riguarda il costo operativo della contromisura. A ciascuna delle sei azioni è associato un costo crescente in funzione del grado di invasività: da zero per ALLOW fino a 30 per QUARANTINE. Tale costo viene sottratto dalla ricompensa e introduce un criterio di proporzionalità nella selezione delle contromisure. Anche quando due azioni producono una riduzione analoga della minaccia, quella caratterizzata da un impatto operativo inferiore risulta maggiormente favorita. È inoltre prevista una penalizzazione aggiuntiva di 25 punti quando l'agente seleziona ALLOW durante uno scenario di attacco. Questa componente mira a scoraggiare il mantenimento dell'accesso in presenza di un'attività malevola, anche nei casi in cui la variazione degli indicatori di rischio risulti limitata. Per incentivare il completamento dell'intervento di sicurezza, viene infine introdotto un bonus di 20 punti quando la probabilità di attacco nello stato successivo raggiunge un valore inferiore o uguale a 0,05. Il bonus premia quindi il raggiungimento di una condizione di sostanziale neutralizzazione dell'attacco, indipendentemente dalla sola variazione relativa del threat score. L'ultima componente prevede una penalizzazione supplementare qualora il livello complessivo di minaccia aumenti. Quando ΔT è negativo, viene aggiunto alla ricompensa il termine 40ΔT, che accentua il segnale negativo già prodotto dalla componente di efficacia. In questo modo, le azioni che determinano un peggioramento delle condizioni di sicurezza vengono ulteriormente scoraggiate.

La combinazione delle diverse componenti permette di definire una funzione di ricompensa che considera contemporaneamente l'efficacia delle contromisure, il rischio residuo e il costo operativo degli interventi. La distinzione tra scenari normali e scenari di attacco è particolarmente rilevante, poiché consente di differenziare il comportamento desiderato in base alle condizioni simulate: negli episodi normali viene privilegiata la continuità del servizio, mentre negli episodi di attacco la priorità diventa la riduzione della minaccia, senza trascurare l'impatto delle azioni adottate.
Durante l'addestramento, il DQN utilizza il segnale di reward ricevuto a ogni transizione per aggiornare la stima del valore delle azioni nei diversi stati. Attraverso l'esperienza accumulata, l'agente può così apprendere quali contromisure risultano più vantaggiose nelle differenti condizioni, considerando non soltanto il risultato immediato dell'intervento, ma anche le conseguenze delle decisioni successive.
La funzione di reward costituisce pertanto il principale elemento di collegamento tra la simulazione dell'ambiente e l'apprendimento della politica di difesa.

4.4.4 Architettura della rete DQN

Una volta definito e normalizzato lo stato dell'ambiente e stabilite le modalità di interazione con il sistema, è stata implementata la rete neurale utilizzata dall'agente DQN per approssimare la funzione di valore (Q).
Come già descritto nella sezione 2.4, la DQN sostituisce la rappresentazione tabellare del Q-learning con un modello neurale in grado di stimare i valori associati alle diverse azioni a partire dallo stato osservato. Nel presente lavoro la rete è quindi progettata per ricevere in ingresso il vettore di stato definito nelle sezioni precedenti e restituire, in un'unica elaborazione, un valore (Q) per ciascuna delle sei azioni disponibili.
La rete è costituita da un livello di ingresso seguito da tre livelli completamente connessi *Dense* e da un livello di uscita. L'input presenta una dimensionalità pari a nove, corrispondente alle nove variabili che compongono lo stato dell'agente e i tre livelli nascosti sono costituiti rispettivamente da 128, 64 e 32 neuroni, tutti caratterizzati da una funzione di attivazione ReLU. La progressiva riduzione del numero di neuroni permette di trasformare gradualmente la rappresentazione dello stato in una forma più compatta prima della stima finale dei valori Q. L'ultimo livello è costituito da sei neuroni, uno per ogni possibile azione dell'ambiente. A differenza dei livelli nascosti, il livello di uscita utilizza una funzione di attivazione lineare. I sei valori prodotti dalla rete non rappresentano quindi probabilità, ma le stime dei rispettivi valori (Q(s,a)). Per uno stato (s), l'output può essere rappresentato come: $Q(s)=[Q(s,a_0),Q(s,a_1),...,Q(s,a_5)$ dove ogni elemento del vettore corrisponde a una delle sei contromisure definite nell'ambiente. La scelta dell'azione finale viene quindi effettuata confrontando i valori ottenuti e scegliendo l'azione associata al valore Q più elevato $a^* = \arg\max_a Q(s, a)$.

L'architettura complessiva della rete è pertanto
Input(9) → Dense(128, ReLU) → Dense(64, ReLU) → Dense(32, ReLU) → Dense(6, linear)

Un elemento caratteristico dell'implementazione è la presenza di due reti con la stessa architettura, denominate rispettivamente *online network* e *target network*. La prima costituisce la rete principale dell'agente e viene utilizzata per apprendere e aggiornare i valori Q durante l'addestramento. La seconda mantiene una copia dei parametri della rete principale e viene invece utilizzata per calcolare il valore del prossimo stato durante la costruzione del target di apprendimento. La separazione tra le due reti riduce la dipendenza tra i valori utilizzati per la stima e quelli aggiornati durante l'apprendimento, contribuendo a rendere più stabile il processo di training. Nel modello sviluppato, la target network viene inizializzata copiando i pesi della rete principale, successivamente, durante l'addestramento, i suoi parametri non vengono modificati a ogni aggiornamento ma vengono riallineati periodicamente con quelli della rete principale.

4.4.5 Configurazione e addestramento del DQN

L'addestramento del DQN viene effettuato mediante interazione ripetuta dell'agente con l'ambiente di RL. Ad ogni iterazione l'agente osserva lo stato corrente, seleziona un'azione, riceve una ricompensa e procede verso un nuovo stato. Attraverso la ripetizione di queste interazioni, la rete apprende a stimare i valori (Q(s,a)) associati alle diverse azioni disponibili. Nel progetto, il processo di addestramento finale è stato eseguito per 1000 episodi, utilizzando un batch size pari a 64. Un episodio del DQN rappresenta una simulazione dell'interazione tra agente e ambiente. La target network, che viene utilizzata per rendere più stabile l'apprendimento, viene aggiornata attraverso due meccanismi, dopo ogni 100 operazioni di training effettuate dall'agente e ogni 20 episodi.
Per ridurre la correlazione tra esperienze consecutive, durante ogni aggiornamento viene estratto casualmente dal replay buffer, ovvero dove vengono memorizzate le esperienze raccolte, un insieme di transizioni di dimensione pari al batch size. L'aggiornamento non viene quindi effettuato utilizzando esclusivamente l'esperienza immediatamente precedente, ma su un insieme di esperienze già raccolte. In questo modo le interazioni precedenti possono essere riutilizzate durante più fasi di addestramento. Per ogni stato appartenente al batch, la rete principale calcola inizialmente il vettore dei valori (Q) relativi alle sei azioni disponibili. Parallelamente, la target network viene utilizzata per stimare i valori Q associati agli stati successivi. Per l'azione effettivamente eseguita, il valore stimato dalla rete principale viene sostituito con il valore target calcolato in funzione della ricompensa ottenuta e della migliore azione stimata nello stato successivo.
Nel sistema sviluppato viene utilizzato un valore di (γ=0,95).

Questo parametro permette di attribuire rilevanza non soltanto alla ricompensa immediatamente ottenuta, ma anche alle conseguenze future delle decisioni dell'agente.
La rete viene addestrata utilizzando l'ottimizzatore ***Adam***, con un ***learning rate*** pari a **0,0003**. Come funzione di perdita viene utilizzata la ***Huber Loss***, particolarmente adatta all'apprendimento dei valori Q poiché combina il comportamento della perdita quadratica per errori ridotti con quello della perdita assoluta per errori più elevati, limitando quindi l'influenza di eventuali aggiornamenti caratterizzati da valori anomali. Per migliorare ulteriormente la stabilità numerica dell'ottimizzazione viene inoltre applicato il ***gradient clipping*** tramite un limite della norma del gradiente pari a 1.
Un ulteriore parametro fondamentale riguarda il compromesso tra esplorazione ed exploitation. La selezione delle azioni viene effettuata attraverso una politica ε-greedy. In particolare, all'inizio dell'addestramento il parametro viene impostato a 1,0, facendo sì che l'agente scelga inizialmente le azioni in modo casuale. Con il procedere dell'apprendimento, il valore viene progressivamente ridotto secondo un fattore di decadimento pari a 0,998, fino a raggiungere un valore minimo di 0,05. Il valore di ε determina quindi la probabilità con cui viene effettuata un'azione casuale. Quando l'esplorazione non viene selezionata, l'agente utilizza invece i valori Q prodotti dalla rete principale e sceglie l'azione corrispondente al valore maggiore. La strategia consente di quindi di iniziare l'addestramento esplorando in modo esteso lo spazio delle azioni e di passare progressivamente a un comportamento maggiormente basato sulle conoscenze acquisite.
Al termine del processo di addestramento, i pesi del DQN vengono salvati e successivamente utilizzati nella fase di inferenza. In questa fase l'esplorazione non è più necessaria: dato uno stato normalizzato, la rete produce i sei valori Q corrispondenti alle possibili contromisure e la scelta viene effettuata selezionando il valore massimo.

L'azione selezionata costituisce alla fine l'output operativo della componente decisionale e rappresenta la contromisura da applicare all'evento osservato. La separazione tra il risultato della BiLSTM e quello del DQN permette di mantenere distinti i due compiti, il primo modello stima il livello di rischio associato alla sequenza, mentre il secondo utilizza tale informazione all'interno di uno stato più ampio per determinare la risposta di sicurezza.

4.5 Pipeline completa

L'implementazione finale del sistema è organizzata in due modalità complementari, sviluppate per rispondere a due esigenze differenti. La prima, realizzata attraverso il modulo *run_pipeline.py*, è finalizzata alla valutazione sperimentale della pipeline sul dataset disponibile e permette di analizzare il comportamento congiunto della BiLSTM e del DQN. La seconda, realizzata attraverso *interactive_pipeline.py*, rappresenta la modalità operativa "online" del sistema e consente di analizzare nuovi eventi senza eseguire nuovamente l'addestramento dei modelli. In entrambe le modalità vengono utilizzati gli stessi artefatti precedentemente ottenuti durante la fase di training, costituiti dal modello BiLSTM, dal modello DQN, dallo scaler e dagli encoder. I modelli vengono quindi mantenuti congelati durante l'inferenza, garantendo che le decisioni siano prodotte a partire dalla stessa configurazione utilizzata nella fase sperimentale, senza eseguire ulteriori operazioni di fit o di aggiornamento dei pesi. La modalità offline permette di verificare quantitativamente il comportamento della soluzione sull'insieme di dati selezionato per il test. La modalità interattiva, invece, consente di mostrare il funzionamento della soluzione in uno scenario più vicino all'utilizzo operativo: un evento viene acquisito, associato alla propria entità, elaborato in funzione della storia disponibile e trasformato in una decisione di mitigazione. Verrà poi descritto nel dettaglio il funzionamento nelle prossime sezioni.

4.5.1 Flusso completo di elaborazione

Le prime fasi del flusso di elaborazione offline hanno il compito di trasformare i dati grezzi in una rappresentazione strutturata e adatta all'elaborazione da parte dei modelli, mentre le fasi successive utilizzano tali informazioni per effettuare prima la detection e successivamente la decisione di risposta. Il flusso complessivo può essere visto cosi: [4.1]
Raw Logs → Correlation → Flattening → Feature Engineering → Sequence Builder → BiLSTM → Attack Probability → RL State → DQN → Mitigation Action
Il processo parte dalla lettura del file di log e prosegue con la correlazione dei record, la creazione dei log flat, il caricamento degli encoder e il feature engineering. Le caratteristiche ottenute vengono successivamente utilizzate per costruire le sequenze temporali richieste dalla BiLSTM. Per la valutazione finale viene mantenuta la suddivisione temporale definita nella fase sperimentale. Quando è attiva la configurazione *EVALUATE_TEST_ONLY*, la pipeline utilizza esclusivamente la porzione di test, cioè gli eventi che non sono stati utilizzati per l'aggiornamento dei pesi della BiLSTM durante il training. Le sequenze ottenute vengono quindi fornite al modello precedentemente addestrato, che produce per ciascun caso una probabilità di attacco.

La probabilità restituita dalla BiLSTM viene successivamente integrata con le informazioni contestuali e trasformata nello stato a nove dimensioni utilizzato dal DQN. Lo stato viene normalizzato secondo gli stessi criteri adottati durante l'addestramento dell'agente e fornito alla rete Q. Il DQN produce un valore Q per ciascuna delle sei azioni disponibili e, durante questa fase di inferenza, viene selezionata direttamente l'azione associata al valore massimo.
La pipeline offline permette così di osservare non soltanto le prestazioni della BiLSTM, ma anche il comportamento del livello decisionale successivo. In particolare, viene costruito un confronto tra la classe reale dell'evento e l'azione selezionata dal DQN. Ai fini di questa analisi, ALLOW viene interpretata come decisione di non intervenire, mentre tutte le altre azioni rappresentano un intervento di mitigazione. Su questa base viene prodotto un *DQN Classification Report*, che verrà mostrato nel dettaglio nel capitolo 5 dei risultati. La stessa esecuzione consente inoltre di analizzare la distribuzione delle azioni DQN separatamente per eventi realmente normali e di attacco e di osservare come il livello decisionale si comporti nei diversi casi della matrice di confusione della BiLSTM. In questo modo è possibile valutare il ruolo del DQN non soltanto come classificatore derivato dall'azione scelta, ma come componente decisionale che determina la contromisura sulla base dello stato prodotto dal rilevatore.

La modalità interattiva finale implementa invece il funzionamento online del sistema. All'utente sono possibili due scelte iniziali, o inizializzare una sessione vuota, completamente da zero, oppure precaricare il file di log *logsprova.json* come storia iniziale per avere un passato da valutare. Una volta fatta la prima scelta, il menù che apparirà vedrà alcune voci tra cui poter inserire un nuovo evento, mostrare le entità conosciute e mostrare la cronologia delle richieste per un'entità. Quando si sceglie di visualizzare le entità conosciute, uscirà un elenco numerato con le diverse coppie user e ip con il relativo numero di eventi. Per la cronologia degli eventi di un'entità invece, si ha la possibilità di scelta dall'elenco delle entità, tramite il numero progressivo, e successivamente, una volta scelta, comparirà l'elenco dei corrispondenti eventi, ordinati per timestamp e riportanti l'esito di fallimento o meno della richiesta.
Gli eventi vengono organizzati per entità (user, IP) e mantenuti in memoria in ordine temporale. Quando viene aggiunto un nuovo evento attraverso l'inserimento manuale, questo viene salvato, cosi come la sua decisione. Sono stati infatti aggiunti due file *jsonl*, basati su righe dove ogni riga rappresenta un oggetto json, per l'esattezza *online_ decisions.jsonl* e *online_ events.jsonl*. I nuovi file aggiunti vengono quindi salvati nel file jsonl e sono poi successivamente "incorporati" quando all'inizio della pipeline, all'avvio, viene scelto di caricare la storia passata. Questi file persistiti vengono quindi aggiunti, senza però essere scritti nel file di log iniziale. Una volta ottenuta la decisione sull'azione da intraprendere, anche quest'ultima viene salvata, per mantenere uno storico delle decisioni prese che sia possibile consultare.

Per ogni analisi viene infatti generato un record di audit contenente il contesto osservato, la modalità utilizzata, i parametri rilevanti e, nel caso di esecuzione completa, la probabilità di attacco, lo stato RL e i valori Q del DQN. La decisione può quindi essere successivamente ricostruita a partire dalle informazioni registrate.

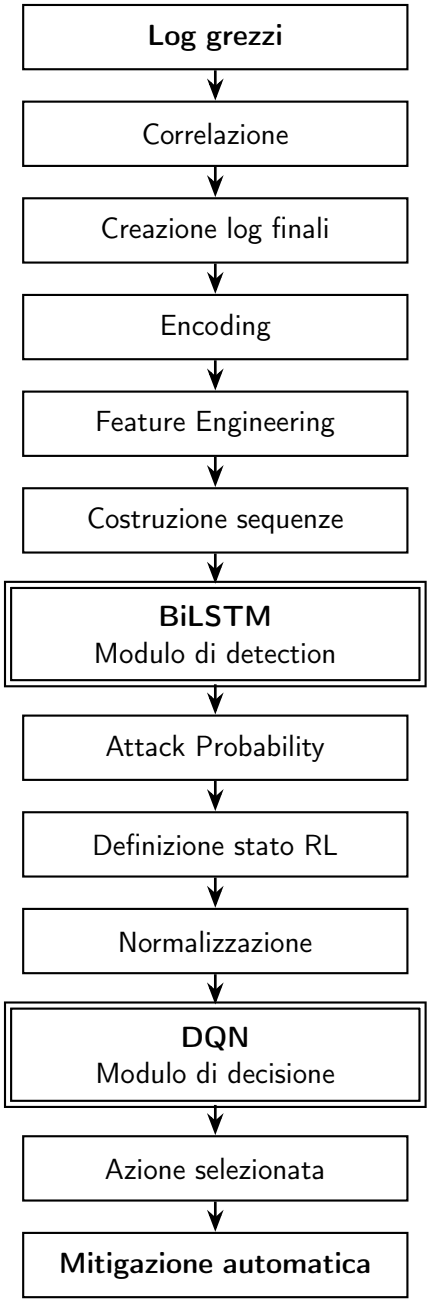


Figura 4.1: Diagramma di flusso del sistema di rilevamento e mitigazione

4.5.2 Gestione di un nuovo evento

Come detto, la modalità interattiva è stata progettata proprio per gestire nuovi eventi, richieste in tempo reale per le quali viene effettuata una decisione, tra permettere la richiesta e scegliere una contromisura. L'elaborazione prende avvio dall'inserimento di un singolo evento JSON. Il record viene validato e ricondotto a uno schema uniforme, indipendentemente dalla struttura originaria della sorgente. In questa fase vengono identificati principalmente timestamp, utente, indirizzo IP, tipo di evento, codice HTTP, ruolo, risorsa, certificato e ulteriori attributi di sicurezza necessari alla costruzione delle feature.
Il nuovo evento viene quindi associato alla coppia (user, IP) corrispondente e inserito nella relativa cronologia. La storia dell'entità viene mantenuta ordinata temporalmente e, per l'analisi del nuovo record, vengono utilizzati esclusivamente gli eventi che risultano disponibili fino al suo timestamp. Questa scelta permette di evitare l'utilizzo di informazioni future rispetto all'evento sotto analisi e mantiene il comportamento coerente con una simulazione temporale reale.
Successivamente viene eseguito il feature engineering sulla storia causale dell'entità. Per il modello vengono così ricostruite le quindici feature definite nel sistema. A questo punto viene individuata la finestra temporale di cinque minuti che termina con l'evento corrente. La finestra può contenere al massimo cinquanta eventi e deve includerne almeno cinque affinché sia considerata sufficiente per l'analisi della BiLSTM.
Quando tale condizione è soddisfatta, viene eseguito il normale percorso completo, chiamato ***FULL ANALYSIS***. La sequenza viene fornita alla BiLSTM, dalla quale viene ottenuta la probabilità di attacco. Questa probabilità viene quindi inserita nello stato RL insieme al rapporto di fallimenti, al numero di richieste osservate, all'ora, all'attività notturna, al privilegio dell'utente, alla criticità della risorsa, alla presenza del certificato e al rischio storico. Lo stato normalizzato viene infine fornito al DQN, che restituisce i sei valori Q corrispondenti alle azioni disponibili. L'azione con valore Q massimo rappresenta la contromisura selezionata dal sistema come decisione finale.

4.5.3 Modalità Cold-Start

La pipeline completa descritta nelle sezioni precedenti presuppone la disponibilità di un numero sufficiente di eventi recenti per costruire una sequenza temporale valida. Il caso può verificarsi, ad esempio, quando viene osservata per la prima volta una determinata coppia ((User, IP)), quindi una nuova entità, quando la precedente attività dell'entità è ormai esterna alla finestra temporale considerata oppure quando gli eventi recenti disponibili non sono sufficienti. Per gestire tale situazione è stata introdotta una modalità di ***COLD-START***, che permette al sistema di produrre comunque una decisione senza ricorrere alla BiLSTM e alla DQN. La modalità Cold-Start viene attivata quando nella finestra temporale di cinque minuti non sono disponibili almeno cinque eventi, cioè il numero minimo richiesto per costruire una sequenza valida.
Questa distinzione è particolarmente importante nella modalità interattiva. Un'entità può infatti possedere una storia molto lunga ed essere già nota al sistema, ma non avere alcuna attività recente. Nel caso di un nuovo evento associato ad un'entità con migliaia di record storici, se l'ultimo evento precedente risale a un momento distante più di cinque minuti, la sequenza corrente contiene comunque un numero insufficiente di eventi e il modello BiLSTM non viene eseguito.
Quando si verifica questa situazione, il sistema non forza artificialmente la costruzione della sequenza e non utilizza il DQN in assenza dell'output della BiLSTM. Viene invece attivata una politica di cold-start, politica rule-based contestuale, basata esclusivamente sulle informazioni contestuali immediatamente disponibili, progettata come meccanismo temporaneo di fallback.
Il punteggio di rischio contestuale viene calcolato combinando otto segnali: fallimento dell'evento corrente, rapporto di richieste fallite, attività notturna, privilegio dell'utente, criticità della risorsa, assenza del certificato, rischio storico e pressione delle richieste. A ciascun segnale viene associato un peso esplicito e il punteggio complessivo viene ottenuto come somma dei contributi pesati.

Caratteristica	**Peso**
failed	0.20
failure_ratio	0.10
is_night	0.10
privileged	0.15
critical_resource	0.20
no_certificate	0.05
historical_risk	0.15
request_pressure	0.05

Tabella 4.4: Pesi associati alle caratteristiche nello scenario COLD_START

Il punteggio ottenuto viene confrontato con delle soglie predefinite, e in funzione di questo, la policy assegna una decisione, una delle quattro azioni temporanee previste. In questa modalità non vengono selezionate BLOCK_IP e QUARANTINE, che rimangono disponibili esclusivamente nel percorso completo BiLSTM-DQN, nel quale il sistema dispone anche dell'informazione temporale prodotta dal rilevatore. Per valori inferiori a 0,20 viene selezionata ALLOW, nell'intervallo compreso tra 0,20 e 0,40 viene selezionata MONITOR; tra 0,40 e 0,65 viene richiesta MFA; per valori pari o superiori a 0,65 viene applicata BLOCK_SESSION. Il Cold-Start costituisce quindi una politica sostitutiva e temporanea, e non una seconda modalità di funzionamento della DQN.
La modalità complessiva può essere sintetizzata nel seguente flusso:

$$\text{Nuovo evento} \rightarrow \text{Verifica contesto temporale} \rightarrow \begin{cases} \text{Sequenza sufficiente} \rightarrow \text{BiLSTM} \rightarrow \text{RL State} \rightarrow \text{DQN} \\ \text{Sequenza insufficiente} \rightarrow \text{Cold-Start Policy} \end{cases}$$

La modalità cold-start mantiene inoltre una distinzione tra la criticità della risorsa utilizzata dal modello congelato e una valutazione contestuale estesa impiegata dalla sola politica di fallback. La prima rimane coerente con la definizione adottata durante l'addestramento della BiLSTM; la seconda consente alla politica iniziale di considerare una gamma più ampia di risorse potenzialmente sensibili senza modificare il comportamento dei modelli già addestrati.
L'output della modalità cold-start non fornisce quindi una semplice azione finale, ma anche gli elementi utilizzati per giustificarla: punteggio complessivo, componenti del punteggio e segnali di rischio rilevati.
Quando, con l'evoluzione della storia dell'entità, diventano disponibili almeno cinque eventi nella finestra temporale prevista, il sistema abbandona il cold-start e passa automaticamente alla modalità di analisi completa. In questo modo il fallback costituisce un meccanismo di avvio e continuità operativa, mentre il percorso BiLSTM-DQN rappresenta il normale processo decisionale quando è disponibile una quantità sufficiente di informazione temporale.

Capitolo 5

Risultati

In questo capitolo verranno illustrati i risultati sperimentali dell'architettura progettata, dai risultati della BiLSTM a quelli del DQN, con alcuni esempi di aggiunta di nuovi eventi relativi alla pipeline interattiva.

In questi primi risultati si è utilizzata una suddivisione del dataset che mantenesse l'ordine temporale degli eventi, in modo da evitare che informazioni provenienti da periodi successivi fossero utilizzate durante l'addestramento. Il dataset è stato suddiviso in tre porzioni: 80% per il training, 10% per la validation e 10% per il test. La porzione di training è stata utilizzata per l'aggiornamento dei pesi della rete BiLSTM, mentre la validation è stata impiegata durante l'addestramento per monitorare l'andamento del modello e gestire le procedure di controllo della convergenza. La porzione di test è stata mantenuta separata e utilizzata esclusivamente nella fase finale di valutazione.
Sul test set sono state complessivamente analizzate 5.626 sequenze, di cui 1.320 appartenenti alla classe normale e 4.306 appartenenti alla classe di attacco. Per ciascuna sequenza, la BiLSTM produce una probabilità continua di appartenenza alla classe di attacco. Tale informazione costituisce successivamente uno degli elementi dello stato osservato dalla DQN, insieme alle variabili contestuali e temporali. La valutazione è stata articolata in due livelli distinti. Nel primo livello è stata analizzata la capacità della BiLSTM di distinguere gli eventi normali da quelli di attacco mediante le principali metriche di classificazione e la matrice di confusione. Nel secondo livello è stata valutata la componente di decisione automatica, costituita dal DQN, considerando sia le azioni selezionate nei diversi scenari sia l'interazione tra la probabilità prodotta dalla BiLSTM e lo stato utilizzato dall'agente. Per rendere confrontabili le due componenti, nella valutazione della pipeline integrata sono stati utilizzati gli stessi campioni del test set e gli stessi modelli addestrati, mantenuti congelati durante l'inferenza. In questo modo, la DQN non modifica il comportamento della BiLSTM, ma utilizza il suo output come informazione di ingresso per la selezione della contromisura.

Le analisi successive distinguono pertanto la fase di rilevazione, relativa alla BiLSTM, dalla fase di decisione, relativa al DQN. Questa distinzione consente di valutare non solo la qualità della classificazione iniziale, ma anche la capacità della pipeline complessiva di associare alle situazioni osservate una risposta difensiva coerente con lo stato di rischio.

5.1 Valutazione BiLSTM

La prima fase della valutazione riguarda il modello BiLSTM. La valutazione considera le metriche di classificazione ottenute dal modello e la relativa matrice di confusione, con l'obiettivo di quantificare la capacità del rilevatore di distinguere le due classi (classe 0 normali, classe 1 attacchi).
Le metriche analizzate sono quelle discusse nel paragrafo 4.3.4.

Classe	**Precision**	**Recall**	**F1-score**	**Support**
0	0.95	0.99	0.97	1320
1	1.00	0.98	0.99	4306
Macro avg	0.97	0.99	0.98	5626
Weighted avg	0.98	0.98	0.98	5626
Accuracy	0.98			
ROC-AUC	0.99948			

Tabella 5.1: Classification report della BiLSTM

I risultati del modello ottenuti in questa configurazione sono ottimi e mostrano prestazioni elevate. Il modello ha raggiunto un'accuratezza complessiva pari al 98%. Tuttavia, considerata la diversa numerosità delle due classi, il solo valore di accuracy non è sufficiente a descrivere in modo completo il comportamento del classificatore e per questo motivo sono state analizzate anche precision, recall e F1-score. Queste ultime metriche sono riportate separatamente per le due classi, poiché misurano le prestazioni del modello considerando alternativamente la classe normale e quella sospetta come classe di riferimento.
Il valore di precision è risultato pari al 95% per la classe normale e al 100% per la classe di attacco, mentre il recall è risultato rispettivamente pari al 99% e al 98%. I corrispondenti valori di F1-score sono 0,97 per la classe normale e 0,99 per la classe di attacco. Nel complesso, il *macro average* presenta precision pari a 0,97, recall pari a 0,99 e F1-score pari a 0,98, mentre il *weighted average* assume un valore pari a 0,98 per tutte e tre le metriche. Per quanto riguarda *macro avg*, questo è semplicemente la media aritmetica delle metriche delle due classi, mentre *weighted avg* tiene conto di quanti campioni ci sono per ciascuna classe. Il valore di ROC-AUC pari a 0,99948 evidenzia inoltre un'elevata capacità del modello di separare le due classi sulla base della probabilità di attacco prodotta.

Il comportamento delle due classi può essere ulteriormente interpretato distinguendo precision e recall. Per la classe di attacco, una precision pari a 1,00 indica che, tra le sequenze classificate dal modello come attacchi, non sono stati osservati falsi positivi nel campione considerato secondo la metrica riportata dal classification report. Il recall pari a 0,98 indica invece che una parte limitata degli attacchi presenti nel test set non è stata rilevata dalla BiLSTM. Per la classe normale, il recall pari a 0,99 indica che la quasi totalità dei campioni normali è stata correttamente riconosciuta, mentre la precision pari a 0,95 riflette la presenza di alcuni campioni classificati come normali che, rispetto all'etichetta reale, appartengono invece alla classe di attacco.
La matrice di confusione ottenuta sul test set è:

	Pred 0	**Pred 1**
True 0 (normale)	1306	14
True 1 (attacco)	76	4230

Tabella 5.2: Matrice di Confusione

I risultati mostrano 1306 eventi normali correttamente classificati (veri negativi, TN) e 4230 eventi sospetti correttamente identificati (veri positivi, TP). Gli errori sono costituiti da 14 falsi positivi (FP), ovvero eventi normali classificati come sospetti, e 76 falsi negativi (FN), ovvero eventi sospetti classificati come normali. Nel complesso, risultano quindi correttamente classificati 5536 campioni su 5626. Il numero contenuto di falsi positivi mostra che il modello classifica come sospette soltanto 14 sequenze effettivamente normali, mentre i 76 falsi negativi rappresentano i casi nei quali una sequenza di attacco non viene rilevata dalla BiLSTM. Considerando separatamente le due classi, il modello riconosce correttamente 1.306 delle 1.320 sequenze normali, corrispondenti a circa il 98,9%, e 4.230 delle 4.306 sequenze di attacco, corrispondenti a circa il 98,2%. Nello specifico si può dire come i 76 falsi negativi siano particolarmente rilevanti, questi infatti costituiscano l'errore di maggiore interesse dal punto di vista della sicurezza, poiché corrispondono proprio ad attacchi non rilevati dal modulo di classificazione. Questi campioni costituiscono successivamente uno degli elementi principali dell'analisi dell'interazione tra BiLSTM e DQN, poiché permettono di verificare se le informazioni contestuali presenti nello stato dell'agente possano portare comunque alla selezione di una contromisura.
Nel complesso, i risultati mostrano che, sul test set considerato, la BiLSTM fornisce una separazione molto marcata tra le due classi, caratterizzata da elevati valori di precision, recall e F1-score e da un numero limitato di errori. La successiva analisi della DQN non viene quindi interpretata come una sostituzione del modello di rilevazione, ma come un livello decisionale che utilizza la probabilità di attacco insieme alle informazioni contestuali dello stato RL per determinare la risposta difensiva.

5.2 Analisi dello stato utilizzato dalla DQN

Prima di analizzare il comportamento della componente decisionale, per comprenderlo meglio, è stata analizzata la distribuzione degli stati forniti al DQN, distinguendo i campioni appartenenti alla classe normale da quelli appartenenti alla classe di attacco, esclusivamente per il test set. L'analisi permette di verificare come l'informazione prodotta dalla BiLSTM venga combinata con gli ulteriori indicatori temporali e contestuali che compongono lo stato dell'agente.

Variabile	**Normale**	**Attacco**
Attack probability – media	0,0096	0,9689
Failure ratio – media	0,01	0,79
Requests 5m – media	701,38	2487,81
Privileged = 1	71.74%	39.97%
Critical resource = 1	0,08%	59,61%
Has certificate = 1	31,29%	0,00%
Historical risk – media	0,0107	0,9275
Risk $\geq$ 0,50	1,06%	97,72%

Tabella 5.3: Confronto della distribuzione delle principali variabili dello stato RL

La principale differenza più evidente tra le due classi è la probabilità di attacco prodotta dalla BiLSTM. Nei campioni normali il valore medio è pari a 0,0096, con una mediana di 0,0006 e un valore massimo di 0,9104, mostrando la presenza di quei rari casi nei quali il rilevatore assegna una probabilità elevata anche a sequenze appartenenti alla classe normale. Nei campioni di attacco, invece, la media raggiunge 0,9689, la mediana 0,9999 e il 90° percentile coincide con 1,0000. Questo risultato evidenzia una netta separazione tra i due gruppi e conferma il comportamento già osservato nella valutazione del classificatore.
L'output della BiLSTM costituisce quindi un'informazione fortemente discriminante tra le due classi. Allo stesso tempo però, la presenza di valori elevati anche in alcuni stati normali mostra perché l'uso della sola probabilità di attacco non sia sufficiente per descrivere completamente il contesto nel quale deve essere presa la decisione. Un'ulteriore differenza è osservabile negli indicatori relativi alle richieste recenti e ai loro esiti. Per gli stati normali, il valore medio della variabile relativa alle richieste fallite è pari a 0,01, negli stati di attacco invece la media sale invece a 0,79. Il confronto mostra quindi una maggiore presenza di richieste fallite negli scenari classificati come attacco. Analogamente, il numero medio di richieste nella finestra di cinque minuti passa da 701,38 nei campioni normali a 2.487,81 nei campioni di attacco.

Lo stato comprende inoltre variabili binarie utilizzate per rappresentare il contesto della richiesta. Nel test considerato, il flag *is night* assume valore 1 nel 100% degli stati, sia per le sequenze normali sia per quelle di attacco, e questo è coerente con il fatto che il test set, avendo diviso temporalmente il dataset, ricade totalmente nel periodo notturno. Di conseguenza, all'interno di questo specifico insieme di valutazione, tale caratteristica non introduce una differenziazione osservabile tra le due classi.
Un'ulteriore differenza significativa riguarda invece la criticità della risorsa. Nei campioni normali soltanto lo 0,08% degli stati è associato a una risorsa critica, mentre negli attacchi tale valore raggiunge il 59,61%. La variabile relativa alla presenza del certificato presenta invece una distribuzione opposta: il certificato è presente nel 31,29% dei campioni normali e in nessuno dei campioni di attacco.
Il rischio storico evidenzia infine una separazione particolarmente marcata tra le due classi. Nei campioni normali il valore medio è 0,0107, mentre negli attacchi la media raggiunge 0,9275. Inoltre, il rischio è maggiore o uguale a 0,50 nel 1,06% dei campioni normali e nel 97,72% dei campioni di attacco. La differenza osservata evidenzia il contributo della memoria temporale all'interno dello stato. L'agente non riceve quindi esclusivamente informazioni relative alla singola sequenza analizzata, ma anche un indicatore sintetico derivato dalla storia precedente dell'entità considerata.

La probabilità di attacco della BiLSTM presenta quindi la separazione più marcata tra le classi, mentre failure ratio, request pressure, criticità della risorsa e rischio storico aggiungono ulteriori elementi descrittivi dello scenario. La distribuzione di alcune variabili binarie risulta fortemente associata alla classe e questi aspetti devono essere considerati nell'interpretazione dei risultati, soprattutto in relazione alla natura sintetica del dataset.

5.3 Valutazione DQN

La seconda fase della valutazione riguarda il comportamento dell'agente DQN nella selezione della contromisura. A differenza della BiLSTM, la DQN non produce una classificazione binaria dell'evento, ma associa allo stato osservato una delle sei azioni definite nello spazio d'azione.

L'analisi è stata effettuata sugli stessi 5.626 stati del test set utilizzati per la valutazione della BiLSTM. L'obiettivo qui è stato quello di osservare la distribuzione delle decisioni prodotte dall'agente e verificare come queste variano tra situazioni normali e di attacco.

Azione	**Sequenze**	**Percentuale**
ALLOW	1.393	24,76%
MONITOR	0	0,00%
MFA	23	0,41%
BLOCK_SESSION	24	0,43%
BLOCK_IP	4.181	74,32%
QUARANTINE	5	0,09%

Tabella 5.4: Distribuzione complessiva delle azioni selezionate dal DQN

Il dato più evidente è la forte concentrazione delle decisioni sulle azioni ALLOW e BLOCK_IP, che rappresentano complessivamente il 99,08% delle decisioni. Le altre azioni vengono selezionate solo in una frazione limitata dei casi: MFA nello 0,41%, BLOCK_SESSION nello 0,43% e QUARANTINE nello 0,09%, mentre MONITOR non viene selezionata nel test set considerato.
La distribuzione varia sensibilmente considerando separatamente le sequenze appartenenti alle due classi, come è possibile notare nella tabella 5.5. Tra le 1.320 sequenze realmente normali, l'azione ALLOW è stata scelta in 1.305 casi, pari al 98,86%. Le restanti decisioni comprendono 1 MFA, 1 BLOCK_SESSION e 13 BLOCK_IP. Per le 4.306 sequenze di attacco, invece, la DQN ha selezionato BLOCK_IP in 4.168 casi, pari al 96,80%, mentre ALLOW è stato utilizzato in 88 casi (2,04%); le restanti decisioni sono distribuite tra MFA, BLOCK_SESSION e QUARANTINE.

	TRUE NORMAL		**TRUE ATTACK**	
Azione	**Sequenze**	**Percentuale**	**Sequenze**	**Percentuale**
ALLOW	1.305	98,86%	88	2,04%
MONITOR	0	0,00%	0	0,00%
MFA	1	0,08%	22	0,51%
BLOCK_SESSION	1	0,08%	23	0,53%
BLOCK_IP	13	0,98%	4.168	96,80%
QUARANTINE	0	0,00%	5	0,12%

Tabella 5.5: Distribuzione delle azioni DQN per True Normal e True Attack

Questo comportamento evidenzia una netta separazione della politica decisionale tra i due gruppi: nei campioni normali la DQN tende a mantenere l'accesso consentito, mentre nei campioni di attacco tende ad adottare una misura di contenimento diretto dell'entità associata all'indirizzo IP. Si può notare tuttavia una forte concentrazione della politica su BLOCK_IP, dovuta probabilmente alla divisione temporale e al fatto che nel test set siano presenti più eventi malevoli rispetto a quelli normali.

Le altre azioni risultano assenti o quasi, indicando che nella configurazione sperimentale considerata, l'agente ha appreso una politica fortemente orientata verso due comportamenti principali.

5.3.1 Interazione BiLSTM–DQN

Una possibile valutazione iniziale del comportamento del DQN può essere ottenuta trasformando la decisione DQN in una classificazione operativa binaria, considerando ALLOW come comportamento associato a una situazione normale e le azioni di mitigazione come risposta a una situazione di attacco. Con questa interpretazione, la pipeline integrata ottiene un'accuratezza pari al 98,17%, precision 99,65%, recall 97,96% e F1-score 98,79%.
La relativa matrice di confusione è:

	Pred 0	**Pred 1**
True 0 (normale)	1305	15
True 1 (attacco)	88	4218

Tabella 5.6: Matrice di Confusione DQN

Il confronto con la sola BiLSTM mostra una variazione limitata delle metriche aggregate. L'accuratezza integrata risulta in questo caso specifico quindi leggermente inferiore a quella della sola BiLSTM (98,17% contro 98,40%). Questo dato è coerente con il differente obiettivo dei due moduli: *il DQN non è progettato per migliorare direttamente la classificazione binaria, ma per selezionare una contromisura sulla base dello stato di sicurezza osservato.* Questo confronto non deve essere interpretato come una semplice competizione tra due classificatori, poiché la DQN non ha come obiettivo primario quello di migliorare la classificazione binaria. Il suo compito è trasformare le informazioni contenute nello stato in una azione difensiva. Tuttavia, verrà illustrato nella sezione successiva un differente scenario operativo, nel quale l'azione di reinforcement learning del DQN fornisce un incremento significativo rispetto alle metriche della componente BiLSTM.

Per questo motivo, una parte più significativa dell'analisi riguarda il comportamento del DQN nei confronti dei casi già classificati correttamente o erroneamente dalla BiLSTM.
Per valutare quindi il comportamento della pipeline, in particolare del secondo livello decisionale, è necessario verificare come il DQN reagisca alle diverse tipologie di risultato prodotto dalla BiLSTM, distinguendo in particolare tra veri positivi, veri negativi, falsi positivi e falsi negativi. Sono state quindi analizzate separatamente queste quattro categorie.

I 1.306 veri negativi della BiLSTM vengono mantenuti come ALLOW dal DQN in 1.305 casi, pari al 99,92%. Soltanto un campione viene associato a un'azione diversa, ovvero MFA. Questo comportamento è sintetizzato dal *TN preservation rate* del 99,92%, che misura la quota di decisioni normali della BiLSTM conservate dalla componente decisionale.
Per i 14 falsi positivi, invece, il DQN non seleziona mai ALLOW. In 13 casi viene scelta BLOCK_IP, mentre in un caso viene selezionata BLOCK_SESSION. La metrica di *FP correction rate*, definita nella sperimentazione, risulta pertanto pari allo 0,00%; la componente decisionale non corregge i falsi positivi prodotti dal rilevatore.
Il comportamento più rilevante riguarda i 76 falsi negativi della BiLSTM, ossia i casi nei quali un attacco non viene correttamente riconosciuto dal modello di rilevazione. In 55 casi il DQN seleziona ALLOW, mentre nei rimanenti 21 casi produce un'azione difensiva: 9 MFA, 9 BLOCK_IP e 3 QUARANTINE. Ne deriva un *FN recovery rate* pari al 27,63% (21 su 76). Questo risultato è particolarmente rilevante per la valutazione dell'architettura proposta, poiché mostra che una parte degli errori del rilevatore viene comunque associata dalla componente decisionale a una risposta difensiva. Il DQN non modifica la classificazione prodotta dalla BiLSTM, ma sfrutta lo stato RL, che comprende anche informazioni contestuali e storiche, per determinare l'azione da eseguire, e in questo caso "risolve" il precedente errore commesso.
Infine, tra i 4.230 veri positivi della BiLSTM, la DQN seleziona un'azione difensiva in 4.197 casi, mentre in 33 casi sceglie ALLOW. Il *TP response rate* raggiunge quindi il 99,22%, indicando che nella quasi totalità dei casi già riconosciuti come attacchi la componente decisionale produce effettivamente una contromisura.
I risultati relativi all'interazione possono essere riassunti come segue:

Tipologia	**Totale**	**ALLOW**	**Azione difensiva**	**Tasso osservato**
TN	1.306	1.305	1	99,92% TN preservation
FP	14	0	14	0,00% FP correction
FN	76	55	21	27,63% FN recovery
TP	4.230	33	4.197	99,22% TP response

Tabella 5.7: Interazione tra classificazione BiLSTM e azione DQN

Questi valori permettono di caratterizzare il contributo del DQN in modo più specifico rispetto alle sole metriche di classificazione. Il risultato più significativo è rappresentato dal recupero di 21 dei 76 falsi negativi prodotti dalla BiLSTM, ottenuto attraverso una decisione difensiva basata sullo stato RL anche quando la classificazione iniziale risultava errata. Allo stesso tempo, il comportamento osservato evidenzia che una parte dei falsi negativi rimane non rilevata e che i falsi positivi della BiLSTM non vengono corretti dalla politica appresa nel test considerato.

Per fare un confronto complessivo, alla fine, dei 5536 risultati inizialmente corretti, 5502 vengono mantenuti corretti, corrispondenti al 99.39%. Dei 90 risultati invece inizialmente errati dalla BiLSTM, il DQN ne corregge 21, quindi il 23.33% degli errori iniziali viene recuperato.
La qualità della pipeline integrata non è pertanto descritta esclusivamente dalla variazione dell'accuracy, ma anche dalla capacità della componente decisionale di preservare le decisioni corrette e di intervenire su una parte degli attacchi non identificati dal rilevatore.

5.3.2 Differente scenario sperimentale

Per analizzare il comportamento della componente RL anche in presenza di un rilevatore meno performante, è stato considerato un secondo scenario sperimentale, nel quale il dataset è stato suddiviso in 60% training, 10% validation e 30% test. La maggiore dimensione della porzione di test consente inoltre di valutare il comportamento della pipeline su un insieme più ampio, costituito in questo caso da 16.887 sequenze, di cui 9.522 normali e 7.365 di attacco.
In questo scenario la differenza rispetto al caso principale risulta particolarmente evidente già nella valutazione della BiLSTM. Il modello raggiunge infatti un'accuratezza pari soltanto al 56,52%, con una probabilità media di attacco pari a 0,1399. La matrice di confusione è:

	Pred 0	**Pred 1**
True 0 (normale)	9522	0
True 1 (attacco)	7343	22

Tabella 5.8: Matrice di confusione split 60-10-30

Classe	**Precision**	**Recall**	**F1-score**	**Support**
0	0.56	1.00	0.72	9522
1	1.00	0.00	0.01	7365
Macro avg	0.78	0.50	0.36	16887
Weighted avg	0.75	0.57	0.41	16887
Accuracy	0.57			
ROC-AUC	0.85714			

Tabella 5.9: Classification report BiLSTM 60-10-30

Il modello riconosce quindi correttamente tutti i 9.522 campioni normali, ma identifica come attacco soltanto 22 dei 7.365 casi realmente appartenenti alla classe 1. Il recall della classe di attacco è pertanto pari allo 0,30%, mentre l'F1-score della stessa classe è pari a 0,01. Il valore di ROC-AUC pari a 0,8571 indica comunque una capacità di separazione superiore a quella che emerge dalla classificazione ottenuta con la soglia utilizzata, ma non si traduce in una rilevazione efficace degli attacchi nel test set.

Questo scenario costituisce quindi una condizione particolarmente sfavorevole per la pipeline, poiché la componente di rilevazione fornisce una quantità molto elevata di falsi negativi, 7.343 attacchi su 7.365 non vengono infatti classificati correttamente.
L'integrazione del DQN modifica radicalmente il comportamento osservato. Applicando la stessa interpretazione operativa utilizzata nello scenario precedente, nella quale ALLOW rappresenta la decisione associata a una situazione normale e le altre azioni costituiscono una risposta difensiva, la pipeline raggiunge un'Accuracy del 97,39%, Precision di 99,50%, Recall del 94,50% e F1-score pari a 96,94%. La matrice di confusione associata alla decisione DQN è:

	Pred 0	**Pred 1**
True 0 (normale)	9487	35
True 1 (attacco)	405	6960

Tabella 5.10: Matrice di confusione DQN split 60-10-30

Tipologia	**Totale**	**ALLOW**	**Azione difensiva**	**Tasso osservato**
TN	9522	9487	35	99,63% TN preservation
FP	0	0	0	0,00% FP correction
FN	7343	405	6938	94,48% FN recovery
TP	22	0	22	100,00% TP response

Tabella 5.11: Interazione tra classificazione BiLSTM e azione DQN 60-10-30

La variazione è particolarmente evidente nel trattamento degli attacchi. Dei 7.343 falsi negativi della BiLSTM, la DQN produce una risposta difensiva in 6.938 casi, pari al 94,48%.
Il *FN recovery rate* pari al 94,48% rappresenta il risultato più significativo di questo secondo esperimento. Il DQN riesce infatti a selezionare una contromisura nella maggior parte dei casi che il rilevatore aveva classificato erroneamente come normali. In particolare, dei 7.343 falsi negativi della BiLSTM, 4.705 ricevono BLOCK_IP, 2.143 MFA e 90 MONITOR. I restanti 405 vengono invece lasciati in ALLOW. Anche il comportamento sui campioni normali rimane fortemente conservativo: dei 9.522 veri negativi della BiLSTM, 9.487, pari al 99,63%, ricevono ALLOW. Nei restanti casi vengono selezionate MONITOR in 2 occasioni e MFA in 33 occasioni. Il valore di *TN preservation rate* pari al 99,63% mostra quindi che l'aumento delle risposte difensive non comporta, nello scenario considerato, una perdita significativa della capacità di mantenere autorizzati i campioni normali.

Diversamente dal primo scenario, la distribuzione delle azioni risulta inoltre più articolata. Nei campioni di attacco, BLOCK_IP rappresenta il 64,18% delle decisioni, MFA il 29,10%, MONITOR l'1,22% e ALLOW il 5,50%. Nei campioni normali, invece, ALLOW raggiunge il 99,63%, mentre le rimanenti azioni risultano residuali.

Il confronto tra i due scenari evidenzia pertanto un comportamento differente della politica DQN. Nel primo esperimento, caratterizzato da una BiLSTM già molto accurata, l'agente assume prevalentemente decisioni ALLOW o BLOCK_IP. Nel secondo, dove la capacità di rilevazione della BiLSTM è fortemente ridotta, la DQN utilizza maggiormente anche MFA e MONITOR, producendo comunque una classificazione operativa complessiva prossima al 97% di accuratezza.

Questi risultati evidenziano il contributo della componente di reinforcement learning in una condizione nella quale il rilevatore supervisionato presenta prestazioni nettamente inferiori. In particolare, il DQN riesce a sfruttare le informazioni disponibili nello stato RL per produrre una risposta difensiva anche in un numero molto elevato di casi classificati erroneamente dalla BiLSTM come normali.
Il secondo esperimento mostra quindi soprattutto la capacità dell'architettura integrata di compensare parzialmente una debolezza del modulo di rilevazione attraverso il livello decisionale, evidenziando il valore aggiunto dell'approccio a più livelli adottato nel progetto.

5.4 Valutazione modalità online

La valutazione sperimentale è stata completata analizzando anche il comportamento della pipeline in modalità online. In questa sezione verranno mostrati quelli che sono i risultati, le decisioni del "decision point", in relazione a diversi tipi di richieste esempio date in input.
Per dimostrare i diversi tipi di funzionamento, sono stati analizzati sia casi relativi a entità già presenti nello storico del sistema, sia casi relativi a nuove entità. La distinzione permette di verificare il comportamento della pipeline in presenza di differenti quantità di informazioni pregresse e, soprattutto, di mostrare come il sistema gestisca il caso in cui la storia recente non sia sufficiente per l'analisi sequenziale e quindi agisca la policy *cold-start* a differenza della *full analysis*.

5.4.1 Caso COLD START nuova entità

Il primo caso riguarda una nuova entità, per la quale quindi non esiste nessuna storia passata e un rischio storico, identificata dall'utente *matteo* e dall'indirizzo IP *192.168.10.11*. L'evento ricevuto costituisce quindi il primo elemento associato alla coppia user-ip.


```
{       "time": "22-Dec-2025 00:12:17",
```

```
        "message": {
            "IP": "192.168.10.11",
            "User": "matteo",
            "HTTPStatusCode": 400,
            "ResourceContext": {
                "database": "DataAuth",
                "table": "dbo.DatabaseBackupHistory"},
            "ClientCertificate": null,
            "UserContext": {
                "roles": ["Administrator"]              }
}}}
```


Nell'input della richiesta sono state inserite solo le caratteristiche principali utili all'analisi.
Il sistema identifica correttamente l'entità come **NEW** e rileva un solo evento disponibile. In questo caso, quindi, essendo una nuova entità, non è possibile costruire una sequenza valida né per la BiLSTM e né tanto meno per il successivo DQN.
Per definizione infatti, una nuova entità entra inizialmente in una condizione di cold-start: al primo evento non esiste infatti una storia temporale sufficientemente lunga per soddisfare il requisito minimo di cinque eventi richiesto dalla pipeline sequenziale. Non si tratta quindi di un malfunzionamento o di un limite accidentale dell'implementazione, ma di una condizione prevista dal modello operativo.
Le caratteristiche temporali e contestuali estratte dall'evento sono riportate nella seguente tabella.

Feature	**Valore**
Failed	1
Failure ratio	1,0000
Requests 5m	1,0000
Historical risk	0,0000
Hour	0
Is night	1
Privileged	1
Critical resource	1
Has certificate	0

Tabella 5.12: Caratteristiche temporali e contestuali del caso cold-start su nuova entità (attacco)

Si può subito notare come il parametro *historical risk* assuma il valore zero, visto che si sta parlando di una nuova entità mai osservata prima, non essendo disponibile alcuna memoria del comportamento precedente.

Indicatore	Valore
Context risk score	0,8041
Request pressure	0,0824
Temporary decision	**BLOCK_SESSION**
Decision source	Rule-based cold-start policy

Tabella 5.13: Output della politica contestuale di cold-start (attacco)

Il *context risk score* ottenuto è pari a 0,8041 e in questo caso la politica di cold-start determina la decisione **BLOCK_SESSION**.
La decisione è nata da alcuni fattori tra cui: evento corrente fallito, attività notturna, utente privilegiato, risorsa critica e mancanza di certificato. I singoli contributi numerici possono essere osservati di seguito.

Segnale	Contributo
Failed	0,2000
Failure ratio	0,1000
Is night	0,1000
Privileged	0,1500
Critical resource	0,2000
No certificate	0,0500
Historical risk	0,0000
Request pressure	0,0041

Tabella 5.14: Segnali di rischio e contributi al punteggio della politica di cold-start (attacco)

Come esempio aggiuntivo viene anche fornito il caso di una richiesta normale quindi con diversi contributi al punteggio finale.


```
{
        "time": "26-Nov-2025 02:02:01",
        "message": {
            "IP": "192.169.26.11",
            "User": "luca",
            "HTTPStatusCode": 200,
            "ResourceContext": {
                "database": "Data",
                "table": "dbo.Model"
            },
            "ClientCertificate": "A8F821318660A91949753F4A36
   EF6542095B96B5",
            "UserContext": {
                "roles": [
                    "user"
                ]
```

```
            }
        }
}
```

Indicatore	**Valore**
Context risk score	0,1041
Request pressure	0,0824
Temporary decision	**ALLOW**
Decision source	Rule-based cold-start policy

Tabella 5.15: Output della politica contestuale di cold-start (normale)

In questo caso il *context risk score* assume un valore molto basso pari a 0.1041 e quindi la policy di cold-start assegna la decisione **ALLOW**.

Segnale	**Contributo**
Failed	0,0000
Failure ratio	0,0000
Is night	0,1000
Privileged	0,0000
Critical resource	0,0000
No certificate	0,0000
Historical risk	0,0000
Request pressure	0,0041

Tabella 5.16: Segnali di rischio e contributi al punteggio della politica di cold-start (normale)

Si può notare come in questo caso, la presenza di certificato, l'utente non privilegiato, la risorsa non critica e l'evento non fallito, contribuiscano alla decisione di lasciar passare la richiesta senza nessuna contromisura.

5.4.2 Caso COLD START entità esistente

Il secondo caso analizzato riguarda un'entità presente nello storico, ma in una condizione nella quale non è disponibile una sequenza temporale sufficiente per la BiLSTM. Questa è caratterizzata da una lunga assenza di attività recente, infatti sono presenti 4674 eventi, ma soltanto 1 nella finestra temporale di 5 minuti. Il sistema riconosce in questo caso l'entità come **EXISTING**, ma per mancanza di storia recente viene gestita dalla policy di cold-start. La motivazione risulta infatti "*existing entity, but previous activity is outside the 5m window*".

Feature	Valore
Failed	0
Failure ratio	0,0000
Requests 5m	1,0000
Historical risk	0,9500
Hour	22
Is night	1
Privileged	1
Critical resource	0
Has certificate	0

Tabella 5.17: Caratteristiche temporali e contestuali del caso cold-start su entità esistente

Indicatore	Valore
Context risk score	0,4466
Request pressure	0,0824
Temporary decision	**MFA**
Decision source	Rule-based cold-start policy

Tabella 5.18: Output della politica contestuale di cold-start entità esistente

Il *context risk score* risultante è pari a 0,4466, mentre la componente relativa alla pressione delle richieste è pari a 0,0824. La politica produce quindi la decisione **MFA**. Il punteggio contestuale è costruito considerando separatamente diversi segnali: attività notturna, utente privilegiato, assenza del certificato e rischio storico dell'utente.

Segnale	Contributo
Failed	0,0000
Failure ratio	0,0000
Is night	0,1000
Privileged	0,1500
Critical resource	0,0000
No certificate	0,0500
Historical risk	0,1425
Request pressure	0,0041

Tabella 5.19: Segnali di rischio e contributi al punteggio della politica di cold-start entità esistente

La principale differenza rispetto al caso precedente è la presenza del rischio storico, questa richiesta dispone di un valore pari a 0.95 che influenza il punteggio finale di 0.1425. Anche se di base l'evento non sembri eccessivamente sospetto, tramite questo contributo è stato possibile definire una contromisura, seppur ridotta come MFA, che non lasciasse passare indisturbata la richiesta.

5.4.3 Caso FULL ANALYSIS

Il primo esempio di modalità di analisi completa riguarda un'entità già presente nello storico. L'evento analizzato appartiene all'utente *andrea*, associato all'indirizzo IP *192.168.3.20*, con timestamp *2025-12-21 22:40:45*. Al momento dell'elaborazione sono presenti 4.675 eventi nello storico e 50 eventi nella finestra temporale di cinque minuti, corrispondente al numero massimo di eventi utilizzato per la sequenza.
Il sistema rileva correttamente l'entità come **EXISTING** e le informazioni temporali presenti sono sufficienti per costruire una sequenza valida e procedere con la catena completa.


```
{
        "time": "21-Dec-2025 22:40:45",
        "message": {
            "IP": "192.168.3.20",
            "User": "andrea",
            "HTTPStatusCode": 400,
            "ResourceContext": {
                "database": "DataAuth",
                "table": "dbo.DatabaseBackupHistory"
            },
            "ClientCertificate": null,
            "UserContext": {
                "roles": [
                    "Administrator"
                ]
            }
        }
}
```

Feature	Valore
Failed	1
Failure ratio	0,3400
Requests 5m	3253,0000
Historical risk	0,9500
Hour	22
Is night	1
Privileged	1
Critical resource	0
Has certificate	0
BiLSTM attack probability	0.999851

Tabella 5.20: Caratteristiche temporali e contestuali del caso full-analysis (attacco)

La BiLSTM produce una *attack probability* pari a 0.999851, indicando una probabilità molto elevata di appartenenza alla classe di attacco. Tali informazioni vengono successivamente integrate nello stato RL e normalizzate prima dell'ingresso nel DQN. Il valore di *requests_5m* viene riportato a 0,725368 e l'ora a 0,956522, mentre le altre componenti dello stato mantengono i valori mostrati.
Il DQN produce quindi un valore Q per ciascuna delle sei azioni disponibili.

Azione	Q-value
ALLOW	0,306173
MONITOR	43,647064
MFA	48,031460
BLOCK_SESSION	62,965572
BLOCK_IP	65,270912
QUARANTINE	58,068073

Tabella 5.21: Valori Q-value calcolati per le diverse azioni (attacco)

L'azione associata al valore Q massimo è BLOCK_IP. La decisione finale della pipeline è pertanto: "**DECISION: BLOCK_IP**"
L'esempio mostra il funzionamento completo della modalità online dove la disponibilità di una storia temporale consente al sistema di utilizzare entrambe le componenti apprese. Il nuovo evento viene integrato nello storico, viene costruita la rappresentazione temporale necessaria alla BiLSTM, la rete produce la probabilità di attacco e tale informazione viene successivamente combinata con il contesto per consentire alla DQN di selezionare la contromisura.

Per completare l'analisi online, viene considerato come esempio anche un secondo caso di FULL ANALYSIS relativo ad un evento normale non sospetto. Questo scenario dimostra come la pipeline non associa automaticamente una contromisura, ma quando la sequenza e il suo contesto indicano una situazione compatibile ad uno scenario normale, il DQN mantiene l'accesso selezionando ALLOW.


```
{
        "time": "21-Dec-2025 20:30:37",
        "message": {
            "IP": "192.168.3.120",
            "User": "vm-11",
            "HTTPStatusCode": 200,
            "ResourceContext": {
                "database": "Data",
                "table": "dbo.Chat"
            },
            "ClientCertificate": null,
            "UserContext": {
                "roles": [
                    "Administrator"
                ]
            }
        }
}
```


L'entità in questo esempio appartiene all'utente *vm-11*, con ip *192.168.3.120*. Sono presenti ben 36431 eventi complessivi nello storico di questa entità e nella finestra temporale dei 5 minuti c'è il numero massimo di 50 eventi.

Feature	**Valore**
Failed	0
Failure ratio	0,0000
Requests 5m	923,0000
Historical risk	0,0000
Hour	20
Is night	0
Privileged	1
Critical resource	0
Has certificate	0
BiLSTM attack probability	0.091059

Tabella 5.22: Caratteristiche temporali e contestuali del caso full-analysis (normale)

Il rischio storico a zero indica un'entità prettamente non malevola, anche considerando i numerosi eventi presenti. L'utente possiede il ruolo *Administrator*, che costituisce un indicatore associato a un'entità privilegiata e quindi un possibile rischio.

Nonostante però la presenza di quest'ultimo elemento, il contesto complessivo dell'evento non presenta altri segnali particolarmente elevati.
La BiLSTM produce una *attack probability* pari a 0,091059, nettamente inferiore rispetto al valore osservato nell'esempio di attacco. Lo stato normalizzato fornito al DQN presenta inoltre *failure_ratio* = 0, *requests_5m* = 0,205751, *is_night* = 0, *critical_resource* = 0 e *historical_risk* = 0.
I valori Q prodotti dal DQN sono:

Azione	**Q-value**
ALLOW	406.644318
MONITOR	177.065872
MFA	10.237235
BLOCK_SESSION	50.963036
BLOCK_IP	-41.163994
QUARANTINE	-11.728246

Tabella 5.23: Valori Q-value calcolati per le diverse azioni (normale)

L'azione associata al valore Q massimo è ALLOW. La decisione finale della pipeline è pertanto: "**DECISION: ALLOW**".
Si può notare come nel caso precedente i valori erano più compatti, ad eccezione del valore per ALLOW, mentre ora il valore associato ad ALLOW è nettamente superiore a quello delle altre azioni, che risultano comunque meno compatte. Si può inoltre osservare anche la presenza di valori negativi, ad indicare come quelle contromisure nel caso normale in esame siano estremamente sconsigliate. Si riprecisa che i valori Q non rappresentano direttamente il livello di rischio dell'evento, ma il ritorno atteso associato a ciascuna azione nello stato osservato.
Questo caso è significativo perché mostra che la presenza di un singolo indicatore potenzialmente rilevante, come il privilegio dell'utente, non determina automaticamente una risposta restrittiva. La decisione viene invece effettuata considerando congiuntamente le informazioni presenti nello stato: l'evento non presenta errori, il failure ratio è nullo, la risorsa non è considerata critica, l'attività non avviene in orario notturno e il rischio storico è pari a zero. La probabilità di attacco prodotta dalla BiLSTM è inoltre bassa.
Il comportamento osservato evidenzia quindi la capacità della pipeline di differenziare due eventi appartenenti a contesti differenti, evitando di applicare una contromisura invasiva esclusivamente sulla base di un singolo attributo.
Nel complesso, la modalità online permette di estendere la pipeline dalla valutazione offline del dataset alla gestione di eventi individuali, mantenendo una distinzione esplicita tra analisi completa e gestione delle situazioni di cold-start. Gli esempi analizzati mostrano come il sistema possa produrre una risposta sia nel normale percorso BiLSTM+DQN sia in assenza di una sequenza temporalmente sufficiente, attraverso il meccanismo di fallback contestuale.

Capitolo 6

Conclusioni e sviluppi futuri

6.1 Conclusioni

Il presente lavoro di tesi ha affrontato il problema della rilevazione e della gestione adattiva delle minacce in sistemi caratterizzati da richieste di accesso e attività distribuite, facendo riferimento ai principi del paradigma Zero-Trust. In questo contesto, la semplice verifica dell'identità dell'utente non è infatti sufficiente a garantire un adeguato livello di sicurezza, poiché il comportamento di un'entità può variare nel tempo e una richiesta apparentemente legittima può inserirsi all'interno di una sequenza di attività potenzialmente malevole. Questo ha reso necessario considerare anche il comportamento dell'entità e il contesto in cui le richieste vengono effettuate.

L'obiettivo principale del lavoro è stato quindi la progettazione e l'implementazione di una pipeline sperimentale in grado di combinare tecniche di *deep learning* e *reinforcement learning* per realizzare un processo di rilevazione e risposta adattivo. L'architettura sviluppata segue una sequenza composta inizialmente dalla rilevazione tramite *BiLSTM*, scelta per la sua capacità di elaborare sequenze di eventi e di individuare relazioni temporali tra le attività osservate che porta ad una stima della probabilità che l'attività osservata sia sospetta, e ad una successiva selezione della contromisura che avviene invece mediante un agente *DQN*. Lo stato fornito all'agente DQN combina la probabilità di attacco prodotta dal modello BiLSTM con informazioni contestuali relative all'attività osservata e al rischio accumulato.
Tale impostazione permette di separare la fase di rilevazione da quella di risposta, rendendo l'architettura potenzialmente più flessibile rispetto a un sistema nel quale ogni rilevazione corrisponde necessariamente a una singola azione predefinita.

I risultati sperimentali hanno mostrato ottime prestazioni del modello nella distinzione tra attività normali e sospette, evidenziando il potenziale dell'approccio proposto. I risultati hanno inoltre evidenziato l'importanza della selezione delle feature in quanto alcune caratteristiche possono fornire informazioni eccessivamente discriminative all'interno di un dataset sintetico, producendo prestazioni apparentemente molto elevate ma potenzialmente meno rappresentative di uno scenario reale.
Per questo motivo, l'analisi dei risultati non è stata limitata alle sole metriche ottenute, ma ha considerato anche la capacità di generalizzazione dell'approccio. Le prove effettuate mediante suddivisioni più restrittive dei dati, considerando la separazione temporale tra addestramento e test, hanno infatti mostrato un peggioramento delle prestazioni rispetto alle configurazioni più semplici. Questo comportamento rappresenta un elemento importante nell'interpretazione dei risultati, poiché evidenzia come prestazioni elevate su dati sintetici e fortemente correlati non implichino necessariamente la stessa efficacia in presenza di utenti, comportamenti e scenari non osservati durante l'addestramento.

Nel complesso, il lavoro ha permesso di verificare la fattibilità di una pipeline nella quale la rilevazione automatica delle attività sospette viene integrata con una successiva fase di selezione della risposta.
È tuttavia necessario considerare alcuni limiti. Il dataset utilizzato per la sperimentazione è sintetico e non riproduce necessariamente tutta la variabilità, il rumore e la complessità presenti nei sistemi reali. Inoltre, alcune caratteristiche del dataset possono risultare fortemente correlate con l'etichetta e facilitare il processo di classificazione. Anche la definizione delle finestre temporali, delle sessioni, delle feature di rischio e delle possibili azioni di mitigazione costituisce una semplificazione rispetto a un ambiente operativo reale. Di conseguenza, i risultati ottenuti devono essere interpretati principalmente come una validazione sperimentale dell'architettura proposta e non come una dimostrazione definitiva della sua efficacia in contesti produttivi.

Nonostante tali limitazioni, il lavoro mostra come l'integrazione di modelli sequenziali di *deep learning* e tecniche di *reinforcement learning* possa costituire una possibile direzione per lo sviluppo di sistemi di sicurezza più adattivi. L'architettura realizzata fornisce infatti una base sulla quale poter costruire sistemi capaci non soltanto di identificare comportamenti sospetti, ma anche di utilizzare le informazioni ottenute per modificare dinamicamente il livello di risposta in funzione del contesto osservato.

6.2 Sviluppi futuri

Alla luce dei risultati ottenuti e delle limitazioni evidenziate, il lavoro potrebbe essere ulteriormente sviluppato e amplificato lungo diverse direzioni che si pensa possano migliorarne l'efficacia e l'applicabilità.

- Un primo possibile sviluppo potrebbe essere l'estensione a contesti operativi reali, utilizzando dataset reali e maggiormente diversificati. L'utilizzo di dati provenienti da ambienti operativi, caratterizzati da una varietà maggiore in termine di utenti, dispositivi, risorse, comportamenti e tipologie di attacchi, permetterebbe di valutare in maniera più accurata la capacità di generalizzazione del modello e di ridurre il rischio che le prestazioni siano legate a caratteristiche specifiche del dataset sintetico.

- Un secondo possibile sviluppo consiste nell'integrazione dell'architettura direttamente all'interno di un ambiente Zero-Trust completo. Si potrebbe collegare direttamente ai componenti responsabili della gestione delle richieste di accesso, integrando il processo di rilevazione e decisione con un *Policy Enforcement Point(PEP)*, al quale spetta il compito di applicare concretamente le azioni stabilite dall'agente.

- Un'ulteriore direzione futura riguarda la gestione dinamica dello stato del sistema. Nel prototipo realizzato le informazioni relative al comportamento osservato sono sintetizzate attraverso feature temporali e indicatori di rischio. Una possibile evoluzione potrebbe prevedere l'uso di uno stato più ricco, capace di rappresentare in maniera più dettagliata l'evoluzione del comportamento di utenti, dispositivi, sessioni e risorse nel tempo.

- Una particolare attenzione potrebbe essere dedicata al miglioramento della componente di RL, all'evoluzione della funzione di reward. Sarebbe possibile studiare strategie di reward più articolate e automatiche, rendendo i parametri maggiormente adattabili al contesto, con procedure di calibrazione basate sui dati o strategie di apprendimento automatiche. L'importanza attribuita ai diversi indicatori di rischio e il costo delle azioni potrebbero essere adattati alle caratteristiche dello specifico ambiente, evitando di utilizzare necessariamente gli stessi valori in scenari differenti. Questo permetterebbe di valutare in maniera più completa il compromesso tra efficacia della mitigazione, sicurezza e impatto sugli utenti legittimi, rendendo la politica appresa dall'agente maggiormente adatta a scenari operativi reali.

- Infine, un ulteriore ambito riguarda l'interpretabilità delle decisioni. L'integrazione di tecniche di *Explainable AI* potrebbe consentire di analizzare quali caratteristiche e quali eventi abbiano contribuito maggiormente alla probabilità di attacco stimata dalla BiLSTM, e successivamente quali informazioni abbiano influenzato il DQN nella scelta dell'azione di mitigazione. Questa maggiore interpretabilità sarebbe particolarmente utile in sistemi di sicurezza, nei quali una decisione automatica dovrebbe poter essere analizzata e verificata da un amministratore.

Bibliografia

[1] Jericho Forum. The de-perimeterisation of information security: The jericho forum, zero trust, and narrativity. https://doi.org/10.1177/03063127231221107.

[2] David Heath. The evolution of zero trust and the frameworks that guide it. https://www.ibm.com/it-it/think/insights/the-evolution-of-zero-trust-and-the-frameworks-that-guide-it.

[3] Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. Zero trust architecture, 2020. https://doi.org/10.6028/NIST.SP.800-207.

[4] Antonio Francesco Gentile, Davide Macrì, and Emilio Greco. Studio delle tecnologie zero trust, zerotier, vpn ipsec, openvpn, wireguard e gestione accesso mediante micro servizi finalizzato alla progettazione di un ambiente zero trust, 2024. https://iris.cnr.it/retrieve/8d698dba-8879-4a76-ad09-895460426975/RT-ICAR-CS-24-01.pdf.

[5] Danial Haider, Shougfta Mushtaq, Hasnat Ali, and Mazliham Mohd Su'ud. Enhancing zero trust cybersecurity using machine learning and deep learning approaches. *Journal of Informatics and Web Engineering*, 4(3):24–35, 2025. https://doi.org/10.33093/jiwe.2025.4.3.2.

[6] IBM. Cos'è il machine learning? https://www.ibm.com/it-it/think/topics/machine-learning/.

[7] Cole Stryker. Cos'è una rete neurale ricorrente (rnn)? https://www.ibm.com/it-it/think/topics/recurrent-neural-networks.

[8] Simone Raponi. Problemi delle recurrent neural networks: Scomparsa del gradiente, 2023. https://www.redhotcyber.com/post/problemi-delle-recurrent-neural-networks-scomparsa-del-gradiente/.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. https://www.researchgate.net/publication/13853244_Long_Short-Term_Memory.

[10] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997. https://www.researchgate.net/publication/3316656_Bidirectional_recurrent_neural_networks.

[11] Jacob Murel and Eda Kavlakoglu. Cos'è l'apprendimento per rinforzo? https://www.ibm.com/it-it/think/topics/reinforcement-learning/.

[12] Christopher J. C. H. Watkins. *Learning From Delayed Rewards*. PhD thesis, University of Cambridge, 1989. https://www.researchgate.net/publication/33784417_Learning_From_Delayed_Rewards.

[13] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992. https://doi.org/10.1007/BF00992698.

[14] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2 edition, 2018. https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf.

[15] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 30–37, San Francisco, CA, USA, 1995. Morgan Kaufmann. http://leemon.com/papers/1995b.pdf.

[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015. https://daiwk.github.io/assets/dqn.pdf.

[17] Andrea Rossi. Architettura zero-trust: sistema di identificazione e monitoraggio degli eventi, 2025. https://tesi.univpm.it/handle/20.500.12075/25529.

[18] TensorFlow. Keras. https://www.tensorflow.org/guide/keras/.

[19] IBM. Scikit-learn. https://www.ibm.com/it-it/think/topics/scikit-learn/.

[20] Farama Foundation. Gymnasium. https://gymnasium.farama.org/introduction/basic_usage/.

[21] Matplotlib Development Team. Matplotlib. https://matplotlib.org/.

[22] Matteo Bocci. Tirocinio-bilstm-dqn, 2026.
`https://github.com/matteobocci45/Tirocinio-BiLSTM-DQN`.

Elenco delle figure

1.1 Componenti dell'architettura Zero Trust [3] 12

2.1 Differenza negli schemi di architettura delle reti neurali 21
2.2 Flusso dati dei gate lstm 22
2.3 Architettura BiLSTM con più fasi temporali 24
2.4 Schema funzionamento del Reinforcement Learning 26
2.5 Differenza tra Q-learning e DQN. © 2021 IEEE. 31

3.1 Topologia cyber-range Zero Trust a tre livelli 34
3.2 Distribuzione IP, utenti e sessioni 41
3.3 Tasso di sospetto per utenti 42
3.4 Concentrazione eventi sospetti per ip 43
3.5 Volume eventi nel corso del tempo 44
3.6 Volume eventi normali/sospetti nel corso del tempo 45
3.7 Distribuzione degli stati HTTP 46
3.8 Distribuzione degli stati HTTP normali e sospetti 47
3.9 Loghi TensorFlow e Keras 49
3.10 Logo Scikit-Learn 51
3.11 Logo Gymnasium 52
3.12 Logo Matplotlib 53

4.1 Diagramma di flusso del sistema di rilevamento e mitigazione . . . 86

Elenco delle tabelle

3.1 Descrizione dei principali campi del log 35
3.2 Principali campi degli eventi di audit applicativo 36
3.3 Campi dell'evento del captive portal 38
3.4 Campi dell'evento p0f 39
3.5 Dimensione e composizione dataset 40
3.6 Analisi temporale eventi 44

4.1 Descrizione dei campi del log finale 58
4.2 Costi associati alle azioni di sicurezza 77
4.3 Reward associati alle azioni nello scenario NORMAL 79
4.4 Pesi associati alle caratteristiche nello scenario COLD_START . . 88

5.1 Classification report della BiLSTM 92
5.2 Matrice di Confusione 93
5.3 Confronto della distribuzione delle principali variabili dello stato RL 94
5.4 Distribuzione complessiva delle azioni selezionate dal DQN 96
5.5 Distribuzione delle azioni DQN per True Normal e True Attack . . 96
5.6 Matrice di Confusione DQN 97
5.7 Interazione tra classificazione BiLSTM e azione DQN 98
5.8 Matrice di confusione split 60-10-30 99
5.9 Classification report BiLSTM 60-10-30 99
5.10 Matrice di confusione DQN split 60-10-30 100
5.11 Interazione tra classificazione BiLSTM e azione DQN 60-10-30 . . 100
5.12 Caratteristiche temporali e contestuali del caso cold-start su nuova entità (attacco) 102
5.13 Output della politica contestuale di cold-start (attacco) 103
5.14 Segnali di rischio e contributi al punteggio della politica di cold-start (attacco) 103
5.15 Output della politica contestuale di cold-start (normale) 104
5.16 Segnali di rischio e contributi al punteggio della politica di cold-start (normale) 104
5.17 Caratteristiche temporali e contestuali del caso cold-start su entità esistente 105

5.18 Output della politica contestuale di cold-start entità esistente . . . 105
5.19 Segnali di rischio e contributi al punteggio della politica di cold-start entità esistente 105
5.20 Caratteristiche temporali e contestuali del caso full-analysis (attacco) 107
5.21 Valori Q-value calcolati per le diverse azioni (attacco) 107
5.22 Caratteristiche temporali e contestuali del caso full-analysis (normale) 108
5.23 Valori Q-value calcolati per le diverse azioni (normale) 109