UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

UNIVERSITÀ DEGLI STUDI DI MODENA

E REGGIO EMILIA

Dipartimento di Ingegneria "Enzo Ferrari"

Corso di Laurea Magistrale in Ingegneria Informatica (LM-32)

Analisi e Mitigazione del Concept Drift in ML Security Operations Center

Relatore:
Prof. Mirco Marchetti
Correlatore:
Ing. Dimitri Galli

Candidato:
Mattia Compagnone
Matricola 206894

ANNO ACCADEMICO 2025/2026

Indice

Elenco delle Figure

Elenco dei Codici

1. Introduzione

1.1 L'evoluzione delle minacce informatiche nell'era dei Big Data

Nell'ultimo decennio il panorama delle minacce informatiche ha subito una trasformazione strutturale che ha reso obsoleti molti degli approcci difensivi tradizionali. La disponibilità di strumenti di attacco automatizzati, la crescita esponenziale delle superfici di attacco connesse a Internet e la professionalizzazione degli attori malevoli hanno prodotto un aumento sia del volume sia della sofisticazione degli attacchi, rendendo il rilevamento manuale delle intrusioni praticamente impossibile su scala operativa.

I rapporti di settore stimano che il numero di alert generati dai sensori di sicurezza in una rete di medie dimensioni superi le decine di migliaia al giorno [1]. Gran parte di questi alert è costituita da falsi positivi (segnali benigni classificati erroneamente come malevoli) o da rumore generato da attività lecita insolita: studi recenti documentano tassi di falsi positivi che raggiungono il 99% in contesti SOC reali [1]. Il risultato è un fenomeno noto come *alert fatigue*: gli analisti, sommersi da un volume di alert che non possono analizzare individualmente, sviluppano una soglia di attenzione crescente che porta a ignorare o posticipare l'analisi anche di segnali genuinamente critici [4].

A questa pressione volumetrica si aggiunge la crescente asimmetria tra attaccanti e difensori. Le Advanced Persistent Threat, campagne di attacco condotte da attori sponsorizzati da stati o da organizzazioni criminali strutturate, si caratterizzano per la capacità di rimanere latenti nelle infrastrutture bersaglio per settimane o mesi, modificando continuamente le proprie tattiche, tecniche e procedure (TTP) per eludere i meccanismi di rilevamento esistenti. L'evoluzione delle TTP comporta che le firme e le regole di rilevamento basate su pattern statici abbiano una vita utile sempre più breve: ciò che è rilevabile oggi potrebbe non esserlo più tra sei mesi, semplicemente perché l'avversario ha modificato il proprio comportamento.

In questo contesto, il Machine Learning ha assunto un ruolo centrale nella ricerca sulla sicurezza informatica come strumento per automatizzare il rilevamento di pattern anomali e classificare le

sessioni di traffico di rete in modo scalabile [3]. Tuttavia, come si argomenterà nel corso di questa tesi, l'adozione del Machine Learning non risolve il problema fondamentale dell'evoluzione delle minacce: lo sposta da un problema di aggiornamento delle regole a un problema di aggiornamento dei modelli.

1.2 Il ruolo critico dei Security Operations Center (SOC) nella difesa proattiva

Il Security Operations Center è la struttura organizzativa e tecnologica deputata al monitoraggio continuo della sicurezza informatica di un'organizzazione. Secondo il framework *PPTGC* (People, Process, Technology, Governance, Compliance) [18], un SOC efficace integra tre dimensioni inscindibili: le persone (analisti specializzati con competenze di threat intelligence e incident response), i processi (procedure operative standardizzate per la gestione degli alert e degli incidenti) e la tecnologia (gli strumenti di raccolta, correlazione e analisi dei log di sicurezza).

Tra gli strumenti tecnologici, i Network Intrusion Detection System basati su regole, come Suricata, svolgono un ruolo fondamentale: analizzano il traffico di rete in tempo reale e generano alert ogni volta che viene rilevata una sequenza di pacchetti che corrisponde a una firma di attacco nota. Suricata, in particolare, è adottato da un numero crescente di Managed Security Service Provider grazie alla sua architettura multithreaded, alla compatibilità con il ruleset Emerging Threats e al formato di output EVE JSON, che facilita l'integrazione con piattaforme SIEM e pipeline di analisi automatizzata.

Nonostante l'efficacia dei sensori, il processo di *triage*, ovvero la valutazione di criticità di ogni alert e la sua assegnazione alle priorità di risposta corrette, rimane prevalentemente manuale e costituisce il principale collo di bottiglia operativo del SOC. Un sistema di triage assistito da ML riceve come input sequenze di alert generate da sensori, le aggrega in unità di analisi contestualizzate (le *Alarm Chains*) e produce una classificazione binaria (benigno/malevolo) che orienta il lavoro degli analisti, riducendo il volume di sessioni da esaminare manualmente e concentrando l'attenzione sulle più probabili minacce reali. Lavori precedenti hanno dimostrato la fattibilità di questo approccio su dati operativi reali, ottenendo riduzioni significative del volume di allerte da revisionare con recall elevato per catene malevole.

1.3 Definizione del problema: obsolescenza dei modelli di Machine Learning e il Concept Drift

Il limite fondamentale dei sistemi ML-SOC esistenti, inclusi i lavori precedenti appena descritti, è di natura temporale. I modelli vengono addestrati una volta su un campione storico di dati etichettati e poi deployati in produzione come sistemi statici: il classificatore che valuta le sessioni di oggi è lo stesso che è stato addestrato su dati di mesi fa, senza alcun meccanismo che tenga conto dell'evoluzione della distribuzione dei dati.

Questo approccio ignora un fenomeno che la letteratura di Machine Learning chiama *concept drift*: il cambiamento, nel tempo, della relazione statistica tra le feature osservate e la variabile target [14]. Nel dominio della sicurezza informatica, il concept drift si manifesta attraverso almeno tre meccanismi concorrenti:

- Le campagne di attacco evolvono: nuovi malware, nuove varianti di exploit, nuovi pattern comportamentali dei botnet;
- I ruleset Suricata vengono aggiornati: nuove regole vengono aggiunte e vecchie modificate, cambiando la distribuzione delle categorie di alert;
- L'infrastruttura del cliente evolve: nuove applicazioni, modifiche alla topologia di rete, variazioni nel comportamento degli utenti alterano il profilo di traffico benigno;

Il risultato è che un modello di ML-SOC accurato al momento del deployment tende a degradare progressivamente nel tempo, fino a diventare inutilizzabile nell'arco di pochi mesi. Come dimostra questa tesi su dati reali, sei degli otto clienti analizzabili mostrano una perdita dell'F1-score superiore al 98% tra il periodo di training e il mese di ottobre, a distanza di soli sette mesi dall'inizio della raccolta dati. Questo dato quantifica in modo inequivocabile la distanza tra le prestazioni misurate in laboratorio, su cui si basano le pubblicazioni scientifiche del settore, e le prestazioni ottenibili in un sistema realmente operativo.

La letteratura ha affrontato il problema del concept drift nel dominio della sicurezza attraverso approcci diversi: il rilevamento del drift tramite valutatori conformali (Transcend [11]), la robustezza tramite apprendimento attivo e pseudo-labeling (INSOMNIA [2]), e la modellazione del

rilevamento del malware come flusso evolutivo di dati (Fast & Furious [6]). Tuttavia, nessuno di questi lavori combina la valutazione temporalmente corretta con una pipeline di aggiornamento continuo su dati operativi reali provenienti da un NIDS basato su Suricata in un contesto SOC multi-cliente.

Questo è precisamente il gap che la presente tesi intende colmare: la combinazione di una valutazione rigorosa nel tempo con un meccanismo di adattamento continuo, validata su un dataset operativo reale multi-cliente.

1.4 Obiettivi della tesi

La tesi si propone di raggiungere due obiettivi principali.

Obiettivo 1: analisi empirica del concept drift Misurare in modo rigoroso e quantitativo la degradazione delle prestazioni di un sistema ML-SOC nel tempo su un dataset operativo reale, adottando il protocollo di valutazione temporalmente corretto prescritto da TESSERACT e la metrica Area Under Time come indice sintetico di resilienza. La documentazione del drift su dati reali costituisce una precondizione scientifica necessaria: prima di proporre strategie di mitigazione, occorre dimostrare che il fenomeno esiste, è misurabile e ha un'entità operativamente rilevante.

Obiettivo 2: progettazione e confronto di strategie adattive Implementare e confrontare tre strategie di aggiornamento continuo del modello, il modello statico come baseline, l'apprendimento incrementale a finestra crescente (con classificatori RF e MLP) e la rolling window a dimensione fissa, misurandone il beneficio rispetto alla baseline sull'asse temporale. Le strategie devono essere operativamente semplici, non richiedere etichette in tempo reale e compatibili con un ciclo di retraining settimanale eseguibile in produzione senza interruzione del servizio.

A complemento di questi due obiettivi, la tesi include un'analisi esplorativa sull'ingegneria delle feature temporali: si valuta l'effetto della ponderazione esponenziale temporale degli eventi (Time Decay) sulle prestazioni del classificatore, verificando se l'informazione sul momento di occorrenza degli eventi all'interno della finestra di aggregazione migliori la separabilità tra classi benigna e malevola. Questa analisi non costituisce un obiettivo primario della tesi, ma un approfondimento metodologico a supporto dell'Obiettivo 2.

I due obiettivi vengono perseguiti su un dataset operativo reale fornito da un MSSP italiano, che comprende dieci clienti con profili di traffico eterogenei e un arco temporale di circa nove mesi (settembre 2024 - maggio 2025).

1.5 Organizzazione della tesi

Il documento è organizzato in sette capitoli, la cui struttura riflette la progressione logica dall'inquadramento teorico alla valutazione sperimentale.

Il **Capitolo 2** introduce il background teorico necessario: il funzionamento dei Security Operations Center, il ruolo di Suricata come NIDS, il problema dell'alert fatigue, il paradigma ML-SOC e, in particolare, la formalizzazione del concept drift e della sua tassonomia (sudden, gradual e incremental drift) con le relative evidenze empiriche nella letteratura di sicurezza.

Il **Capitolo 3** posiziona la tesi nel panorama della ricerca esistente, analizzando i due lavori fondativi precedenti, i sette contributi internazionali più pertinenti e chiarendo il gap che la tesi intende colmare.

Il **Capitolo 4** formalizza le scelte metodologiche: il threat model, la modellazione ad alto livello del SOC, l'architettura del sistema proposto e le tre componenti metodologiche originali (Alarm Chains, Time Decay e Incremental Learning con Rolling Window).

Il **Capitolo 5** descrive l'implementazione concreta del sistema: il dataset operativo con la sua distribuzione per cliente, la pipeline di preprocessing, l'ablation study sulla costruzione delle Alarm Chains, la selezione e il tuning dei classificatori (RF, HGB, MLP) e l'implementazione dei contributi sperimentali con relativi snippet di codice.

Il **Capitolo 6** presenta i risultati sperimentali secondo tre scenari progressivi: il pre-deployment (baseline statica), il post-deployment senza mitigazione (prova empirica del drift) e il post-deployment con mitigazione (confronto delle strategie adattive tramite AUT), chiudendo con un'analisi critica delle limitazioni del sistema.

Il **Capitolo 7** sintetizza i traguardi raggiunti, discute le implementazioni pratiche per l'integrazione in un SOC di produzione e delinea le principali direzioni di ricerca futura.

2. Background: Attacchi informatici e Security Operations Center

Il presente capitolo introduce le fondamenta concettuali su cui si costruisce l'intero lavoro di tesi. Partendo da un'analisi del panorama delle minacce informatiche contemporanee, si descrivono le strutture organizzative preposte alla loro rilevazione e gestione, cioè i *Security Operations Center*, con particolare attenzione alle componenti tecnologiche, alle criticità operative e all'evoluzione verso approcci basati su *machine learning*. Il capitolo si conclude esaminando il fenomeno del *concept drift*, ovvero il principale ostacolo alla sostenibilità a lungo termine dei sistemi di rilevamento automatico delle intrusioni in contesti reali.

2.1 Attacchi Informatici: panorama e tendenze evolutive

La sicurezza dei sistemi informatici è diventata, nel corso dell'ultimo decennio, una delle priorità strategiche per organizzazioni di qualsiasi dimensione e settore. La crescente digitalizzazione dei processi aziendali, la proliferazione di dispositivi connessi e la transizione verso le architetture cloud hanno ampliato enormemente la superficie di attacco a disposizione degli avversari, rendendo al contempo più articolate e difficili da individuare le tecniche offensive impiegate.

2.1.1 Tipologie di attacchi alle reti enterprise

Gli attacchi informatici si articolano in un'ampia gamma di tecniche, spesso combinate tra loro all'interno di campagne multi-stadio che mirano a compromettere riservatezza, integrità o disponibilità delle risorse bersaglio. Dal punto di vista tassonomico è utile distinguere alcune macrocategorie fondamentali.

Intrusioni di Rete Gli attacchi di rete sfruttano vulnerabilità nei protocolli di comunicazione o nelle configurazioni dei dispositivi perimetrali. Le tecniche più diffuse includono la scansione di porte o servizi esposti (*port scanning*), l'exploitation di vulnerabilità in servizi esposti su Internet,

gli attacchi di tipo *man-in-the-middle* (MitM) e gli attacchi volumetrici di negazione del servizio (*Distributed Denial of Service*, DDoS). Questi ultimi sfruttano botnet composte da migliaia di nodi compromessi per saturare la banda o le risorse computazionali del bersaglio.

Malware Il software malevolo rappresenta il vettore di attacco più trasversale. Le famiglie principali comprendono: *ransomware*, che cifra i dati della vittima richiedendo un riscatto; *trojan* ad accesso remoto (RAT), che aprono canali di comunicazione persistenti verso infrastrutture di comando e controllo (C2); *rootkit*, che alterano il sistema operativo per nascondere la propria presenza; ed *exploit kit*, framework automatizzati che identificano e sfruttano vulnerabilità nei browser e nei plugin delle vittime.

Phishing e ingegneria sociale Le campagne di phishing mirano a sottrarre credenziali o indurre l'utente a eseguire codice malevolo attraverso e-mail contraffatte, siti web fraudolenti o messaggi istantanei. La variante *spear phishing* prevede la personalizzazione del messaggio sulla base di informazioni pubblicamente disponibili sulla vittima, aumentando significativamente l'efficacia dell'inganno.

Attacchi alla supply chain Una tendenza in forte crescita è rappresentata dalla compromissione di componenti software o hardware nelle catene di fornitura. Anziché attaccare direttamente il bersaglio finale, l'avversario compromette un fornitore di fiducia - librerie open source, aggiornamenti firmware, piattaforme di distribuzione software - propagando il codice malevolo a tutti i clienti downstream. Casi emblematici come *SolarWinds* (2020) e *XZ Utils* (2024) hanno dimostrato la portata sistemica di questa strategia.

Minacce interne Le minacce interne (*insider threat*) originano da soggetti che dispongono già di accesso legittimo alle risorse dell'organizzazione: dipendenti malintenzionati, collaboratori distratti o account compromessi. La loro rilevazione è particolarmente complessa poiché il comportamento malevolo può essere difficilmente distinguibile dall'attività legittima sulla base dei soli metadati di rete.

La superficie di attacco si è ulteriormente ampliata con l'adozione di paradigmi architetturali quali il *cloud computing*, che sposta i confini del perimetro tradizionale, e l'Internet of Things (IoT),

che introduce in rete dispositivi con capacità di elaborazione limitate e cicli di aggiornamento del firmware spesso insufficienti. Il lavoro da remoto, accelerato dalla pandemia di COVID-19, ha inoltre moltiplicato i punti di accesso VPN e ha portato i dispositivi personali - spesso meno protetti - all'interno del perimetro logico aziendale.

2.1.2 Tendenze e dati statistici sul panorama delle minacce

L'analisi dei dati empirici sul panorama delle minacce rivela una tendenza preoccupante in termini di frequenza, complessità e impatto degli incidenti di sicurezza.

Secondo lo studio sistematico di Vielberth et al. [18], il numero di violazioni della sicurezza ha registrato un incremento dell'11% tra il 2017 e il 2018, passando da 130 a 145 incidenti rilevati nelle organizzazioni analizzate. Questi dati rispecchiano una tendenza globale: *Data Breach Investigations Report* annuale di Verizon documenta costantemente l'aumento degli incidenti sia in termini numerici sia di sofisticazione.

Uno degli indicatori più critici rimane il *tempo di rilevamento* (*Mean Time to Detect*, MTTD). Vielberth et al. [18] riportano che il tempo medio di rilevamento di un incidente si attesta intorno ai 196 giorni, a cui si aggiungono circa 69 giorni per il contenimento. Questo significa che, in media, un attaccante trascorre quasi nove mesi all'interno dell'infrastruttura bersaglio prima che la sua presenza venga scoperta, disponendo di tutto il tempo necessario per esfiltrare dati, stabilire meccanismi di persistenza e propagarsi lateralmente.

L'impatto economico degli incidenti è altrettanto significativo. Il caso Target del 2013 è emblematico: nonostante l'organizzazione disponesse di un sistema di monitoraggio FireEye, il cui costo ammontava a 1,6 milioni di dollari, gli avvisi generati dal sistema non furono adeguatamente gestiti dagli analisti SOC, permettendo l'esfiltrazione dei dati di circa 40 milioni di carte di credito [1]. Questo episodio illustra chiaramente che il problema non risiede esclusivamente nella capacità di rilevamento tecnologica, ma anche nella gestione umana e organizzativa degli avvisi.

Il panorama delle minacce è inoltre caratterizzato da una continua evoluzione delle tecniche offensive. I gruppi APT (*Advanced Persistent Threat*), spesso riconducibili ad attori statali o a organizzazioni criminali altamente specializzate, adattano costantemente le proprie *TTP* (Tattiche, Tecniche e Procedure) per eludere le difese esistenti. Questo fenomeno di adattamento dinamico è alla base del problema del *concept drift* che verrà analizzato in dettaglio nella Sezione 2.3.

Sul piano dell'automazione offensiva, strumenti di *evasion* basati su reti generative avversariali (GAN) e tecniche di *adversarial machine learning* sono stati proposti in letteratura come metodo per aggirare classificatori ML-based [3]. Sebbene il loro impiego operativo rimanga per ora limitato, il trend suggerisce che i sistemi di rilevamento dovranno fare i conti, in futuro, non solo con la deriva naturale dei dati, ma anche con avversari capaci di manipolare deliberatamente le distribuzioni di input.

2.2 Il Security Operations Center come prima linea di difesa

Il Security Operations Center (SOC) è la struttura organizzativa e tecnologica preposta al monitoraggio continuativo dell'infrastruttura IT di un'organizzazione, al rilevamento delle minacce e alla gestione degli incidenti di sicurezza. La sua definizione, composizione e funzionamento sono stati oggetto di sistematizzazione nella letteratura accademica, con contributi significativi che ne hanno analizzato i processi, i ruoli e le sfide operative.

2.2.1 Architettura e flusso operativo: Ingestion, Detection, Triage e Incident Response

Vielberth et al. [18] propongono una revisione sistematica della letteratura sui SOC, identificando cinque dimensioni fondamentali che ne caratterizzano l'operatività: *Persone* (People), *Processi* (Process), *Tecnologia* (Technology), *Governance* e *Compliance* (Compliance), sintetizzate nell'acronimo **PPTGC** (*People, Process, Technology, Governance, Compliance*).

Persone Il capitale umano rappresenta il fattore più critico all'interno di un SOC. La struttura tipica prevede una gerarchia di analisti organizzati su tre livelli:

- **Livello 1 - Triage Analyst**: si occupa del monitoraggio in tempo reale delle dashboard di sicurezza, della classificazione iniziale degli avvisi generati dagli strumenti automatici e dell'escalation degli eventi rilevanti ai livelli superiori;

- **Livello 2 - Incident Response**: conduce indagini approfondite sugli incidenti escalati dal livello 1, raccoglie evidenze forensi, correla eventi provenienti da sorgenti multiple e coordina le attività di contenimento;

- **Livello 3 - Threat Hunter**: svolge attività proattive di ricerca di minacce latenti (*threat hunting*), analizza malware, sviluppa nuove regole di rilevamento e mantiene aggiornata la base di conoscenza sulle TTP degli avversari;

Processi I processi di un SOC coprono l'intero ciclo di vita della gestione degli incidenti: dalla raccolta e normalizzazione degli eventi di log, passando per il triage, l'investigazione e il contenimento, fino alla fase di *post-incident review* e all'aggiornamento delle politiche di sicurezza. La standardizzazione dei processi attraverso playbook operativi è fondamentale per garantire la coerenza delle risposte, specialmente nelle fasi di turnover del personale.

Tecnologia Il nucleo tecnologico di un SOC moderno ruota attorno a una piattaforma di *Security Information and Event Management* (SIEM), che aggrega log provenienti da firewall, endpoint detection & response (EDR), intrusion detection system (IDS), server applicativi e qualsiasi altra sorgente rilevante. A questi si affiancano strumenti di *Security Orchestration, Automation and Response* (SOAR), che automatizzano le azioni di risposta ripetitive, e piattaforme di *Threat Intelligence* (TI), che forniscono contesto sugli indicatori di compromissione (IoC) e sulle TTP degli attori di minaccia noti.

Governance La governance definisce le politiche di sicurezza, i KPI operativi, quali MTTD e *Mean Time To Respond* (MTTR), le procedure di escalation e il quadro di conformità normativa (GDPR, NIS2, ISO 27001).

Organizzazione Dal punto di vista organizzativo, i SOC possono essere interni (*in-house*), esternalizzati presso fornitori di servizi gestiti (*Managed Security Service Provider*, MSSP) o ibridi. La scelta dipende da fattori quali la dimensione dell'organizzazione, la sensibilità dei dati trattati e le risorse disponibili.

2.2.2 Strumenti di monitoraggio: SIEM, EDR e NIDS - focus su Suricata

All'interno dello stack tecnologico di un SOC, i sistemi di *Network Intrusion Detection* (NIDS) svolgono un ruolo fondamentale: analizzano il traffico di rete in tempo reale alla ricerca di pattern riconducibili ad attività malevole.

Architettura dei NIDS I NIDS operano tipicamente in modalità *passive tap* o *inline*, intercettando una copia del traffico di rete su segmenti strategici dell'infrastruttura (perimetro, segmenti, interni critici e traffico nei datacenter). L'analisi del traffico può avvenire con due approcci complementari:

- **Signature-based detection**: confronta il payload o i metadati dei pacchetti con una base di regole che codificano pattern noti di attività malevola. Offre alta precisione per minacce conosciute, ma è cieco di fronte a varianti nuove o a zero-day;
- **Anomaly-based detection**: stabilisce un modello del comportamento normale del traffico e segnala deviazioni statisticamente significative. Potenzialmente in grado di rilevare attacchi sconosciuti, ma soggetto a elevati picchi di falsi positivi;

Suricata Suricata è un motore NIDS/NIPS open source ad alte prestazioni, sviluppato e mantenuto dalla *Open Information Security Foundation* (OISF) [12]. Supporta ispezione multi-thread del traffico, analisi dei protocolli applicativi (HTTP, DNS, TLS, SMB) e generazione di log in formato JSON strutturato (*EVE JSON*). Il sistema di regole di Suricata è compatibile con il formato Snort e viene distribuito attraverso diversi *ruleset*: il più diffuso in ambito open-source è *Emerging Threat Open* (ET Open), aggiornato quotidianamente dalla community e contenente migliaia di firme per le minacce più recenti.

Ogni regola Suricata è composta da:

- Un'intestazione che specifica l'azione (`alert`, `drop`, `pass`), il protocollo, gli indirizzi e le porte sorgente/destinazione;
- Un corpo di opzioni che include il messaggio descrittivo (`msg`), il pattern di ricerca nel payload (`content`, `pcre`), i metadati di classificazione (`classtype`, `sid`, `rev`);

Nell'architettura del sistema descritto in questa tesi, Suricata è il componente che produce gli eventi di sicurezza grezzi. Gli avvisi generati, ciascuno associato al messaggio testuale della regola che lo ha innescato, vengono aggregati in *catene di allarmi* e sottoposti a una pipeline di analisi basata su tecniche di *machine learning*, come descritto nel Capitolo 4.

2.2.3 Alert Fatigue: cause, impatto cognitivo e conseguenze operative

Il volume di avvisi generato dai sistemi di monitoraggio in un SOC reale è tale da superare largamente la capacità di analisi del personale disponibile. Questo fenomeno, noto come *alert fatigue*, è uno dei problemi operativi più critici e documentati nella gestione dei SOC moderni.

Dimensione del problema I dati empirici sulla prevalenza dell'alert fatigue sono eloquenti. La Cloud Security Alliance stima che oltre la metà delle imprese utilizzi più di sei strumenti differenti, ciascuno capace di generare decine o centinaia di migliaia di avvisi al giorno [4]. Ban et al. [4] riportano che il 31,9% degli analisti di sicurezza ignora sistematicamente gli avvisi a causa del sovraccarico dovuto ai falsi positivi.

Lo studio qualitativo di Alahmadi et al. [1] fornisce una prospettiva ancora più sfumata, derivata da interviste a 20 professionisti di 7 SOC reali. L'analisi rivela che la maggior parte degli avvisi classificati come "falsi positivi" dagli analisti non sono, in senso stretto, errori del sistema di rilevamento: sono piuttosto *benign triggers*, ovvero segnalazioni corrette di comportamenti che il sistema ha identificato come potenzialmente malevoli ma che, nella specifica realtà operativa dell'organizzazione, sono associati ad attività legittime.

Ad esempio, uno strumento di amministrazione remota usato dal team IT può generare avvisi di intrusione perché il NIDS non conosce il contesto operativo in cui opera. Dal punto di vista del modello di rilevamento l'avviso è corretto, ma nella pratica operativa è un segnale rumoroso che distrae l'analista dalle minacce reali.

Conseguenze operative L'alert fatigue produce conseguenze concrete e misurabili:

- **Latenza nella risposta**: gli analisti impiegano più tempo a prioritizzare gli avvisi critici, aumentando il MTTD;

- **Insensibilità agli avvisi**: la desensibilizzazione psicologica porta gli analisti a ignorare interi cluster di avvisi, con il rischio di perdere eventi genuinamente malevoli;
- **Turnover del personale**: la pressione cognitiva e la frustrazione contribuiscono all'elevato tasso di abbandono che caratterizza il ruolo di analista L1 [1];

Il caso Target (2013) rimane l'esempio più citato delle conseguenze catastrofiche di una gestione inadeguata degli avvisi: il sistema FireEye generò notifiche corrette sull'attività malevola, ma la combinazione di volume eccessivo di allarmi e procedure operative inadeguate portò gli analisti a non dare seguito alle segnalazioni, con una perdita stimata di miliardi di dollari [1].

Il framework REACT Per affrontare strutturalmente il problema, Alahmadi et al. [1] propongono il framework **REACT** (*Relevant, Enriched, Actionable, Contextualized, Triaged*), che definisce cinque requisiti qualitativi che un avviso deve soddisfare per essere effettivamente utile all'analista:

1. **Relevant**: l'avviso deve essere pertinente al contesto operativo specifico dell'organizzazione;
2. **Enriched**: l'avviso deve essere arricchito con informazioni di contesto (reputazione, IP, dati WHOIS, correlazione IoC noti);
3. **Actionable**: l'avviso deve suggerire o abilitare azioni di risposta concrete;
4. **Contextualized**: deve essere interpretato in relazione al comportamento storico dell'entità che lo ha generato;
5. **Triaged**: l'avviso deve essere prioritizzato rispetto agli altri avvisi sulla base del rischio effettivo;

Questi requisiti definiscono implicitamente le caratteristiche che un sistema di triage automatizzato basato su ML dovrebbe possedere per essere davvero efficace in un SOC reale.

2.2.4 ML-SOC: verso l'AI-Assisted Triage

L'impiego del machine learning per il supporto alle attività dei SOC è un'area di ricerca in rapida evoluzione, motivata dalla necessità di automatizzare il triage di avvisi e ridurre il carico cognitivo sugli analisti.

Applicazioni del ML nei SOC Il paradigma *ML-SOC* descritto in questa tesi si inquadra nella categoria del triage automatizzato. Il sistema riceve in ingresso sequenze di avvisi Suricata aggregati in catene temporali, le trasforma in rappresentazioni vettoriali mediante tecniche di *term frequency-inverse document frequency* (TF-IDF) applicate ai messaggi testuali delle regole, e produce in uscita una classificazione binaria (benigno/malevolo) per ogni catena. I classificatori impiegati (Random Forest, HistGradientBoosting e Multi-Layer Perceptron) sono addestrati su dati storici etichettati dagli analisti del SOC.

Ban et al. [4] dimostrano la fattibilità di questo approccio su dati reali, raggiungendo un tasso di richiamo (*recall*) del 99,598% per gli avvisi critici e un tasso di falsi positivi dello 0,001%, a conferma del potenziale del ML nel ridurre il carico di lavoro manuale senza compromettere la copertura delle minacce.

Il divario tra ricerca e deployment Nonostante i risultati promettenti in ambiente controllato, Apruzzese et al. [3] documentano un significativo divario tra le prestazioni dei modelli ML riportate in letteratura e quelle osservate nel deployment operativo. Le cause principali di questo divario sono molteplici:

- **Dataset non rappresentativi**: i dataset pubblici di sicurezza informatica sono spesso ottenuti in ambienti di laboratorio, con distribuzioni del traffico lontane da quelle reali;
- **Concept Drift**: la distribuzione dei dati reali cambia nel tempo in modo imprevedibile, degradando le prestazioni dei modelli addestrati su dati storici;
- **Bias sperimentali**: l'uso di dati temporalmente consecutivi senza adeguata separazione tra training e test introduce ottimismo artificiale nelle valutazioni;
- **Target sparsity**: in dataset reali le istanze malevole rappresentano una frazione minuscola del totale, rendendo addestramento e valutazione molto complessi;

Questi problemi sono interconnessi e richiedono soluzioni che vadano oltre la semplice ottimizzazione del modello, coinvolgendo la progettazione della pipeline di addestramento, la strategia di aggiornamento del modello e la metrica di valutazione adottata. Il prossimo capitolo (Stato dell'Arte) analizzerà in dettaglio i contributi della letteratura a ciascuna di queste sfide; la Sezione 2.3 che segue introduce invece il più fondamentale di questi problemi: il concept drift.

2.3 Il Concept Drift nei sistemi ML-SOC: impatto nel tempo

Nei sistemi di rilevamento automatico delle intrusioni basati su machine learning, la robustezza a lungo termine non dipende soltanto dalla qualità del modello al momento dell'addestramento, ma dalla sua capacità di mantenere prestazioni accettabili man mano che il mondo reale evolve. Il fenomeno che governa questa degradazione temporale è noto in letteratura con due termini spesso usati in modo intercambiabile, *data drift* e *concept drift*, ma che, come si vedrà, designano aspetti distinti del medesimo problema [14, 8].

2.3.1 Formalizzazione: Data Drift vs Concept Drift

Sia $f : \mathcal{X} \to \mathcal{Y}$ un classificatore addestrato su un insieme di campioni (x_i, y_i) estratti da una distribuzione congiunta $P_t(X, Y)$ valida al tempo t. In un sistema ML-SOC, X è la rappresentazione vettoriale di una catena di allarmi Suricata e $Y \in \{0, 1\}$ la sua etichetta (benigno/malevolo).

Il termine **data drift** (o *covariate shift*) indica un cambiamento della distribuzione marginale degli input:

$$P_{t'}(X) \neq P_t(X), \quad \text{con } P_{t'}(Y \mid X) = P_t(Y \mid X) \tag{2.1}$$

La relazione tra feature e classe rimane invariata, ma cambiano i pattern osservati. In ambito SOC questo si verifica, ad esempio, quando l'organizzazione adotta nuove applicazioni o modifica la propria infrastruttura, alterando la distribuzione del traffico legittimo senza che cambi la natura delle minacce. Il modello continua ad essere *corretto* nella logica di classificazione, ma può trovarsi a operare su input fuori distribuzione rispetto al training, con conseguente incertezza nelle predizioni.

Il **concept drift** in senso stretto indica invece un cambiamento della funzione di classe condizionale:

$$P_{t'}(Y \mid X) \neq P_t(Y \mid X) \tag{2.2}$$

Lo stesso pattern di traffico o la stessa sequenza di allarmi può corrispondere a una classe diversa nel tempo. Un esempio tipico in ambito SOC è l'adozione di un nuovo strumento di amministrazione remota da parte del team IT: i pattern di connessione associati, inizialmente segnalati come

potenzialmente malevoli, diventano legittimi una volta che il contesto operativo cambia. Dal punto di vista del modello, la regola di decisione appresa è diventata obsoleta.

Una terza componente è il **label drift**, ovvero il cambiamento della distribuzione marginale delle etichette $P(Y)$ indipendentemente dalla struttura condizionale. Nel contesto di un SOC reale, il label drift è frequente: periodi di intensa attività offensiva (campagne malware, attacchi mirati) aumentano transitoriamente la prevalenza delle istanze malevole, mentre in periodi di calma il rapporto tra benigni e malevoli può essere estremamente sbilanciato (*target sparsity*). Questa variabilità rende instabili le soglie di classificazione e le metriche di valutazione calcolate su singoli snapshot temporali [3].

In sintesi, nella letteratura sulla sicurezza informatica i due termini vengono spesso sovrapposti perché in pratica i tre fenomeni coesistono e si intrecciano. Nella presente tesi si adotta la convenzione per cui il termine *concept drift* ha accezione generale, comprensiva di covariate shift, concept drift stretto e label drift, mentre il termine *data drift* è riservato alla sola componente di covariate shift, in linea con l'uso prevalente nella letteratura di riferimento [14].

2.3.2 Tassonomia: Sudden, Gradual e Incremental Drift in ambito SOC

La modalità con cui il drift si manifesta nel tempo ha implicazioni dirette sulla strategia di monitoraggio e mitigazione più adeguata. La letteratura distingue quattro archetipi principali [8, 14].

Sudden Drift La distribuzione cambia bruscamente in un istante ben definito: prima e dopo quel punto il processo generativo è stazionario, ma le due distribuzioni sono sostanzialmente diverse. In ambito SOC questo scenario è associato ad eventi discreti e identificabili: l'emersione di una nuova campagna di attacco con TTP differenti di quelle precedenti, la pubblicazione di un exploit per una vulnerabilità zero-day, o una significativa modifica della configurazione di rete. Il sudden drift è il più semplice da rilevare, cioè un test statistico sul flusso di predizioni o sulle feature che può identificare il punto di rottura, ma richiede una risposta rapida, poiché le prestazioni del modello degradano immediatamente dopo il cambiamento.

Gradual Drift La transizione dalla vecchia alla nuova distribuzione avviene in modo progressivo: per un periodo di tempo le due distribuzioni coesistono con probabilità variabile, fino a quando quella nuova sostituisce completamente quella vecchia. È il tipo di drift più comune nei SOC operativi, poiché riflette l'evoluzione graduale delle TTP degli avversari e dei pattern del traffico legittimo. Difficile da rilevare tempestivamente perché la degradazione delle prestazioni è lenta e inizialmente indistinguibile dal rumore statistico, richiede meccanismi di monitoraggio continuo su finestre temporali sufficientemente ampie.

Incremental Drift La distribuzione si sposta monotonicamente lungo una direzione dello spazio delle feature, senza ritorno ai valori precedenti. In un contesto di rete, questo può riflettere l'adozione progressiva e irreversibile di nuovi protocolli (es. la migrazione da HTTP a HTTPS, o da IPv4 a IPv6), che altera strutturalmente le feature estratte dagli avvisi Suricata. A differenza del gradual drift, non c'è un punto in cui la transizione si completa: il processo è continuo.

Recurring concepts Pattern distributivi già osservati in passato riemergono periodicamente. Nei SOC questo fenomeno è associato alla stagionalità del traffico di rete, poiché, per esempio, i periodi di chiusura aziendale presentano distribuzioni anomale rispetto ai giorni lavorativi. Un modello addestrato prevalentemente su dati di un determinato periodo ciclico può degradare in modo sistematico sui periodi complementari.

Implicazioni per il sistema ML-SOC Nella pratica operativa di un SOC, il drift osservato è tipicamente una combinazione di questi archetipi che operano simultaneamente su scale temporali diverse. Questa eterogeneità rende inadeguate le strategie di mitigazione basate su un unico meccanismo e motiva la ricerca di approcci adattativi, come quelli analizzati in questa tesi.

2.3.3 Evidenze empiriche del degrado nei sistemi di detection reali

Le implicazioni del concept drift non sono puramente teoriche: la letteratura empirica documenta in modo consistente una degradazione significativa delle prestazioni dei sistemi di rilevamento automatico nel tempo, con conseguenze operative concrete.

Il framework TESSERACT e la metrica AUT Il contributo più influente sulla quantificazione del degrado temporale nei sistemi ML-based è quello di Pendlebury et al. [14], che introduce il framework **TESSERACT** e la metrica **AUT** (*Area Under Time*).

Il framework identifica due forme di bias sperimentale sistematico che portano a sovrastimare le prestazioni dei classificatori nella letteratura:

1. **Temporal bias**: i dataset di training e test vengono mescolati senza rispettare l'ordinamento cronologico, cosicché il modello viene valutato sui dati "precedenti" rispetto a parte del suo training. In un deployment reale il modello opera sempre su dati futuri, rendendo questa configurazione di valutazione intrinsecamente ottimistica;

2. **Spatial bias**: la distribuzione del dataset non riflette quella reale di deployment, ad esempio per effetto di bilanciamento artificiale delle classi o di campionamento non rappresentativo;

Per misurare la robustezza di un classificatore rispetto al degrado temporale, viene introdotta la metrica AUT, definita come l'area sotto la curva di performance nel tempo calcolata con il metodo dei trapezi su N finestre temporali successive:

$$\mathrm{AUT}(f, N) = \frac{1}{N-1} \sum_{k=1}^{N-1} \frac{f_(x_{k+1}) + f(x_k)}{2} \tag{2.3}$$

dove $f(x_k)$ è il valore della metrica di performance (tipicamente F_1-score) sulla k-esima finestra di test. Un classificatore perfettamente stabile nel tempo avrebbe AUT = 1; un modello che degrada rapidamente produce valori di AUT significativamente inferiori, anche qualora la sua performance al momento dell'addestramento fosse elevata. L'AUT è la metrica di riferimento adottata nella presente tesi per confrontare le diverse strategie di retraining e time decay (Capitolo 5).

Evidenze quantitative Applicando TESSERACT su un dataset di 129.000 applicazioni Android (da gennaio 2014 a dicembre 2016), Pendlebury et al. [14] mostrano che la performance dei migliori classificatori valutati con la metodologia standard (temporal biased) degrada fino al 50% quando ri-valutati in configurazione realistica. Questo risultato, replicato su diversi dataset e architetture, ha evidenziato che gran parte dei risultati pubblicati sulla detection di malware era affetta da ottimismo sistematico, e ha promosso l'adozione di protocolli di valutazione temporalmente corretti come standard de facto.

Apruzzese et al. [3] confermano questo scenario in un contesto più ampio: il divario tra le prestazioni riportate in letteratura e quelle osservate nel deployment operativo è una delle cause principali della lentezza nell'adozione di soluzioni ML nei SOC in produzione.

Sfide specifiche del contesto ML-SOC Oltre alla degradazione intrinseca documentata da TESSERACT, i sistemi ML-SOC presentano ulteriori fattori che amplificano l'impatto del concept drift:

- **Target sparsity**: in un dataset operativo reale la frazione di istanze malevole è tipicamente inferiore all'1%. Piccole variazioni assolute nel numero di veri positivi producono fluttuazioni amplissime nelle metriche standard come F_1-score e precision, rendendo difficile distinguere il segnale del drift dal rumore statistico;
- **Latenza nell'etichettatura**: le etichette di ground truth sono prodotte dagli analisti con un ritardo variabile rispetto all'arrivo degli eventi. Questa latenza introduce un disallineamento temporale tra i dati disponibili per il retraining e i dati su cui il modello opera in produzione, rallentando la velocità di adattamento;
- **Natura locale del drift**: il concept drift in un SOC ha una componente fortemente client-specific. Ogni organizzazione ha pattern di traffico, applicazioni e politiche di sicurezza propri; il drift osservato su un cliente non è generalizzabile ad altri, il che impedisce strategie di mitigazione basate su distribuzioni aggregate e richiede approcci personalizzati per ciascun contesto;
- **Supervisione parziale e ritardata**: i sistemi ML-SOC operano tipicamente in modalità *human-in-the-loop*. Il feedback degli analisti, che costituisce il segnale di supervisione per il retraining, è incompleto (non tutti gli eventi vengono revisionati) e rumoroso (la qualità dell'etichettatura dipende dall'esperienza e dal carico di lavoro dell'analista). Ne risulta un problema di *learning sotto distribuzione non stazionaria con supervisione parziale e ritardata*, per il quale le soluzioni standard della letteratura richiedono adattamenti non banali;

È in risposta a queste sfide che la presente tesi propone e valuta strategie di retraining continuo e time decay esponenziale: meccanismi progettati per mantenere il modello allineato alla distribuzione

corrente dei dati, contenendo al contempo il costo computazionale del ciclo di aggiornamento. Il Capitolo 3 analizzerà in dettaglio i lavori della letteratura che hanno affrontato questi problemi; il Capitolo 4 descriverà le soluzioni adottate nel sistema sviluppato.

3. Stato dell'Arte e Lavori Correlati

Il presente capitolo colloca il lavoro di tesi nel panorama della ricerca esistente, seguendo una struttura a due livelli. Nella prima parte si analizzano i lavori precedenti sviluppati all'interno della stessa linea di ricerca, identificando il contributo che la presente tesi intende apportare rispetto a quella base. Nella seconda parte si esaminano tre contributi della letteratura internazionale che affrontano problemi affini, ovvero il degrado temporale dei classificatori ML in ambito sicurezza, la robustezza al concept drift nei sistemi di rilevamento delle intrusioni e la modellazione del rilevamento di malware con flusso di dati evolutivo, discutendone approcci, punti di forza e limitazioni rispetto all'obiettivo di questa tesi. Il capitolo si chiude con un'analisi del posizionamento del presente lavoro rispetto alla letteratura, chiarendo il gap che si intende colmare.

3.1 Fondamenta di questo lavoro

Il sistema ML-SOC analizzato in questa tesi non nasce dal nulla: si inserisce in una linea di ricerca consolidata in questo ambito e, più in generale, in letteratura internazionale.

3.1.1 Analisi dei precedenti lavori

Alert fatigue e triage assistito da ML Questo lavoro affronta il problema dell'alert fatigue in un SOC reale, proponendo un sistema di triage automatizzato basato su classificatori supervisionati addestrati su catene di allarmi Suricata. Il sistema riceve in ingresso sequenze di avvisi aggregati per sessione di rete, le vettorizza mediante TF-IDF applicato ai messaggi testuali delle regole Suricata e produce una classificazione binaria (benigno/malevolo) per ogni catena. Sono state confrontate tre famiglie di classificatori: Random Forest (RF), HistGradientBoosting (HGB) e Multi-Layer Perceptron (MLP).

A questa pipeline di classificazione si affianca un modulo di *Explainability* basato su SHAP (*SHapley Additive exPlanations*), che fornisce agli analisti una spiegazione delle feature più influenti per ogni decisione, rendendo il sistema interpretabile e utile nel workflow operativo del SOC.

I risultati sperimentali documentano un'elevata capacità di riduzione del volume di avvisi da revisionare mantenendo recall elevato per le catene malevole.

Il lavoro stabilisce la struttura della pipeline (dalla raccolta degli alert alla classificazione delle catene) e dimostra la fattibilità dell'approccio ML-SOC su dati operativi reali. Tuttavia, il modello viene addestrato una volta su dati storici e valutato in modo *statico*: non è previsto alcun meccanismo di aggiornamento nel tempo, né viene analizzato l'effetto del degrado della distribuzione dei dati sulla qualità del classificatore in produzione.

Generalizzazione cross-client Questo lavoro affronta un aspetto complementare: la portabilità di un modello ML-SOC tra clienti diversi dello stesso fornitore di sicurezza. L'ipotesi investigata è se un modello addestrato sull'infrastruttura di un cliente possa essere impiegato direttamente su un altro cliente, evitando il costo del retraining su dati locali.

I risultati mostrano che la generalizzazione cross-client è limitata: la variabilità nei pattern di traffico, nelle applicazioni adottate e nelle politiche di sicurezza proprie di ciascuna organizzazione determina una deriva contestuale che riduce significativamente le prestazioni del modello trasferito rispetto a un modello addestrato localmente. La conclusione principale è che il concept drift ha una componente fortemente *client-specific*: le strategie di adattamento devono essere calibrate sul contesto specifico di ogni cliente, non su distribuzioni aggregate.

Questo risultato ha implicazioni dirette per la presente tesi: se il drift è locale, le strategie di retraining devono operare su dati del singolo cliente e devono essere abbastanza agili da inseguire la deriva locale senza richiedere grandi finestre di dati storici condivisi.

3.1.2 Gap identificato: la dimensione temporale e il degrado nel tempo

Entrambi i lavori precedenti condividono una limitazione strutturale: la valutazione è *atemporale*. I modelli vengono addestrati su un periodo storico e valutati su un periodo successivo, ma questo avviene una sola volta, in configurazione statica. Non viene misurato come le prestazioni evolvano nel tempo a partire dal momento del deployment, né vengono proposti meccanismi per mantenere il modello aggiornato rispetto alla distribuzione corrente dei dati.

In termini operativi, un modello ML-SOC in produzione deve funzionare continuamente, su un flusso di dati la cui distribuzione cambia ogni settimana: nuove campagne di attacco, aggiornamenti

del ruleset Suricata, modifiche all'infrastruttura del cliente, turnover dagli utenti. Nessuno dei due lavori affronta questa dimensione temporale.

Il gap che questa tesi intende colmare è precisamente questo:

1. **Valutazione temporalmente corretta**: adottare la metrica AUT per valutare le prestazioni del sistema su finestre temporali successive, invece di un singolo snapshot statico;
2. **Strategie di aggiornamento continuo**: progettare e confrontare meccanismi di retraining con rolling window, ponderazione time decay esponenziale e apprendimento incrementale, misurando l'effetto di ciascuna strategia sulla stabilità delle prestazioni nel tempo;
3. **Validazione su dati operativi reali**: condurre la valutazione su un dataset prodotto da un SOC reale basato su Suricata, evitando i bias sperimentali tipici dei dataset pubblici di benchmark;

3.2 Lavori correlati in letteratura

La letteratura internazionale ha affrontato il problema del concept drift nei sistemi di sicurezza basati su ML attraverso approcci diversi, che si distribuiscono lungo uno spettro che va dalla *rilevazione del drift* al *continuo aggiornamento del modello*. Di seguito si esaminano sette contributi rappresentativi di questo panorama, selezionati per la loro rilevanza diretta rispetto al problema trattato in questa tesi.

3.2.1 Machine Learning in Security Operations Center

Sommer e Paxson (2010) - Le sfide del ML per l'intrusione detection operativa Il lavoro di Sommer e Paxson [16] costituisce un riferimento fondamentale per chiunque intenda applicare il Machine Learning in ambienti di rilevamento di intrusioni reali. Gli autori argomentano che, nonostante i risultati promettenti ottenuti in laboratorio, il trasferimento dei sistemi ML al contesto operativo è ostacolato da una serie di problemi strutturali che i dataset di benchmark non riproducono fedelmente. I fattori chiave identificati sono quattro:

1. **Estrema asimmetria delle classi**: in un ambiente reale, il traffico malevolo rappresenta una frazione minima del totale, rendendo il problema di classificazione intrinsecamente sbilanciato;

2. **Costo asimmetrico degli errori**: un falso negativo (intrusione non rilevata) ha conseguenze ben più gravi di un falso positivo (alert ingiustificato), ma i classificatori ML ottimizzano metriche simmetriche come l'accuratezza;

3. **Non-stazionarietà**: la distribuzione del traffico cambia continuamente, rendendo obsoleti i modelli addestrati su dati storici;

4. **Difficoltà di raccogliere ground truth affidabile**: in un SOC reale le etichette sono prodotte da analisti umani con latenza variabile e possono essere incomplete o rumorose;

Queste osservazioni, formulate nel 2010, rimangono pienamente valide nel contesto di questa tesi e motivano direttamente le scelte metodologiche adottate: la metrica AUT che privilegia la recall, il protocollo di valutazione temporalmente corretto e l'ancoraggio delle etichette al sistema di ticketing del SOC.

Veeramachaneni et al. (2016) - AI^2: aggiornamento continuo con feedback degli analisti Il lavoro di Veeramachaneni et al. [17] propone *AI*^{2} (*Analyst Intuition* + *Artificial Intelligence*), un sistema che combina un classificatore ML con un ciclo di feedback periodico da parte degli analisti di sicurezza. Il sistema analizza in automatico gli eventi di sicurezza, identifica i più anomali e li sottopone all'analista per la validazione; le etichette restituite vengono poi usate per aggiornare il modello in modo incrementale. L'aspetto più rilevante per la presente tesi è il meccanismo di aggiornamento: AI^2 non addestra il modello una sola volta, ma lo aggiorna periodicamente sulla base del feedback umano, inseguendo l'evoluzione della distribuzione dei dati nel tempo. I risultati sperimentali mostrano che, su tre mesi di log di sicurezza, il sistema riduce il numero di eventi da esaminare mantenendo un tasso di rilevamento elevato.

Rispetto a questa tesi, AI^2 condivide l'architettura fondamentale (ML + label prodotte dagli analisti + aggiornamento periodico), ma non affronta esplicitamente il concept drift come fenomeno da misurare e quantificare, né propone un protocollo di valutazione temporalmente corretto che separi il periodo di training da quello di test. Inoltre, il sistema è valutato su dati aziendali proprietari, rendendo difficile la comparazione diretta dei risultati.

Gelman et al. (2023) - TEQ: un framework ML per la prioritizzazione degli alert in un SOC reale Il lavoro di Gelman et al. [9] propone *TEQ* (*That Escalated Quickly*), un framework di

Machine Learning sviluppato da Sophos per ridurre l'alert fatigue negli analisti di SOC con un impatto minimo sul workflow operativo esistente. Il sistema produce due score di priorità: uno a livello di singolo alert (*alert-level actionability*) e uno a livello di incidente aggregato (*incident-level actionability*), consentendo di filtrare gli alert probabilmente benigni prima che raggiungano la coda di revisione degli analisti.

La valutazione è condotta su dati reali raccolti in collaborazione con un SOC nell'arco di otto mesi (gennaio-settembre 2022), mostrando una riduzione del tempo medio di risposta agli incidenti del 22.9%, con il 54% dei falsi positivi soppressi e un tasso di rilevamento del 95.1%. Questi risultati su dati operativi reali rendono TEQ il lavoro più direttamente paragonabile all'approccio della presente tesi in termini di dominio e obiettivi.

La limitazione principale rispetto a questa tesi è che TEQ tratta il classificatore come statico per l'intera finestra di valutazione degli otto mesi: non viene analizzato né mitigato il degrado temporale delle prestazioni, che è invece l'obiettivo centrale del presente lavoro. In altre parole, TEQ dimostra che un classificatore ML funziona bene al momento del deployment in un SOC reale, ma non risponde alla domanda su quanto a lungo continuerà a funzionare bene senza aggiornamento.

Ghadermazi et al. (2024) - ML e ottimizzazione per la gestione efficiente degli alert in un SOC. Il lavoro di Ghadermazi et al. [10] propone un framework integrato che combina tecniche di Machine Learning con metodi di ottimizzazione per migliorare l'efficienza del processo di gestione degli alert in un centro operativo di sicurezza. Il framework affronta il problema congiunto della classificazione degli alert (distinguendo quelli che richiedono intervento da quelli benigni) e dell'allocazione ottimale del carico di lavoro tra gli analisti disponibili, con l'obiettivo di minimizzare sia il tempo di risposta sia il numero di alert irrisolti.

Il contributo è pubblicato su *ACM Digital Threats: Research and Practice* (2024), una delle sedi di riferimento per la ricerca applicata alla sicurezza informatica operativa. La rilevanza per questa tesi è duplice: il lavoro conferma che il ML in contesto SOC deve essere valutato su metriche operative (tempo di risposta, carico degli analisti) e non solo su metriche di classificazione standard; inoltre, dimostra che la complessità del problema di triage supera la semplice classificazione binaria, aprendo la strada a ottimizzazioni che tengano conto dei vincoli organizzativi del SOC. Anche in questo caso, il framework è valutato in configurazione statica e non affronta il problema

dell'obsolescenza del modello nel tempo.

3.2.2 Transcend: rilevazione del drift per modelli di classificazione del malware

Approccio Transcend è il primo lavoro a proporre un meccanismo sistematico per rilevare *l'invecchiamento* di un classificatore ML per malware *in vivo*, ovvero durante il deployment e prima che le prestazioni degradino in modo percettibile. L'approccio si basa sulla teoria degli *stimatori conformali* (*conformal evaluators*): per ogni nuova predizione, il sistema calcola due score, *credibility* e *confidence*, che misurano quanto il campione da classificare sia statisticamente simile ai campioni del training set. Un campione con credibilità bassa indica che il modello si trova ad operare fuori dalla distribuzione di addestramento: è un segnale di drift potenziale.

Il framework non riaddestra il modello: si limita a segnalare quando il modello sta "invecchiando" e le sue predizioni non sono più affidabili, lasciando all'operatore la decisione su quando e come aggiornarlo. La valutazione viene condotta su dataset di malware Android, con esperimenti su finestre temporali di sei mesi.

Punti di forza Transcend introduce un contributo metodologico importante: il rilevamento proattivo del drift prima che le prestazioni degradino, che consente di pianificare il retraining in anticipo invece di reagire a deterioramenti già avvenuti. Il framework è agnostico rispetto al classificatore sottostante e non richiede un accesso continuo a dati etichettati per funzionare. La formalizzazione tramite stimatori conformali offre garanzie probabilistiche sulla qualità della predizione.

Limitazioni rispetto al presente lavoro Il limite principale di Transcend è che *rileva* il drift ma non *adatta* il modello: la risposta rimane completamente manuale. In un SOC operativo, con un flusso continuo di migliaia di catene di allarmi al giorno, una risposta puramente reattiva è insufficiente. Inoltre, il lavoro è valutato esclusivamente su malware Android, un dominio in cui i campioni hanno feature relativamente stabili (manifest, API calls) rispetto ai pattern di traffico di rete (che cambiano su scale temporali molto brevi). Infine, non viene proposta né valutata alcuna strategia concreta di retraining o ponderazione temporale dei campioni.

3.2.3 INSOMNIA: robustezza al concept drift nel rilevamento delle intrusioni di rete

Approccio INSOMNIA affronta direttamente il problema del concept drift nei sistemi NIDS, che è esattamente il dominio applicativo della presente tesi. Il framework propone un rilevatore semi-supervisionato che si aggiorna continuamente man mano che il traffico di rete si evolve, combinando tre componenti:

1. **Active learning**: un modulo che seleziona attivamente i campioni più informativi da sottoporre all'etichettatura da parte degli analisti, riducendo il numero di label necessarie per mantenere il modello aggiornato;

2. **Label estimation**: per i campioni non selezionati per l'etichettatura manuale, il sistema stima automaticamente le etichette (*pseudo-labeling*), permettendo al modello di aggiornarsi anche su campioni non revisionati dall'analista;

3. **Explainable AI**: un modulo basato su tecniche di spiegabilità che aiuta gli analisti a interpretare le decisioni del sistema e a identificare feature che indicano un cambiamento nella distribuzione;

La valutazione è condotta su dataset pubblici di traffico di rete (CICIDS e simili), mostrando che INSOMNIA mantiene prestazioni più stabili nel tempo rispetto a un classificatore statico in presenza di drift.

Punti di forza INSOMNIA è il lavoro più vicino al dominio applicativo di questa tesi, affrontando esplicitamente il concept drift in un sistema NIDS. La combinazione di active learning e label estimation riduce il collo di bottiglia dell'etichettatura manuale, che è uno dei vincoli operativi più critici in un SOC reale. Il framework è progettato per l'aggiornamento continuo, non per il retraining periodico, avvicinandosi maggiormente alla realtà operativa di un SOC.

Limitazioni rispetto al presente lavoro La componente di active learning presuppone un'infrastruttura in cui il sistema può interrogare l'analista in modo interattivo su campioni specifici

(un modello di interazione che non rispecchia la realtà operativa di molti SOC, dove gli analisti revisionano gli alert in batch e con latenza variabile).

La pseudo-etichettatura automatica introduce un rischio di degrado a cascata che è opportuno motivare nel dettaglio. In presenza di concept drift, il modello produce già predizioni errate nella regione dello spazio delle feature colpita dalla deriva; le pseudo-etichette generate in quella regione saranno sistematicamente sbagliate. Addestrare il modello su etichette errate non solo non corregge il drift, ma rischia di rinforzare le decisioni errate (un effetto analogo al *confirmation bias* nell'auto-addestramento, in cui il modello converge verso le proprie predizioni invece che verso il ground truth [16]). Quanto più il drift è intenso, tanto più le pseudo-etichette sono rumorose, e tanto più il retraining su di esse peggiora il modello invece di migliorarlo.

La presente tesi evita interamente questo problema ancorando l'aggiornamento del modello esclusivamente alle etichette prodotte dagli analisti attraverso il sistema di ticketing del SOC: ogni sessione per cui un analista ha aperto un ticket (`has_ticket = 1`) costituisce una label positiva certificata dalla revisione umana, indipendentemente dalla confidenza del modello. Questo approccio rinuncia alla copertura totale dei campioni (solo le sessioni effettivamente revisionate contribuiscono al retraining) ma garantisce la qualità del segnale di supervisione, che è la precondizione necessaria affinché l'aggiornamento produca un effetto correttivo e non amplificante.

Infine, INSOMNIA è valutato su dataset pubblici di benchmark, la cui distribuzione è significativamente diversa da quella di un dataset operativo reale prodotto da un SOC su infrastruttura di clienti reali [3]. La complessità architetturale del sistema, con tre moduli separati che devono coordinarsi, rende più difficile il deployment in ambienti operativi e risorse limitate.

3.2.4 Fast & Furious: il rilevamento del malware come flusso evolutivo

Approccio Questo lavoro adotta una prospettiva radicalmente diversa rispetto ai precedenti: invece di trattare il dataset come statico e aggiungere meccanismi di adattamento al margine, propone di modellare fin dall'inizio il problema del rilevamento del malware come un *data stream* evolutivo, impiegando algoritmi di *stream learning* nativamente progettati per operare su flussi di dati non stazionari.

L'algoritmo principale valutato è *Adaptive Random Forest* (ARF), una variante del Random Forest progettata per i data stream che incorpora un meccanismo di drift detection (tipicamente AD-

WIN) per identificare quale albero della foresta ha degradato e sostituirlo con uno nuovo addestrato sui dati recenti. La valutazione confronta ARF con classificatori batch tradizionali (Random Forest, SVM) su un dataset di 415.000 applicazioni Android, organizzato come flusso temporale, mostrando che l'approccio stream learning produce AUC più stabili nel tempo rispetto ai classificatori batch.

Punti di forza Il punto di forza principale è concettuale: trattare il problema come un data stream fin dall'inizio porta a soluzioni più coerenti con la natura non stazionaria dei dati di sicurezza, invece di rattoppare a posteriori un classificatore batch con meccanismi di adattamento. Adaptive Random Forest offre un aggiornamento continuo, computazionalmente efficiente e automatico, senza richiedere una soglia esplicita di degrado che triggeri il retraining. La valutazione su 415.000 campioni su un arco temporale pluriennale fornisce evidenze empiriche robuste sull'efficacia dell'approccio.

Limitazioni rispetto al presente lavoro Il lavoro è focalizzato sulla classificazione di applicazioni Android (malware detection), un dominio con caratteristiche molto diverse dalla classificazione di catene di allarmi Suricata: le feature degli APK (manifest, API calls, permessi) hanno struttura e dimensionalità diverse rispetto alle rappresentazioni TF-IDF di sequenze di messaggi Suricata. Gli algoritmi di stream learning come ARF assumono un flusso continuo di campioni con disponibilità di label immediata, mentre in un SOC reale le label vengono prodotte dagli analisti con latenza variabile e copertura parziale. Infine, il lavoro non affronta esplicitamente la strategia di ponderazione temporale (*time decay*): l'adattamento avviene attraverso la sostituzione degli alberi degradati, non attraverso la modifica del peso dei campioni storici. Questo rende difficile confrontare direttamente i risultati con le strategie proposte nella presente tesi.

3.3 Posizionamento del presente lavoro rispetto alla letteratura

L'analisi dei lavori precedenti e correlati permette di delineare con precisione il posizionamento e il contributo originale di questa tesi.

Rispetto ai lavori preliminari su questo sistema I lavori preliminari hanno stabilito l'architettura del sistema ML-SOC su cui si basa questa tesi e hanno dimostrato la fattibilità dell'approccio su dati operativi reali. Il presente lavoro estende quella base lungo una dimensione finora inesplorata:

la *robustezza temporale*. Invece di valutare il sistema in un singolo punto nel tempo, si valuta come le prestazioni evolvano su una sequenza di finestre temporali successive, adottando la metrica AUT come indicatore sintetico della stabilità nel tempo.

Rispetto ai lavori ML-SOC della letteratura Sommer e Paxson [16] identificano la non-stazionarietà dei dati e la qualità delle etichette come i due ostacoli principali nell'adozione del ML in ambienti di sicurezza reali: questa tesi li affronta entrambi, il primo con le strategie di retraining continuo, il secondo ancorando le label al sistema di ticketing del SOC. AI^2 [17] condivide con questa tesi l'architettura di aggiornamento periodico basata sul feedback degli analisti, ma non quantifica il concept drift né adotta un protocollo di valutazione temporalmente corretto. TEQ [9] e Ghadermazi et al. [10] dimostrano che sistemi ML per il triage degli alert funzionano bene su dati operativi reali al momento del deployment, ma nessuno dei due affronta il problema dell'obsolescenza del modello nel tempo: entrambi valutano il sistema in configurazione statica sull'intera finestra di osservazione. La presente tesi parte esattamente da questa lacuna, quantificando la degradazione temporale e proponendo strategie per mitigarla.

Rispetto a Transcend Transcend [11] fornisce un meccanismo elegante per rilevare quando un modello sta invecchiando, ma non propone né valuta strategie concrete di adattamento. La presente tesi parte esattamente da quel punto di arrivo: dando per noto che il drift avviene (come dimostrato da Transcend e TESSERACT [14]), si concentra sulla valutazione comparativa di strategie di mitigazione (retraining con rolling window, time decay esponenziale, apprendimento incrementale) su un sistema NIDS reale.

Rispetto a INSOMNIA INSOMNIA [2] condivide con questa tesi il dominio applicativo (NIDS) e l'obiettivo (robustezza al concept drift), ma adotta un'architettura più complessa basata su active learning e pseudo-labeling che presuppone un modello di interazione analista-sistema difficilmente replicabile in produzione. La presente tesi propone strategie più semplici e operativamente sostenibili (rolling window e time decay), che richiedono solo un flusso di label prodotte dagli analisti nel normale workflow operativo del SOC, senza infrastrutture aggiuntive di interrogazione attiva.

Rispetto a Fast & Furious Ceschin et al. [6] dimostrano il vantaggio del paradigma data stream per il rilevamento del malware. La presente tesi non adotta algoritmi di stream learning nativi (che presuppongono label immediate), ma valuta l'alternativa operativamente più praticabile di un retraining batch periodico con ponderazione temporale dei campioni storici. Il confronto tra le due filosofie di adattamento, stream learning vs retraining periodico con time decay, costituisce uno dei contributi di analisi del presente lavoro.

Contributo originale Il valore aggiunto specifico di questa tesi rispetto all'insieme dei lavori analizzati può essere sintetizzato in quattro punti:

1. **Dataset operativo reale**: a differenza dei lavori su dataset pubblici di benchmark, la valutazione viene condotta su dati prodotti da un SOC reale basato su Suricata, con la distribuzione di alert, etichette e pattern temporali tipica di un ambiente di produzione;

2. **Rappresentazione domain-specific**: il sistema utilizza catene di allarmi Suricata vettorizzate con TF-IDF sui messaggi testuali delle regole, una rappresentazione progettata specificamente per il dominio NIDS, a differenza delle feature generiche usate nei sistemi di rilevamento malware;

3. **Valutazione metodologicamente corretta**: si adottano i protocolli di valutazione temporalmente corretti di TESSERACT [14] e la metrica AUT, che permette di confrontare equamente strategie diverse sulla loro stabilità nel tempo, non solo sulle prestazioni al momento dell'addestramento;

4. **Confronto sistematico di strategie**: vengono confrontate in modo sistematico, sullo stesso dataset e con la stessa metodologia, le strategie di baseline (modello statico), retraining con rolling window e ponderazione time decay esponenziale, fornendo indicazioni pratiche sulla scelta della strategia più adeguata in funzione del tipo di drift e delle risorse disponibili;

La Tabella 3.1 riassume il posizionamento del presente lavoro rispetto ai contributi analizzati lungo le dimensioni più rilevanti.

Tabella 3.1: Confronto tra il presente lavoro e i principali contributi correlati

✓ = presente; ∼ = parzialmente presente; × = assente

Caratteristica	***Lavoro di ricerca - 1***	***Lavoro di ricerca - 2***	***Sommer&Paxson***	***AI²***	***TEQ***	***Ghadermazi***	***Transcend***	***INSOMNIA***	***Fast&Furious***	***Questa tesi***
Dati operativi reali (SOC)	✓	✓	×	✓	✓	✓	×	×	×	✓
Contesto SOC (non solo NIDS)	✓	✓	✓	✓	✓	✓	×	×	×	✓
Dominio NIDS / Suricata	✓	✓	∼	∼	∼	×	×	∼	×	✓
Catene di allarmi	✓	✓	×	×	×	×	×	×	×	✓
Valutazione temporale (AUT)	×	×	×	×	×	×	∼	∼	∼	✓
Meccanismo di adattamento	×	×	×	✓	×	×	×	✓	✓	✓
Label certificate (no pseudo)	✓	✓	—	✓	✓	—	—	×	×	✓
Time decay / ponderazione	×	×	×	×	×	×	×	×	×	✓
Rolling window retraining	×	×	×	×	×	×	×	×	×	✓
Semplicità operativa	✓	✓	—	∼	✓	∼	✓	×	∼	✓

4. Metodologia: Analisi e Mitigazione del Concept Drift

Il presente capitolo descrive le scelte metodologiche adottate nella progettazione e nella valutazione del sistema ML-SOC oggetto di questa tesi. La struttura segue una progressione dell'analisi del contesto operativo alla formalizzazione delle soluzioni proposte.

Nella prima sezione si definisce il *threat model*: l'avversario modellato, le sue capacità e i vettori d'attacco rilevanti per il sistema, le assunzioni sull'infrastruttura difensiva e i limiti espliciti del modello. Nella seconda sezione si modella il SOC ad alto livello, stimando i volumi operativi tipici e descrivendo la pipeline operativa dal log grezzo alla decisione di triage. La terza sezione, che costituisce il contributo metodologico centrale della tesi, presenta l'architettura del sistema proposto e le tre strategie di ingegneria delle feature e di aggiornamento del modello che si intendono confrontare: la rappresentazione contestuale degli alert tramite *Alarm Chains*, la modellazione della memoria temporale tramite *Time Decay* esponenziale e la mitigazione del drift tramite *Incremental Learning* con *Rolling Window*.

4.1 Threat Model: scenario di attacco e difesa

La definizione esplicita di un threat model è un prerequisito metodologico fondamentale per qualsiasi sistema di rilevamento basato su ML: stabilisce quale avversario il sistema è progettato per contrastare, quali comportamenti ci si aspetta che il modello classifichi correttamente e quali scenari esulano dal perimetro di validità delle valutazioni sperimentali.

4.1.1 Definizione dell'avversario: capabilities, obiettivi e vettori d'attacco

L'avversario modellato in questa tesi è un attore esterno alla rete monitorata che opera attraverso canali di rete rilevabili da un Network Intrusion Detection System basato su regole. Rientrano in questa categoria sia attori opportunistici, che sfruttano vulnerabilità note con strumenti auto-

matizzati, sia attori avanzati (Advanced Persistent Threat) che conducono campagne mirate in più fasi.

Capacità Operative Si assume che l'avversario sia in grado di: condurre attività di ricognizione della rete (*network scanning*); sfruttare vulnerabilità note in servizi esposti su rete (*exploitation*); distribuire payload malevoli attraverso protocolli applicativi (HTTP, DNS, SMTP); stabilire canali di comando e controllo (Command and Control) verso infrastrutture esterne; eseguire movimenti laterali all'interno del segmento di rete monitorato; installare componenti di mining di criptovalute o ransomware su host compromessi. Tutte queste attività generano traffico di rete che attiva, in misura variabile, le regole del ruleset Suricata Emerging Threats Open installato sui sensori.

Obiettivi Gli obiettivi primari dell'avversario sono la compromissione della riservatezza (esfiltrazione di dati), dell'integrità (modifica di configurazioni o dati) e della disponibilità (interruzione di servizi) delle risorse del cliente monitorato. Ai fini di questa tesi, l'obiettivo rilevante è che le attività malevole producano un numero sufficiente di alert Suricata da rendere la sessione classificabile come malevola, cioè meritevole di apertura di un ticket da parte degli analisti.

Vettori d'attacco rilevanti Il sistema analizzato osserva esclusivamente il traffico di rete a livello di Network Intrusion Detection System: i vettori modellati sono pertanto quelli che lasciano traccia nel flusso di alert Suricata. Le categorie di alert più frequentemente associate ad attività malevole nel dataset operativo utilizzato sono: `web-application-attack`, `trojan-activity`, `attempted-admin`, `attempted-user`, `coin-mining` e `exploit-kit`.

4.1.2 Modello di difesa: assunzioni sull'infrastruttura monitorata

Il modello difensivo si basa sulle seguenti assunzioni, verificate nell'infrastruttura operativa del SOC da cui provengono i dati sperimentali.

Sensoristica Ogni cliente è dotato di almeno un sensore Suricata (Network Intrusion Detection System) in modalità *inline* o *passive tap*, configurato con il ruleset Emerging Threats Open e integrato con regole personalizzate per il singolo cliente. Il sensore emette eventi in formato EVE JSON, che vengono raccolti e trasmessi alla piattaforma centralizzata del SOC in tempo quasi reale.

Dati disponibili per la classificazione Per ogni evento Suricata, sono disponibili: il timestamp di rilevamento, la categoria della regola (`rule_category`), la severità assegnata dalla regola (scala 1-3), l'indirizzo IP sorgente e destinazione, le porte, il numero di byte trasferiti, la direzione del flusso (`to_server` / `to_client`) e il protocollo di rete e di trasporto. Il ground truth è fornito dal sistema di ticketing del SOC: ogni sessione che gli analisti classificano come incidente genera un ticket, che costituisce l'etichetta positiva (`has_ticket` = 1) del dataset.

Multitenancy Il SOC gestisce simultaneamente un insieme di clienti eterogenei per dimensione, settore e profilo di traffico. I dati sperimentali provengono da dieci clienti reali; ciascun cliente viene trattato come un dominio indipendente: i modelli vengono addestrati e valutati separatamente per cliente.

4.1.3 Limitazioni e confini del modello proposto

Il modello proposto presenta alcune limitazioni strutturali che è opportuno dichiarare esplicitamente. In primo luogo, il sistema osserva esclusivamente eventi Network Intrusion Detection System: attività malevole che non generano alert di rete (es. attacchi *insider*, compromissioni a livello di endpoint che non producono traffico anomalo di rete) non sono rilevabili. In secondo luogo, l'avversario modellato non è *adversarially aware*: non si assume che l'attaccante conosca il modello di ML e tenti di eluderlo mediante tecniche di evasione (adversarial examples, poisoning attacks). L'analisi della robustezza ai modelli avversariali adattivi esula dall'obiettivo di questa tesi. In terzo luogo, il ground truth è prodotto da analisti umani tramite il sistema di ticketing: è soggetto alle tipiche imperfezioni del processo di triage manuale (ritardi nell'apertura del ticket, possibili falsi negativi per sessioni borderline). La qualità dell'etichettatura costituisce pertanto un limite ereditato dal sistema operativo e non modificabile nel contesto di questa tesi.

4.2 Modellazione ad alto livello del SOC

Prima di descrivere il sistema proposto è utile collocare il problema in una rappresentazione quantitativa del SOC, in modo da motivare i requisiti di latenza e scalabilità del componente ML.

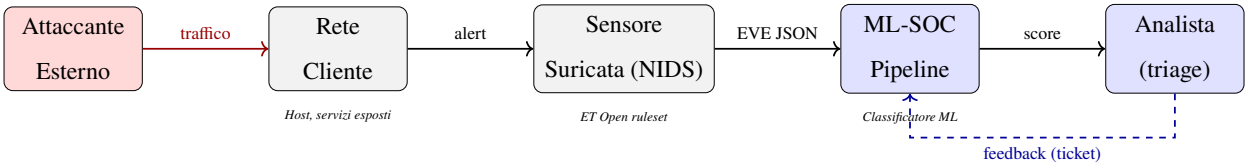

Figura 4.1: Schema del threat model: l'attaccante esterno genera traffico malevolo che il sensore Suricata converte in alert EVE JSON. La pipeline ML-SOC produce uno score di rischio che guida il triage dell'analista; la decisione dell'analista (apertura del ticket) costituisce il feedback per il retraining del modello.

4.2.1 Stima dei volumi operativi: clienti, alert giornalieri, analisti

Il Security Operations Center dell'MSSP coinvolto dispone di 20 analisti attivi che eseguono il triage manuale degli alert; la percentuale media di alert che sfocia in un ticket confermato è inferiore al 20% per tutti i clienti osservati, con casi estremi in cui scende sotto l'1%.

Il tasso di incidenza, ovvero la frazione di finestre che contengono almeno un ticket confermato dagli analisti, è sistematicamente inferiore al 20% per tutti i clienti osservati, con casi estremi che si avvicinano all'1%. Questo forte sbilanciamento di classe (*class imbalance*) è una caratteristica strutturale del dominio e deve essere trattato esplicitamente nella pipeline di addestramento; i dettagli quantitativi sul dataset sono riportati nel Capitolo 5.

4.2.2 Pipeline operativa: dal log grezzo alla decisione di triage

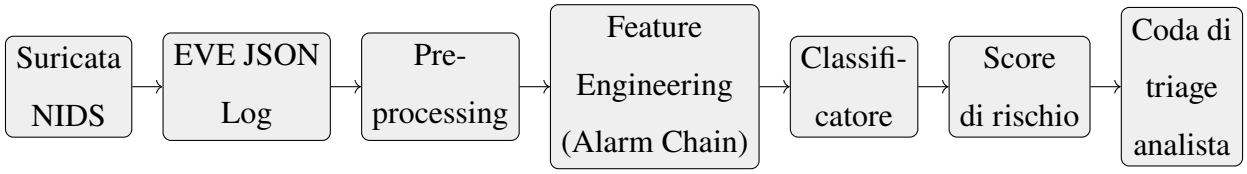

Figura 4.2: Pipeline operativa: dal log grezzo alla decisione di triage

La pipeline operativa del sistema ML-SOC si articola in cinque fasi sequenziali, illustrate nel seguito.

Raccolta e normalizzazione degli eventi I sensori Suricata di ciascun cliente trasmettono eventi in formato EVE JSON alla piattaforma del SOC. Ogni evento contiene, tra i campi rilevanti: il

timestamp di generazione, la categoria della regola (`rule_category`), la severità (1-3), gli indirizzi IP e le porte sorgente e destinazione, il protocollo di rete e di trasporto e il volume di byte trasferiti. In questa fase si eseguono la normalizzazione dei timestamp in UTC, il filtraggio degli eventi con campi obbligatori mancanti e la deduplicazione di eventi identici generati da sensori ridondanti.

Costruzione delle Alarm Chains Gli eventi normalizzati vengono aggregati per finestra temporale scorrevole al fine di costruire *Alarm Chains*, unità di analisi del classificatore. I parametri dell'aggregazione e le feature estratte sono descritti in dettaglio nella §4.3.2.

Vettorizzazione delle feature Ogni Alarm Chain viene trasformata in un vettore numerico combinando due rappresentazioni: una vettorizzazione TF-IDF della sequenza testuale delle categorie di alert e un vettore numerico delle feature aggregative (conteggio ponderato degli eventi, severità ponderata). Il vettore risultante è la concatenazione sparsa delle due rappresentazioni.

Classificazione ML Il vettore di feature viene sottoposto al classificatore ML correntemente in uso per il cliente, che produce una decisione binaria (benigno/malevolo) per la finestra corrente. I classificatori valutati sono descritti nel dettaglio nel Capitolo 5.

Triage e feedback Le finestre classificate come malevole vengono accodate per la revisione da parte degli analisti. La decisione dell'analista (apertura o chiusura del ticket) costituisce il feedback che alimenta il ciclo di retraining del modello.

4.2.3 Requisiti del sistema ML-SOC: latenza, accuratezza, aggiornabilità

I requisiti operativi che il sistema deve soddisfare per essere utilizzabile in produzione sono di tre ordini:

Latenza La finestra di aggregazione degli alert è di breve durata con passo di scorrimento ridotto (cfr. §4.3.2): il sistema produce una nuova osservazione a ogni passo. Il tempo di classificazione di una singola finestra deve essere sufficientemente contenuto da non creare un backlog di osservazioni non processate. I classificatori ensemble adottati hanno tempi di inferenza nell'ordine dei millisecondi su un singolo campione, un requisito di latenza facilmente soddisfatto.

Accuratezza In un contesto di triage il requisito primario è un *recall* elevato per la classe positiva (incidenti): un falso negativo (una sessione malevola non rilevata) ha un costo operativo ben maggiore di un falso positivo, che comporta al più l'analisi inutile di una sessione benigna. La metrica di valutazione principale è l'Area Under Time (definita nella §6.1.2), che cattura la degradazione del recall e dell'F1-score nel tempo a partire dal deployment del modello.

Aggiornabilità Poiché la distribuzione degli alert evolve nel tempo per effetto del concept drift (nuove campagne d'attacco, aggiornamenti del ruleset Suricata, cambiamenti nell'infrastruttura del cliente), il modello deve poter essere aggiornato periodicamente senza interruzione del servizio. Il ciclo di retraining è progettato per essere eseguito con granularità settimanale, ovvero alla chiusura di ogni settimana di osservazione viene valutata la performance corrente e si decide se (e come) aggiornare il modello.

4.3 Progettazione del ML-SOC e strategia di mitigazione

In questa sezione si descrivono le scelte progettuali del sistema proposto: l'architettura complessiva, la rappresentazione delle Alarm Chains con le relative feature, la modellazione della memoria temporale tramite Time Decay e le strategie di aggiornamento continuo del modello.

4.3.1 Architettura del sistema proposto

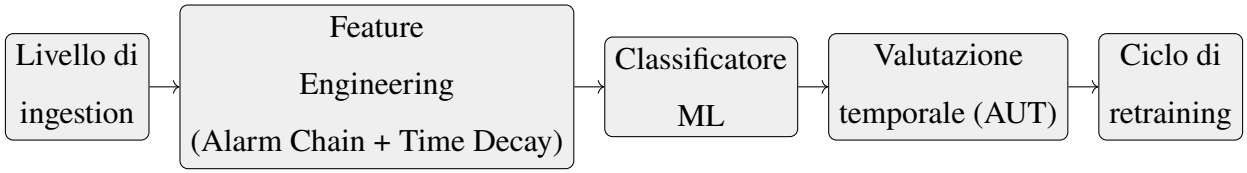

Figura 4.3: Architettura del sistema ML-SOC

Il sistema è organizzato in quattro livelli funzionali:

1. **Livello di ingestion**: raccoglie i dati grezzi di ciascun cliente (un file CSV per cliente, con un record per ogni evento Suricata normalizzato) e li prepara per l'elaborazione successiva;

2. **Feature Engineering**: implementa la costruzione delle Alarm Chains e il calcolo delle feature di Time Decay, producendo un dataset strutturato in cui ogni riga corrisponde a una finestra temporale scorrevole;

3. **Classificazione ML**: addestra e valuta i modelli ML. Per gestire lo sbilanciamento di classe, sistematicamente presente in tutti i clienti del dataset, si adotta una strategia di *undersampling* controllato della classe negativa: il rapporto negativo:positivo nel training set è fissato a un valore scelto sperimentalmente come compromesso tra fedeltà alla distribuzione reale e sufficiente presenza della classe positiva [14];

4. **Valutazione temporale e retraining**: implementa la valutazione temporalmente corretta e i tre meccanismi di aggiornamento del modello, descritti nella §4.3.4;

4.3.2 Rappresentazione contestuale degli alert: Alarm Chains

La rappresentazione degli alert come eventi isolati, ciascuno descritto da categoria e severità, è insufficiente a catturare la struttura contestuale di un attacco: le campagne malevole si manifestano tipicamente come sequenze di alert correlati nel tempo, mentre il traffico benigno genera alert sporadici e non strutturati.

Aggregazione temporale Per catturare questa struttura sequenziale, gli eventi vengono aggregati tramite una finestra temporale scorrevole di ampiezza fissa con passo di scorrimento ridotto: ogni finestra include tutti gli eventi il cui timestamp rientra nell'intervallo $[t, t + W)$, e la finestra successiva inizia a $t + \delta$. La scelta dei parametri W e δ riflette un compromesso tra sensibilità (finestre più strette rilevano pattern rapidi ma perdono campagne lente) e specificità (finestre più ampie catturano sequenze più lunghe ma accumulano rumore); i valori adottati nella valutazione sperimentale sono riportati nel Capitolo 5.

Feature estratte Per ogni finestra, le feature estratte sono le seguenti:

1. `alarm_chain`: la sequenza ordinata per timestamp delle categorie Suricata degli eventi nella finestra, rappresentata come stringa di testo con le categorie separate da spazio (es.

`"attempted-admin trojan-activity attempted-user"`). Questa stringa è successivamente vettorizzata tramite TF-IDF;

2. **`event_count`**: il numero totale di eventi nella finestra, misura del volume di attività nella sessione;

3. **`max_severity`**: la severità massima tra tutti gli eventi nella finestra scala (1-3), misura dell'intensità dell'attività più critica;

4. **`duration_sec`**: l'intervallo in secondi tra il primo e l'ultimo evento nella finestra, misura della distribuzione temporale degli alert all'interno della sessione;

5. **`has_ticket`**: variabile target binaria; vale 1 se almeno un evento nella finestra è associato a un ticket aperto dagli analisti del SOC, 0 altrimenti;

Motivazione della rappresentazione TF-IDF La vettorizzazione TF-IDF della stringa `alarm_chain` consente di pesare le categorie di alert in modo proporzionale alla loro specificità: categorie rare ma altamente indicative di attività malevola (es. `trojan-activity`) ricevono un peso maggiore rispetto a categorie ubique e rumorose (es. `attempted-user`).

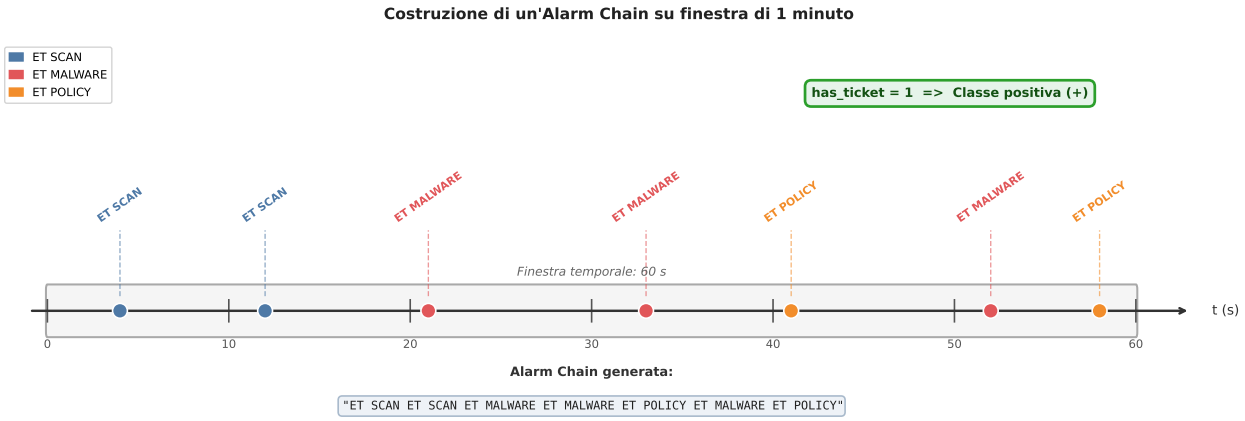

Figura 4.4: Esempio visivo di costruzione di un'Alarm Chain: timeline degli alert di un cliente in una finestra temporale scorrevole, con evidenza della sequenza di categorie e del ticket associato.

4.3.3 Modellazione della memoria temporale: Time Decay

Le feature di aggregazione standard (`event_count` e `max_severity`) trattano tutti gli eventi di una finestra con peso uniforme, indipendentemente dal momento in cui si sono verificati all'interno della finestra stessa. Questa scelta ignora un'informazione potenzialmente utile: in una sequenza di attacco, gli eventi più recenti (prossimi alla fine della finestra) sono tipicamente più significativi degli eventi iniziali, che possono corrispondere a fasi di ricognizione o rumore di fondo.

Formulazione matematica Per tenere conto della distanza temporale degli eventi dalla fine della finestra, si introduce un meccanismo di *Time Decay* esponenziale. Dato un evento e_i che si verifica al tempo t_i all'interno della finestra $[t_0, t_0 + W]$, il suo peso è:

$$w_i = e^{-\lambda \cdot (t_0 + W - t_i)} \tag{4.1}$$

dove $\Delta t_i = (t_0 + W - t_i) \in [0, W]$ è l'età dell'evento misurata a ritroso dalla fine della finestra, W è l'ampiezza della finestra temporale, e $\lambda > 0$ è il parametro di decadimento. Un valore λ elevato penalizza fortemente gli eventi vecchi; un valore ridotto li mantiene quasi equipesati agli eventi recenti.

Feature di Time Decay Le due feature numeriche standard vengono sostituite dalle loro versioni ponderate:

$$\texttt{decay\_event\_count} = \sum_{i=1}^{n} w_i \tag{4.2}$$

$$\texttt{decay\_max\_severity} = \sum_{i=1}^{n} w_i \cdot \text{severity}_i \tag{4.3}$$

dove la somma è estesa a tutti gli n eventi nella finestra. Il vettore di feature numerico del classificatore diventa quindi (`decay_event_count`, `decay_max_severity`), concatenato alla rappresentazione TF-IDF della `alarm_chain`

Selezione del parametro λ La scelta di λ determina l'effettiva *half-life* della memoria della finestra, definita come il ritardo $\tau_{1/2}$ per cui $w(\tau_{1/2}) = 0.5$:

$$\tau_{1/2} = \frac{\ln 2}{\lambda} \tag{4.4}$$

Nella valutazione sperimentale del Capitolo 6 vengono confrontati diversi valori di λ con decadimenti progressivamente più aggressivi, corrispondenti a half-life via via più ridotte rispetto all'ampiezza della finestra temporale. L'analisi di sensibilità a λ è presentata nella §6.4.2.

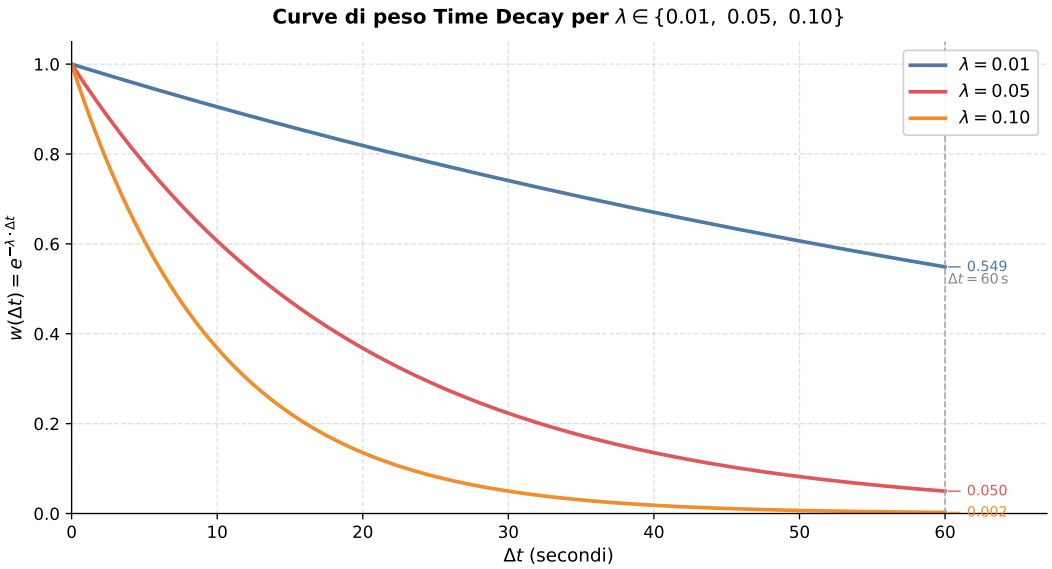

Figura 4.5: Curve di peso $w(\Delta t) = e^{-\lambda \cdot \Delta t}$ per i tre valori di λ su un asse temporale di 0–60 s. I valori annotati a destra indicano il peso residuo di un evento all'inizio della finestra ($\Delta t = 60$ s).

Motivazione operativa L'adozione del Time Decay risponde a una specifica osservazione empirica: le campagne di attacco moderne tendono a concentrare l'attività più aggressiva nella fase terminale di una sessione, dopo una fase di ricognizione iniziale. Attribuire peso maggiore agli eventi recenti consente al classificatore di reagire più rapidamente a questa intensificazione, potenzialmente migliorando il recall su attacchi con pattern temporali crescenti. L'ipotesi viene verificata sperimentalmente confrontando i modelli addestrati con feature standard e con feature di Time Decay (cfr. §6.4.2).

4.3.4 Strategia di mitigazione del drift: Incremental Learning con Rolling Window

Il terzo contributo metodologico riguarda il meccanismo di aggiornamento del modello nel tempo. Come argomentato nel Capitolo 3, un modello addestrato una volta su dati storici subisce una

degradazione progressiva delle prestazioni per effetto del concept drift. Per contrastare questa deriva, si propone e si confronta un insieme di strategie di retraining che operano su granularità settimanale.

Granularità temporale della valutazione L'unità temporale di valutazione è la **settimana ISO**: per ogni settimana dell'anno, si misura la performance del classificatore corrente sul sottoinsieme di finestre di quella settimana. Vengono considerate solo le settimane che contengono almeno cinque istanze positive (`has_ticket = 1`), per garantire la stabilità della stima dell'F1-score.

Il **training iniziale** viene effettuato sulle prime otto settimane di osservazione per ciascun cliente (pari a circa 2 mesi), dopodiché il modello viene valutato settimana per settimana sulle settimane successive. Questa scelta riflette un plausibile scenario operativo in cui il sistema viene messo in produzione dopo un periodo iniziale di osservazione e raccolta dati.

Modello statico Il modello statico viene addestrato una sola volta sulle prime otto settimane e non viene mai aggiornato. Costituisce la baseline rispetto alla quale si misura il beneficio delle strategie adattive. La degradazione delle sue prestazioni nel tempo quantifica l'impatto del concept drift non contrastato.

Modello Incrementale (Growing Window) Il modello incrementale viene riaddestrato ogni settimana aggiungendo i nuovi dati al training set accumulato. In ogni iterazione k, il training set include tutte le osservazioni dall'inizio fino alla settimana $k-1$:

$$\mathcal{D}_k^{\text{inc}} = \{(x_t, y_t) : t < k\} \tag{4.5}$$

Questa strategia massimizza la quantità di dati disponibili per il training, ma non risolve il problema della deriva: le osservazioni più vecchie, ormai obsolete rispetto alla distribuzione corrente, continuano a influenzare il modello.

Modello Rolling Window Il modello con rolling window viene riaddestrato ogni settimana su una **finestra scorrevole** di ampiezza fissa N settimane. In ogni iterazione k, il training set include solo le osservazioni delle ultime N settimane:

$$\mathcal{D}_k^{\text{roll}} = \{(x_t, y_t) : k - N \leq t < k\} \tag{4.6}$$

La finestra scorrevole garantisce che il modello sia sempre addestrato sulla distribuzione più recente, escludendo automaticamente i dati obsoleti. Il valore di N è scelto per offrire sufficiente copertura statistica delle classi positive mantenendo la reattività alla deriva; il valore adottato nella valutazione sperimentale è riportato nel Capitolo 5.

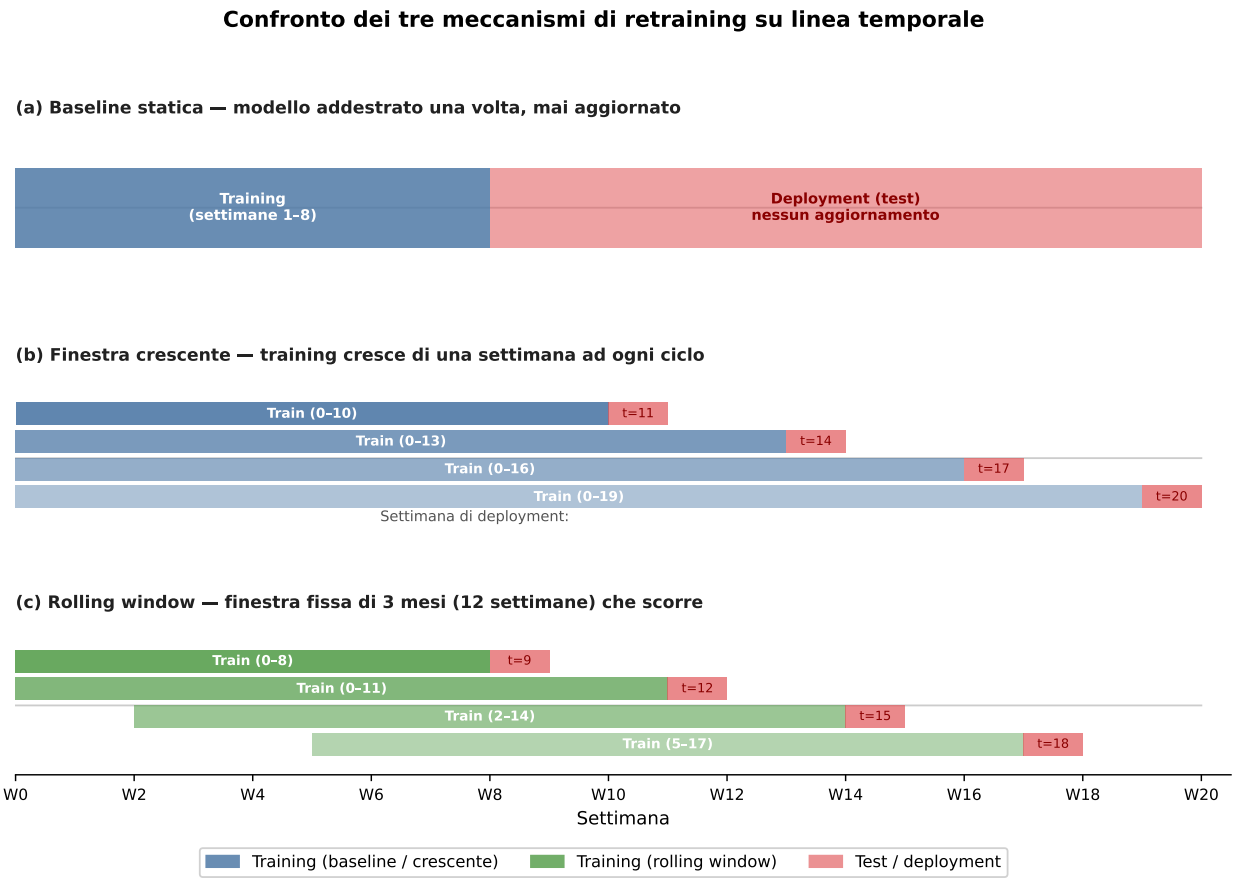

Figura 4.6: Schema dei tre meccanismi di retraining su linea temporale: baseline statica, finestra crescente (growing window) e rolling window. Ogni riga del pannello (b) e (c) rappresenta un ciclo di retraining settimanale; la barra rossa indica la settimana di test.

Confronto delle strategie Le tre strategie si collocano lungo uno spettro che bilancia due obiettivi in tensione: la *stabilità statistica* (più dati di training → stime più affidabili, specialmente per la classe positiva minoritaria) e la *reattività al drift* (meno dati storici → modello più aggiornato alla distribuzione corrente). Il modello statico si pone all'estremo della stabilità, il rolling window all'estremo della reattività, il modello incrementale in una posizione intermedia che tende pro-

gressivamente verso il modello statico man mano che la finestra crescente include sempre più dati obsoleti.

L'Area Under Time computata sulla sequenza settimanale di F1-score è la metrica sintetica che consente il confronto quantitativo tra le strategie sull'intero orizzonte temporale di valutazione. Valori di AUT più elevati indicano una performance media migliore nel tempo, tenendo conto sia del livello assoluto delle prestazioni sia della loro variabilità.

5. Esperimenti ed Implementazione

Il presente capitolo descrive in dettaglio l'implementazione del sistema ML-SOC proposto, documentando le scelte operative che concretizzano la metodologia presentata nel Capitolo 4. La struttura segue l'ordine logico del processo sperimentale: dalla definizione degli obiettivi, alla descrizione del dataset reale, al preprocessing, alla selezione e configurazione dei classificatori, fino all'implementazione dei due contributi sperimentali principali, ovvero il Time Decay e l'Incremental Learning con Rolling Window.

5.1 Ambiente di sviluppo

Tutti gli esperimenti descritti in questo capitolo sono stati eseguiti su una singola macchina locale. La Tabella 5.1 riassume la configurazione hardware e software adottata.

Tabella 5.1: Configurazione hardware e software dell'ambiente sperimentale.

Componente	**Specifica**
CPU	Intel Core i5-9600K @ 3.70GHz, 6 core
RAM	16 GB
Storage	1 TB
S.O.	Windows 11 Home
Python	3.11.8
scikit-learn	1.4.2
pandas	2.2.1
NumPy	1.26.4
SciPy	1.13.0
Matplotlib	3.8.4
JupyterLab	4.1.5

Il codice sperimentale è organizzato in notebook Jupyter (estensione `.ipynb`) con un notebook dedicato per ciascuna fase: preprocessing e feature engineering, ablation study sull'aggregazione,

selezione del modello e tuning degli iperparametri, valutazione temporale (statica, incrementale, rolling window) e calcolo dell'AUT. Non sono stati utilizzati framework di deep learning; tutti i modelli sono implementati tramite l'API di scikit-learn, che garantisce riproducibilità tramite il parametro `random_state`.

5.2 Obiettivi degli esperimenti

La campagna sperimentale persegue due macro-obiettivi distinti ma complementari.

5.2.1 Verifica empirica del Concept Drift nel dataset reale

Il primo obiettivo è verificare che il fenomeno del concept drift si manifesti con evidenza misurabile nel dataset operativo reale, prima ancora di tentare di mitigarlo. Un sistema ML-SOC addestrato su un periodo e poi messo in produzione subisce, nel corso del tempo, una degradazione delle prestazioni che non può essere attribuita all'overfitting né a errori di implementazione, ma è la conseguenza diretta dell'evoluzione della distribuzione dei dati. A tal fine si adotta un protocollo di valutazione *pre-post*: per ciascun cliente si addestra il modello sui dati dei primi tre mesi di osservazione (gennaio-marzo 2025) e lo si valuta, senza alcun aggiornamento, su ciascuno dei mesi successivi fino a ottobre. La metrica principale è l'F1-score sulla classe positiva (sessioni con ticket). Un calo marcato e sistematico dell'F1-score tra la fase di training e le valutazioni successive costituisce un'evidenza empirica del drift.

5.2.2 Valutazione comparativa delle strategie di mitigazione

Il secondo obiettivo è confrontare quantitativamente tre strategie di aggiornamento del modello su una valutazione temporalmente corretta con granularità settimanale:

1. **RF Statico**: modello Random Forest addestrato una volta sulle prime 8 settimane, mai aggiornato. Funge da baseline per misurare il beneficio delle strategie adattive;

2. **RF Incrementale**: modello Random Forest riaddestrato ogni settimana sull'intera finestra crescente di dati passati;

3. **MLP Incrementale**: rete neurale Multi-Layer Perceptron riaddestrata ogni settimana con la stessa politica incrementale;

La metrica di confronto primaria è l'Area Under Time, computata come integrale normalizzato della curva F1-score settimana per settimana. Valori di AUT più elevati indicano una capacità di rilevamento migliore e più stabile nell'arco dell'intero orizzonte temporale di valutazione.

5.3 Dataset

5.3.1 Infrastruttura monitorata e sorgenti dei dati (Suricata NIDS)

Il dataset utilizzato proviene dall'infrastruttura operativa di un Managed Security Service Provider reale e comprende i log di allerta prodotti dai sensori Suricata installati presso dieci clienti distinti. L'arco temporale coperto è di circa nove mesi, da settembre 2024 a maggio 2025. I clienti sono identificati con nomi anonimizzati: ClientAlfa, ClientBeta, ClientGamma, ClientDelta, ClientEpsilon, ClientZeta, ClientEta, ClientTheta, ClientIota, ClientKappa. Si tratta di organizzazioni eterogenee per dimensione, settore di appartenenza e volume di traffico di rete, il che rende il dataset particolarmente realistico e sfidante per la generalizzazione dei modelli.

I dati grezzi di ciascun cliente sono memorizzati in un file CSV con un record per ogni evento Suricata normalizzato, contenente i campi: `client`, `timestamp`, `rule_category`, `net_protocol`, `net_transport`, `src_ip`, `dst_ip`, `src_port`, `dst_port`, `src_bytes`, `dst_bytes`, `flow_direction`, `severity` e `ticket`. Il campo `client` è non nullo solo per gli eventi che gli analisti hanno collegato a un incidente, e costituisce la sorgente del ground truth.

5.3.2 Distribuzione degli alert e degli incidenti per cliente

La Tabella 5.2 riporta la distribuzione delle finestre temporali aggregate e degli incidenti (finestre con `has_ticket=1`) per ciascuno dei dieci clienti, dopo l'applicazione della pipeline di feature engineering descritta nella §5.4.

I dati rivelano un'eterogeneità marcata tra i clienti. Da un lato, clienti come ClientBeta (57.79%) e ClientAlfa (52.73%) presentano tassi di incidenza elevati che riflettono periodi di intensa attività malevola registrata negli archivi del SOC. Dall'altro, clienti come ClientTheta (1.78%),

Tabella 5.2: Distribuzione delle finestre di alert e degli incidenti per cliente dopo l'aggregazione con finestra di 1 minuto e passo di 10 secondi

Cliente	**Finestre tot.**	**Incidenti**	**Tasso inc.**
ClientAlfa	6 292	3 318	52.73%
ClientBeta	3 288	1 900	57.79%
ClientGamma	4 347	81	1.86%
ClientDelta	28 400	1 389	4.89%
ClientEpsilon	187	60	32.09%
ClientZeta	39 803	10 574	26.57%
ClientEta	9 104	304	3.34%
ClientTheta	18 620	332	1.78%
ClientIota	43 861	11 299	25.76%
ClientKappa	44 785	3 656	8.16%
Totale	**198 687**	**32 913**	**16.56%**

ClientGamma (1.86%) e ClientDelta (4.89%) mostrano tassi di incidenza molto bassi, che rendono il problema di classificazione particolarmente difficile per effetto dello sbilanciamento di classe. Quattro clienti (ClientIota, ClientDelta, ClientTheta e ClientEta) vengono selezionati come clienti *focus* per la valutazione temporale approfondita nel Capitolo 6, in quanto presentano una distribuzione temporale dei dati sufficiente a garantire osservazioni in un numero significativo di settimane distinte.

5.3.3 Il fenomeno della Target Sparsity e gestione del Ground Truth

Uno dei problemi strutturali del dataset operativo è la *target sparsity*: la distribuzione degli incidenti nel tempo non è uniforme, ma si concentra in picchi corrispondenti a campagne di attacco

specifiche, separati da periodi in cui il numero di ticket è minimo o nullo. Questo fenomeno ha conseguenze dirette sulla valutazione dei modelli: una settimana priva di ticket non può essere usata per calcolare l'F1-score sulla classe positiva (la divisione per zero renderebbe la metrica indefinita). Nel protocollo di valutazione adottato si escludono pertanto le settimane con meno di cinque istanze positive, preservando la significatività statistica delle stime. Il ground truth stesso è soggetto a latenza: l'apertura di un ticket da parte degli analisti può avvenire ore o giorni dopo gli eventi che ne hanno causato la generazione. Questo introduce un effetto di *label noise* temporale che, pur non essendo eliminabile nel contesto di questa ricerca, è mitigato dall'aggregazione in finestre di 1 minuto: eventi correlati che hanno generato il ticket vengono raggruppati nella stessa finestra.

5.4 Preprocessing

5.4.1 Pulizia e normalizzazione dei log grezzi

La pipeline di preprocessing si apre con una fase di pulizia dei log grezzi applicata per ciascun cliente indipendentemente. Le operazioni eseguite sono:

1. **Parsing dei timestamp**: i campi temporali vengono convertiti in oggetti `datetime` con formato ISO8601 e normalizzati in UTC, scartando i record con timestamp non validi;
2. **Deduplicazione**: record identici su tutti i campi obbligatori (`timestamp`, `rule_category`, `severity`, `src_ip`) generati da sensori ridondanti vengono eliminati;
3. **Filtraggio dei campi**: vengono scartati i record privi di `rule_category` o `severity`, indispensabili per la costruzione delle Alarm Chain;
4. **Ordinamento temporale**: i record vengono ordinati per timestamp crescente prima dell'aggregazione a finestra;

5.4.2 Costruzione delle Alarm Chains: ablation study sull'aggregazione

La scelta dei parametri di aggregazione (ampiezza della finestra, passo di scorrimento, criterio di raggruppamento) influenza direttamente la qualità delle feature estratte. Prima di adottare la confi-

gurazione finale, è stato condotto un ablation study che confronta quattro scenari di aggregazione su un sottoinsieme di clienti:

1. **Baseline (1 ora, per IP)**: finestre orarie raggruppate per indirizzo IP sorgente, con separatore `->` nella Alarm Chain e feature aggiuntive `duration_sec`, `events_per_sec` e `is_severity_uniform`;
2. **Ablation 30 minuti**: finestre di 30 minuti senza raggruppamento per IP;
3. **Ablation 2 ore**: finestre di 2 ore senza raggruppamento per IP;
4. **Ablation Hostname**: raggruppamento per hostname invece che per IP;

Tabella 5.3: Ablation study: impatto della strategia di aggregazione sulle prestazioni del modello. Il baseline originale utilizza finestre da 1 ora con raggruppamento per IP; la configurazione proposta adotta finestre scorrevoli da 1 minuto (passo 10 secondi) con rapporto di campionamento 5:1.

Scenario	**F1-Score**	**Δ vs baseline**
Baseline originale (1h, IP)	0.281	—
Finestra 30 minuti (IP)	0.437	+0.156
Finestra 2 ore (IP)	0.339	+0.058
Aggregazione per Hostname	0.495	+0.214
Proposta (1min/10sec, ratio 5:1)	**0.713**	**+0.432**

A seguito dell'ablation study è stata adottata la configurazione con **finestra di 1 minuto e passo di 10 secondi** senza raggruppamento per IP. Le motivazioni di questa scelta sono:

- Finestre brevi catturano meglio le sequenze di attacco concentrate (es. scan veloci, brute force) rispetto a finestre orarie che diluiscono eccessivamente gli eventi;
- L'assenza di raggruppamento per IP aumenta la completezza delle catene: più eventi simultanei provenienti da IP diversi (es. attacchi distribuiti, scan multi-sorgente) vengono aggregati nella stessa finestra;

- Il passo di 10 secondi garantisce che ogni evento venga incluso in multiple finestre sovrapposte, riducendo il rischio che un evento critico si collochi sull'edge di una finestra e venga perso;

Il Codice 5.1 mostra il nucleo della funzione di generazione delle feature per la finestra da 1 minuto con scorrimento di 10 secondi.

Codice 5.1: Generazione delle Alarm Chain con finestra scorrevole di 1 minuto

```
def generate_features(df, window_min=1, step_sec=10):
    df = df.sort_values('timestamp').set_index('timestamp')
    records = []
    current = df.index.min()
    end = df.index.max()

    while current <= end - pd.Timedelta(minutes=window_min):
        window = df.loc[current : current + pd.Timedelta(minutes=window_min)]
        if not window.empty:
            records.append({
                'window_start': current,
                'has_ticket'  : 1 if window['ticket'].notnull().any() else 0,
                'event_count' : len(window),
                'max_severity': window['severity'].max(),
                'duration_sec': (window.index.max() - window.index.min())
                                .total_seconds(),
                'alarm_chain' : ' '.join(window['rule_category'].astype(str))
            })
        current += pd.Timedelta(seconds=step_sec)

    return pd.DataFrame(records)
```

5.4.3 Feature engineering: TF-IDF e feature numeriche strutturate

Per ogni finestra temporale, il vettore di feature è la concatenazione sparsa di due rappresentazioni.

Rappresentazione testuale (TF-IDF) La stringa `alarm_chain` viene vettorizzata tramite `TfidfVectorizer` di scikit-learn, secondo lo schema di pesatura TF-IDF standard nell'information retrieval [15], con

vocabolario limitato a **50 feature**. Il limite di 50 è stato scelto come compromesso tra capacità espressiva e complessità computazionale: nei dataset dei clienti osservati, il vocabolario effettivo delle categorie Suricata non supera alcune decine di token distinti, per cui un limite di 50 cattura la quasi totalità del vocabolario rilevante.

Feature numeriche Le feature numeriche variano a seconda della configurazione sperimentale:

- **Configurazione standard**: `event_count`, `max_severity` e `duration_sec`;
- **Configurazione Time Decay**: `decay_event_count`, `decay_max_severity` (in sostituzione delle prime due; `duration_sec` viene omessa perché la ponderazione esponenziale incorpora già l'informazione temporale);

Il vettore finale è ottenuto impilando orizzontalmente le matrici sparse:

```
X = sp.sparse.hstack([sp.sparse.csr_matrix(X_num), X_tfidf])
```

La dimensione finale per campione è 2 + 50 = 52 (configurazione Decay) oppure 3 + 50 = 53 (configurazione standard).

5.4.4 Split temporale e gestione dello sbilanciamento delle classi

Split temporale Tutte le suddivisioni train/test adottate in questa tesi rispettano il principio di *no data leakage* temporale: i dati di test sono sempre cronologicamente successivi ai dati di training. Non viene mai utilizzato uno split casuale (*random split*), che introdurrebbe il bias sperimentale documentato da Pendlebury et al. [14]. Vengono utilizzati due protocolli di split:

- **Split 8-1-3**: training sulle prime 8 settimane, validazione sulla settimana 9, test sulle settimane 10-12. Viene utilizzato per la valutazione iniziale statica (Single/Mixed experiments);
- **Split settimanale incrementale**: training iniziale sulle prime 8 settimane ISO; valutazione settimana per settimana sulle settimane successive, con aggiornamento del training set secondo la strategia scelta (statica, incrementale o rolling window). Viene utilizzato per la valutazione temporale principale;

Gestione dello sbilanciamento Il tasso di incidenza varia dall'1.78% al 57.79% tra i clienti, con la maggior parte dei clienti focus (ClientDelta 4.89%, ClientEta 3.34% e ClientTheta 1.78%) fortemente sbilanciati verso la classe negativa. Per gestire questo sbilanciamento si adotta una strategia di *undersampling controllata* della classe negativa a ogni ciclo di training, con rapporto negativo:positivo fissato a $r = 5{:}1$ (cioè al più di 5 istanze negative per ogni istanza positiva nel training set). Se le negative disponibili sono meno di $5 \cdot |\text{pos}|$, si usano tutte; se le positive sono troppe rispetto alle negative disponibili, si riduce anche il numero di positive:

Codice 5.2: Funzione di undersampling controllato con rapporto fisso

```
def flexible_fix_ratio(df, ratio=5):
    df_1 = df[df['has_ticket'] == 1]
    df_0 = df[df['has_ticket'] == 0]
    if df_1.empty or df_0.empty:
        return pd.DataFrame()
    n_0_target = len(df_1) * ratio
    if len(df_0) >= n_0_target:
        df_0 = df_0.sample(n=n_0_target, random_state=42)
    else:
        n_1_allowed = len(df_0) // ratio
        if n_1_allowed < 1:
            return pd.DataFrame()
        df_1 = df_1.sample(n=n_1_allowed, random_state=42)
    return pd.concat([df_1, df_0]).sample(frac=1, random_state=42)
```

La scelta di $r = 5$ è motivata dall'analisi di sensitività presentata nella §5.5.3: rapporti inferiori (es. $r = 3$) favoriscono il recall ma riducono la precisione; rapporti superiori (es. $r = 10$) stabilizzano la precisione a scapito di un recall molto basso sui clienti con tasso di incidenza ridotto.

5.5 Classificatori

5.5.1 Random Forest e HistGradientBoosting

Tutti i classificatori sono implementati tramite la libreria scikit-learn [13]. I due classificatori basati su ensemble di alberi decisionali costituiscono il fulcro della valutazione comparativa.

Random Forest (RF) Il Random Forest [5] costruisce un ensemble di alberi decisionali indipendenti, ciascuno addestrato su un sottocampione casuale delle istanze e delle feature. La predizione finale è ottenuta per voto di maggioranza. La configurazione adottata è `n_estimators=100` con `max_depth=20`: 100 alberi garantiscono stabilità statistica della stima, mentre la profondità massima di 20 limita la complessità di ogni singolo albero riducendo il rischio di overfitting sui clienti con dataset ridotti.

HistGradientBoosting (HGB) L'HGB è un'implementazione histogram-based del Gradient Boosting [7], analoga all'algoritmo LightGBM. Costruisce gli alberi in modo sequenziale, con ogni albero che corregge gli errori residui del precedente. La configurazione ottimale identificata dal grid search è `learning_rate=0.1` e `max_depth=5`. Una profondità massima di 5 garantisce alberi semplici e generalizzabili, mentre il learning rate di 0.1 bilancia la velocità di convergenza con la regolarizzazione implicita.

I risultati comparativi di RF e HGB sulla configurazione con finestra 1 minuto e rapporto 5:1 per i dieci clienti sono riportati nella Tabella 6.1 del Capitolo 6.

5.5.2 Multi-Layer Perceptron: architettura e ottimizzatori

Il terzo classificatore valutato è Multi-Layer Perceptron implementato tramite `MLPClassifier` di scikit-learn. L'MLP è addestrato con algoritmo Adam (`solver='adam'`), `max_iter=500` e funzione di attivazione ReLU sugli strati nascosti. Nella fase di analisi di sensitività (§5.5.3) vengono confrontate tre architetture:

- **(50, 25)**: due strati nascosti con 50 e 25 neuroni;
- **(100, 50)**: due strati nascosti con 100 e 50 neuroni;
- **(100, 50, 25)**: tre strati nascosti con 100, 50 e 25 neuroni;

I risultati dell'analisi di sensitività (Tabella 6.6 nel Capitolo 6) mostrano che le differenze di AUT_{F1} tra le tre architetture sono marginali su tutti i clienti analizzati. L'architettura (50, 25) viene quindi adottata come configurazione di default per la valutazione principale, in quanto offre la complessità computazionale inferiore a parità di performance.

5.5.3 Tuning degli iperparametri con validazione temporale

Iperparametri di RF e HGB Per la selezione degli iperparametri di RF e HGB è stata eseguito un *grid search* con cross-validation temporale (non casuale) sul training set, valutando le combinazioni:

- RF: `n_estimators` $\in \{50, 100\}$, `max_depth` $\in \{10, 20, \text{None}\}$;
- HGB: `learning_rate` $\in \{0.05, 0.1\}$, `max_depth` $\in \{3, 5\}$;

Le configurazioni ottimali emerse (RF: `n_estimators=100`, `max_depth=20`; HGB: `learning_rate=0.1`, `max_depth=5`) sono state adottate per tutti i clienti.

Sensitività al rapporto di undersampling L'analisi di sensitività al rapporto $r \in \{3, 5, 10\}$ è riportata nella Tabella 6.7 del Capitolo 6. Il rapporto $r = 5$ viene mantenuto come valore di default per la valutazione principale come compromesso tra recall e precisione.

5.6 Implementazione dei contributi sperimentali

5.6.1 Time Decay: implementazione tramite tuning di λ

Il Time Decay è implementato come fase di preprocessing che trasforma le feature numeriche standard in feature ponderate esponenzialmente. Per ogni finestra $[t_0,\ t_0 + W]$ con $W = 60$s, il peso di ogni evento e_i avvenuto al tempo t_i è:

$$w_i = e^{-\lambda\,(t_0+W-t_i)}$$

Con $\lambda = 0.05\ \text{s}^{-1}$ (valore derivato dall'analisi empirica dei dati ClientIota) un evento all'inizio della finestra riceve peso $e^{-0.05\times60} \approx 0.05$, mentre un evento alla fine riceve peso unitario.

Il Codice 5.3 mostra il calcolo delle feature di Time Decay per una singola finestra.

Codice 5.3: Calcolo delle feature di Time Decay per una finestra temporale.

```
1 def compute_decay_features(window_df, t0, W=60.0, lam=0.05):
2     """
3     window_df : DataFrame con eventi nella finestra [t0, t0+W]
4     t0        : timestamp di inizio finestra (datetime)
```

```
    W         : ampiezza finestra in secondi (default 60)
    lam       : parametro di decadimento lambda (default 0.05 s^-1)
    """
    t_end = t0 + pd.Timedelta(seconds=W)
    # eta di ogni evento rispetto alla fine della finestra
    ages = (t_end - window_df.index).total_seconds().clip(lower=0)
    weights = np.exp(-lam * ages)

    decay_event_count   = weights.sum()
    decay_max_severity  = (weights * window_df['severity']).sum()
    return decay_event_count, decay_max_severity
```

Confronto standard vs Time Decay Il confronto tra feature standard e feature di Time Decay è riportato nella Tabella 6.5 del Capitolo 6 (§6.4.2).

5.6.2 Incremental Learning: rolling window e passo di aggiornamento

L'implementazione del meccanismo di aggiornamento incrementale segue lo schema descritto nella §4.3.4 della metodologia. Il Codice 5.4 mostra il loop di valutazione settimanale per la strategia incrementale (growing window), che costituisce il nucleo comune a tutte e tre le strategie valutate.

Codice 5.4: Loop di valutazione settimanale per la strategia RF Incrementale.

```
def run_incremental(df, train_weeks, test_weeks, ratio=5):
    current_train = df[df['week'].isin(train_weeks)]
    results = []

    for w in test_weeks:
        test_df = df[df['week'] == w]
        if test_df['has_ticket'].sum() == 0:
            # Nessun ticket: aggiungi al training, salta la valutazione
            current_train = pd.concat([current_train, test_df])
            continue

        # Undersampling e feature extraction
        train_bal = flexible_fix_ratio(current_train, ratio)
```

```
        if train_bal.empty:
            continue
        tfidf = TfidfVectorizer(max_features=50)
        X_tr = sp.sparse.hstack([
            sp.sparse.csr_matrix(
                train_bal[['decay_event_count','decay_max_severity']].values),
            tfidf.fit_transform(train_bal['alarm_chain'].fillna('none'))
        ])
        X_ts = sp.sparse.hstack([
            sp.sparse.csr_matrix(
                test_df[['decay_event_count','decay_max_severity']].values),
            tfidf.transform(test_df['alarm_chain'].fillna('none'))
        ])
        model = RandomForestClassifier(n_estimators=100, random_state=42)
        model.fit(X_tr, train_bal['has_ticket'])
        preds = model.predict(X_ts)

        results.append({
            'Settimana': w,
            'F1'       : f1_score(test_df['has_ticket'], preds, zero_division
               =0),
            'Recall'   : recall_score(test_df['has_ticket'], preds,
               zero_division=0),
            'Precision': precision_score(test_df['has_ticket'], preds,
               zero_division=0)
        })
        # Aggiorna il training set per la settimana successiva
        current_train = pd.concat([current_train, test_df])

    return pd.DataFrame(results)
```

Le differenze tra le tre strategie risiedono esclusivamente nella costruzione di `current_train` prima del ciclo: per il modello *statico*, `current_train` rimane immutato per tutta la durata del ciclo; per la *rolling window* (3 mesi), a ogni iterazione si selezionano solo i dati delle ultime 12 settimane (circa 3 mesi) invece dell'intera storia accumulata.

Calcolo dell'AUT Al termine del ciclo settimanale, l'Area Under Time viene calcolata come integrale della curva F1-score normalizzato per la durata in settimane:

Codice 5.5: Calcolo dell'AUT normalizzato dalla sequenza settimanale di F1.

```
from scipy.integrate import trapezoid

def compute_aut(weekly_results_df):
    df = weekly_results_df.sort_values('Settimana').dropna(subset=['F1'])
    if len(df) < 2:
        return 0.0
    duration = df['Settimana'].max() - df['Settimana'].min()
    if duration == 0:
        return 0.0
    return trapezoid(df['F1'], df['Settimana']) / duration
```

5.6.3 Split temporale per la simulazione del deployment e analisi multi-client

Simulazione del deployment reale Il protocollo di valutazione settimanale è progettato per simulare le condizioni operative di un modello ML-SOC in produzione. Il training iniziale sulle prime 8 settimane corrisponde a un periodo di *warm-up* durante il quale il sistema osserva il traffico reale del cliente senza classificare automaticamente. A partire dalla settimana 9, il modello entra in produzione e viene valutato settimana per settimana senza mai accedere a dati futuri (*no look-ahead*). Questo protocollo è analogo a quanto raccomandato dal framework TESSERACT di Pendlebury et al. [14] per eliminare il *temporal bias* nella valutazione di classificatori ML per la sicurezza.

Esperimenti single-client vs mixed-client È stato condotto anche un esperimento di confronto tra la strategia *single-client* (modello addestrato esclusivamente sui dati del cliente in valutazione) e la strategia *mixed-client* (modello addestrato su un mix di dati provenienti da più clienti contemporaneamente). I risultati (Tabella 6.4 nel Capitolo 6, §6.4.3) mostrano che il training misto non porta a miglioramenti sistematici rispetto al single-client, confermando la natura client-specifica della distribuzione degli alert. La strategia single-client viene pertanto adottata per tutta la valutazione principale.

6. Risultati Sperimentali

Il presente capitolo riporta e interpreta i risultati della valutazione sperimentale del sistema ML-SOC descritto nei capitoli precedenti. La struttura segue una progressione logica: si definiscono dapprima le metriche adottate, si caratterizzano gli scenari di valutazione e si presentano i risultati numerici per concludere con un'analisi critica che collega i dati quantitativi alle ipotesi formulate nella metodologia.

6.1 Metriche di valutazione

6.1.1 Precision, Recall e F1-Score nel contesto del rilevamento delle intrusioni

Le metriche di classificazione binaria adottate sono le standard del dominio:

- **Precision**: $P = \mathrm{TP}/(\mathrm{TP} + \mathrm{FP})$, la frazione di sessioni classificate come malevole che sono effettivamente malevole. Misura la qualità delle allerte generate: una precision bassa implica un elevato numero di falsi allarmi che oberano gli analisti;

- **Recall**: $R = \mathrm{TP}/(\mathrm{TP} + \mathrm{FN})$, la frazione di sessioni malevole effettivamente rilevate. Nel contesto della sicurezza operativa il recall è la metrica più critica: un falso negativo, cioè un incidente non rilevato, ha conseguenze potenzialmente gravi, mentre un falso positivo comporta solo analisi manuale non necessaria;

- **F1-Score**: $F_1 = 2 \cdot (P \cdot R)/(P + R)$, la media armonica di precision e recall. Costituisce una misura sintetica che penalizza gli sbilanciamenti estremi tra le due componenti ed è la metrica principale adottata per il confronto tra configurazioni;

Tutte le metriche sono calcolate sulla classe positiva (`has_ticket = 1`), cioè sulle sessioni che gli analisti hanno classificato come incidenti.

6.1.2 AUT (Area Under Time): definizione e interpretazione come indice di resilienza

La valutazione statica su un singolo split temporale non è sufficiente a caratterizzare il comportamento di un classificatore in produzione, poiché non cattura l'evoluzione delle prestazioni nel tempo. A tale scopo si adotta la metrica Area Under Time, derivata dall'integrazione della curva delle prestazioni lungo l'asse temporale [14].

Dato un insieme di T punti di valutazione settimanali $\{(w_1, s_1), \ldots, (w_T, s_T)\}$ dove s_t è l'F1-score alla settimana w_t, l'AUT è definita come:

$$\text{AUT} = \frac{1}{w_T - w_1} \int_{w_1}^{w_T} F_1(w)\, dw \approx \frac{1}{w_T - w_1} \sum_{t=1}^{T-1} \frac{s_t + s_{t+1}}{2} (w_{t+1} - w_t) \tag{6.1}$$

L'approssimazione discreta è calcolata tramite la regola dei trapezi. Il valore di AUT è normalizzato nell'intervallo $[0, 1]$: un sistema che mantiene F1= 1 per tutta la durata della valutazione ottiene AUT= 1; un sistema che degrada istantaneamente a F1= 0 ottiene AUT= 0.

L'AUT è particolarmente adatto al contesto di questa tesi perché: penalizza un calo precoce delle prestazioni più di un calo tardivo, è insensibile alla scelta arbitraria di un singolo punto di valutazione e consente il confronto quantitativo tra strategie di retraining su orizzonti temporali diversi.

6.2 Scenari di valutazione

La valutazione è strutturata attorno a tre scenari distinti che rispecchiano l'evoluzione temporale di un sistema ML-SOC dalla fase di deployment iniziale alla produzione continuativa.

6.2.1 Pre-deployment: valutazione del modello statico come baseline

Il primo scenario valuta le prestazioni di un classificatore addestrato su dati storici e testato immediatamente su un periodo successivo, senza alcun gap temporale rilevante tra training e testing. Questo scenario corrisponde al *rilascio* del modello in produzione e fornisce una stima delle prestazioni attese in assenza di drift.

Il protocollo adottato è uno split temporale 80/20: i primi quattro quinti dei dati del cliente vengono usati per il training, l'ultimo per il test. La metrica riportata è l'F1-score sul test-set, unitamente a precision e recall. Il classificatore usato in questo scenario è il Random Forest con la configurazione ottimale identificata nell'ablation (`n_estimators=100`, `max_depth=20`).

6.2.2 Post-deployment: simulazione del degrado nel tempo senza mitigazione

Il secondo scenario simula il comportamento di un modello *statico*, cioè mai aggiornato dopo il deployment, valutato a cadenza mensile su un orizzonte di diversi mesi. Il modello viene addestrato sui dati dei primi tre mesi (gennaio-marzo 2025) e valutato, senza alcun aggiornamento, su ciascuno dei mesi successivi (aprile, maggio, settembre, ottobre), simulando esattamente le condizioni operative di un sistema legacy.

Questo scenario fornisce la prova empirica quantitativa del concept drift nel dataset reale: la differenza tra le prestazioni nel periodo di training e quelle nei mesi successivi misura direttamente l'entità della deriva subita dal modello.

6.2.3 Post-deployment con mitigazione: obiettivo e configurazione del sistema incrementale

Il terzo scenario introduce le strategie di mitigazione del drift descritte nella §4.3.4: il modello incrementale (growing window) e il modello con rolling window vengono confrontati con la baseline statica sul medesimo orizzonte temporale. La metrica di confronto è l'AUT sulla sequenza settimanale di F1-score (settimane 11–44 per i clienti focus). Questo scenario costituisce la valutazione principale della tesi: un guadagno di AUT statisticamente rilevante rispetto alla baseline statica costituisce la conferma sperimentale dell'efficacia del meccanismo di Incremental Learning proposto.

6.3 Risultati per scenario

6.3.1 Pre-deployment: baseline di performance per cliente

La Tabella 6.1 riassume i risultati di RF, HGB e MLP sulla configurazione con finestra 1 minuto e rapporto di undersampling negativo:positivo 5:1 (cfr. Capitolo 5) per i dieci clienti (split 80%/20% temporale, configurazione standard senza Time Decay). Il classificatore MLP utilizza architettura $(50, 25)$ con early stopping; le feature numeriche sono normalizzate con StandardScaler prima dell'addestramento.

Tabella 6.1: Performance di RF, HGB e MLP con finestra 1 minuto, passo 10 secondi, rapporto di undersampling negativo:positivo 5:1 (split temporale 80/20). F1, Precision e Recall sulla classe positiva nel test set. Il valore 0.000 indica assenza di predizioni positive nel test set

Cliente	**HGB**			**RF**			**MLP**	
	F1	Prec.	Rec.	F1	Prec.	Rec.	F1	Rec.
ClientAlfa	0.855	0.898	0.815	0.956	1.000	0.916	0.664	0.672
ClientBeta	0.930	0.898	0.964	0.930	0.898	0.964	0.939	0.982
ClientGamma	0.788	0.765	0.813	0.839	0.867	0.813	0.560	0.438
ClientDelta	0.679	0.800	0.590	0.708	0.820	0.622	0.669	0.576
ClientEpsilon	0.571	1.000	0.400	0.571	1.000	0.400	0.000	0.000
ClientZeta	0.094	0.556	0.051	0.570	0.889	0.419	0.000	0.000
ClientEta	0.626	0.816	0.508	0.855	0.800	0.918	0.000	0.000
ClientTheta	0.696	0.848	0.591	0.724	0.840	0.636	0.630	0.515
ClientIota	0.784	0.729	0.848	0.788	0.731	0.854	0.781	0.840
ClientKappa	0.008	0.600	0.004	0.187	0.687	0.108	0.000	0.000

I risultati (riassunti visivamente nella Figura 6.1) mostrano una forte eterogeneità tra i clienti. ClientBeta, ClientAlfa e ClientEta mostrano performance elevate (F1 $>$ 0.85 per RF), mentre ClientDelta, ClientTheta e ClientIota si collocano in una fascia intermedia (F1 $\approx$ 0.71–0.79 per RF). ClientZeta e ClientKappa mostrano performance molto basse con HGB (F1 $<$ 0.1), fenomeno che si ricollega alla target sparsity e alla distribuzione temporale irregolare degli incidenti descritta nella §5.3.3. La differenza HGB-RF è contenuta per la maggior parte dei clienti, con RF che tende a offrire recall leggermente superiore.

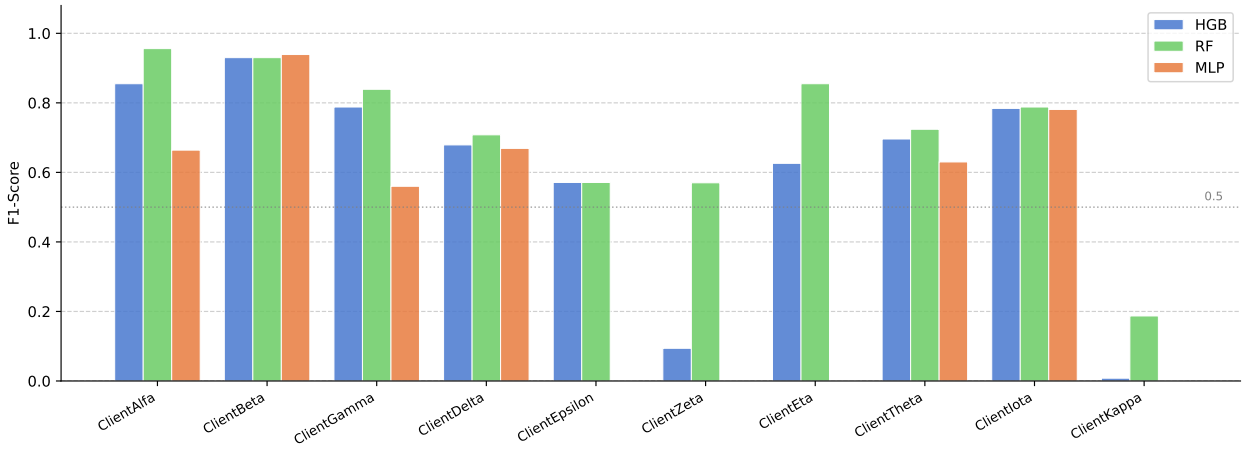

Figura 6.1: F1-Score pre-deployment (split 80/20) per RF, HGB e MLP sui dieci clienti del dataset. La linea tratteggiata indica la soglia F1 = 0.5. ClientZeta, ClientKappa, ClientEpsilon, ClientEta producono F1 nullo per MLP a causa della target sparsity; RF mantiene prestazioni superiori su questi clienti.

Per quanto riguarda MLP, i risultati sono polarizzati: ClientBeta e ClientIota mostrano prestazioni comparabili a RF e HGB (F1 rispettivamente 0.939 e 0.781), mentre ClientEpsilon, ClientZeta, ClientEta e ClientKappa producono F1 nullo, a causa di dataset troppo sbilanciati o con un numero di istanze positive insufficiente per la convergenza della rete neurale nell'approccio statico. L'MLP in configurazione statica non è quindi competitivo con RF su clienti a bassa densità di incidenti; il suo vantaggio emerge invece nella valutazione incrementale (§6.3.4), dove l'aggiornamento continuo compensa la maggiore sensibilità ai dati.

Takeaway. I classificatori considerati esibiscono quindi ottime performance in fase di pre-deployment, in linea con i risultati riportati dai lavori precedenti su task di triage assistito da ML in ambito SOC.

6.3.2 Post-deployment senza mitigazione: evidenza empirica del Concept Drift e quantificazione del degrado

La Tabella 6.2 mostra le prestazioni del modello Random Forest (RF) statico, addestrato sui dati gennaio-marzo 2025 (periodo di *warm-up*) e valutato, senza aggiornamenti, sui mesi successivi fino a ottobre. I valori evidenziano una degradazione sistematica e, nella maggior parte dei clienti,

catastrofica delle prestazioni.

Tabella 6.2: Evidenza del Concept Drift: F1-score del modello Random Forest (RF) statico (addestrato su Gen-Mar 2025) valutato mese per mese senza aggiornamenti. Il colore evidenzia la caduta delle prestazioni rispetto al periodo di training. N.D. = mese senza ticket sufficienti

Cliente	**Gen-Mar '25 (train)**	**Apr**	**Mag**	**Set**	**Ott**	**Degrado (%)**
ClientDelta	0.863	0.067	0.000	N.D.	0.012	−98.6%
ClientZeta	0.399	0.000	0.000	N.D.	0.000	−100%
ClientTheta	1.000	0.000	0.148	N.D.	0.007	−99.3%
ClientIota	0.973	0.747	0.725	0.000	0.497	−48.9%
ClientEta	1.000	0.000	0.000	N.D.	0.012	−98.8%
ClientGamma	1.000	0.000	N.D.	N.D.	0.000	−100%
ClientBeta	1.000	0.982	0.000	N.D.	0.345	−65.5%
ClientEpsilon	1.000	0.000	N.D.	N.D.	0.000	−100%

I dati documentano un fenomeno robusto e generalizzato. Sei degli otto clienti osservabili perdono oltre il 98% dell'F1-score in produzione rispetto al periodo di training: il modello RF addestrato su gennaio-marzo 2025 è sostanzialmente inutilizzabile ad ottobre. Solo ClientIota mostra una degradazione contenuta (−48.9%), mantenendo un F1 di 0.497 ad ottobre, mentre ClientBeta mantiene un livello accettabile in aprile (0.982) prima di collassare nei mesi successivi.

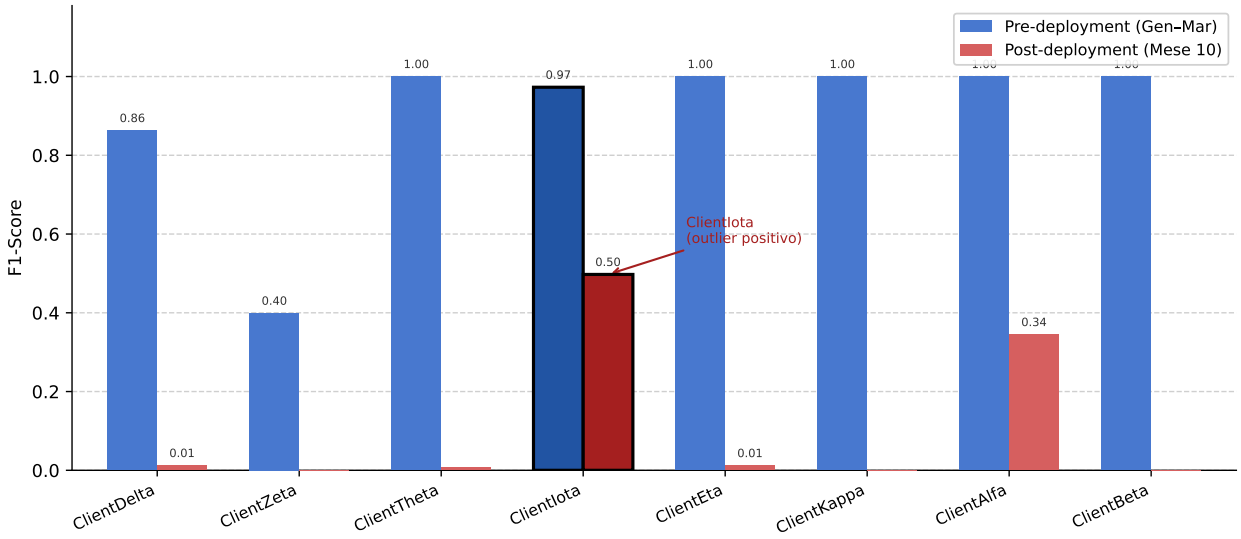

Figura 6.2: Confronto F1-Score pre-deployment (Gen-Mar 2025) vs post-deployment (Mese 10) per tutti i clienti del dataset. Il modello statico (RF) subisce un crollo quasi universale delle prestazioni a distanza di sette mesi dal training. ClientIota costituisce l'unico outlier positivo, mantenendo F1 = 0.50 al Mese 10 grazie a un volume di incidenti sufficientemente elevato e distribuito nel tempo.

Takeaway. Questa evidenza risponde direttamente alla prima domanda di ricerca della tesi: il concept drift non è un fenomeno teorico o marginale, ma una realtà operativa che colpisce sistematicamente i modelli ML-SOC su dati reali, e la sua entità è tale da rendere la valutazione statica una stima del tutto ottimistica delle prestazioni in produzione.

6.3.3 Selezione dei clienti focus per la valutazione incrementale

La valutazione settimanale con le strategie di mitigazione richiede un orizzonte temporale continuo e sufficientemente lungo da rendere significativo il calcolo dell'AUT. Dei dieci clienti presenti nel dataset, sei (ClientAlfa, ClientBeta, ClientGamma, ClientZeta, ClientEpsilon, ClientKappa) non soddisfano questo requisito: presentano serie storiche discontinue, periodi senza ticket sufficienti per la valutazione o orizzonti temporali troppo brevi per rilevare la deriva. Questi clienti compaiono quindi nella valutazione pre-deployment (Tabella 6.1); quattro di essi (ClientBeta, ClientGamma, ClientZeta, ClientEpsilon) figurano anche nell'analisi del drift statico (Tabella 6.2),

mentre ClientAlfa e ClientKappa non vi sono inclusi per l'insufficiente continuità delle rispettive serie storiche mensili.

I quattro clienti selezionati per la valutazione incrementale completa, ovvero ClientIota, ClientDelta, ClientTheta e ClientEta, sono stati scelti per la loro rappresentatività: coprono profili eterogenei di tasso di incidenza, volume di dati e intensità del drift, consentendo un'analisi comparativa significativa delle strategie proposte.

6.3.4 Post-deployment con mitigazione: guadagno dell'approccio incrementale

La Tabella 6.3 riporta i valori di AUT_{F1}, calcolato come integrale normalizzato della curva di F1-score settimanale, per i tre modelli (RF Statico, RF Incrementale, MLP Incrementale) sulla valutazione settimanale (settimane ISO 11-44) per i quattro clienti focus; i valori medi di F1 settimanale sottostanti sono riportati nelle analisi per cliente più avanti in questa sezione, in modo da rendere il confronto leggibile sulla stessa metrica usata in §6.3.1.

L'HGB, pur incluso nella valutazione pre-deployment (Tabella 6.1), non è stato esteso alla pipeline di apprendimento incrementale e non figura quindi in questo confronto. Anche l'MLP statico non viene riproposto come baseline settimanale separata: la valutazione pre-deployment (§6.3.1) ha già mostrato che l'MLP in configurazione statica non è competitivo con RF sui clienti a bassa densità di incidenti. Per questo motivo l'RF Statico è utilizzato come unica baseline operativa di riferimento per entrambe le strategie adattive: il confronto ΔRF (RF Incrementale vs RF Statico) costituisce l'unico confronto rigoroso a parità di modello, mentre ΔMLP esprime il guadagno dell'MLP Incrementale rispetto alla stessa baseline operativa, non un confronto a parità di algoritmo.

Tabella 6.3: AUT_{F1} normalizzato (settimane 11–44) per i tre modelli e i quattro clienti focus. In grassetto il valore migliore per cliente. ΔRF e ΔMLP indicano il guadagno rispetto alla baseline operativa (RF Statico); solo ΔRF rappresenta un confronto a parità di modello, poiché non è disponibile una variante statica settimanale dell'MLP (cfr. discussione nel testo).

Cliente	**RF Statico**	**RF Incr.**	**ΔRF**	**MLP Incr.**	**ΔMLP**
ClientIota	0.3813	0.3752	−0.006	**0.4035**	+0.022
ClientDelta	0.0006	**0.0191**	+0.019	0.0010	+0.000
ClientTheta	0.0447	0.0705	+0.026	**0.1481**	+0.103
ClientEta	0.0000	**0.0047**	+0.005	0.0018	+0.002

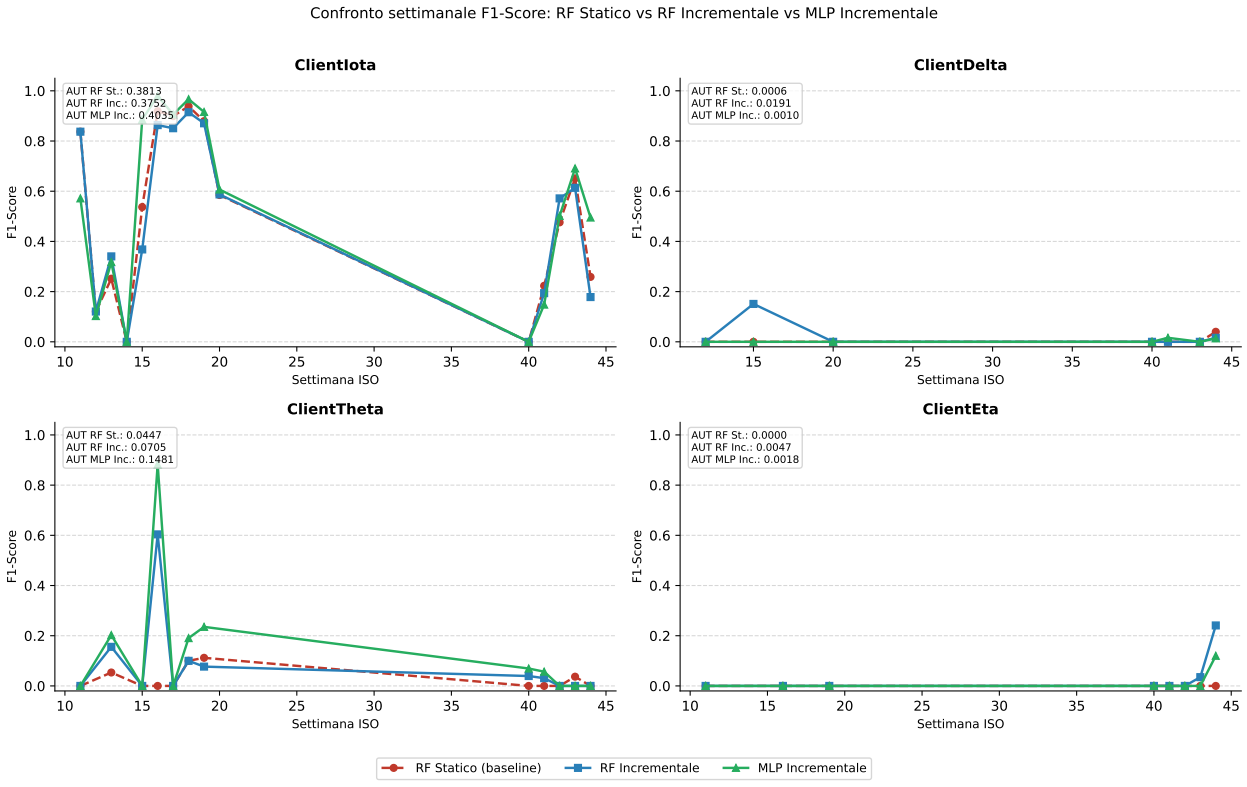

Figura 6.3: Confronto settimanale dell'F1-Score tra RF Statico (baseline), RF Incrementale e MLP Incrementale per i quattro clienti focus (settimane 10-44). I valori AUT normalizzati sono riportati nel riquadro di ciascun pannello. ClientIota è l'unico cliente con prestazioni sostenute nel tempo; ClientDelta e ClientEta mostrano F1 $\approx$ 0 per quasi tutto l'orizzonte di valutazione a causa della target sparsity.

I risultati mostrano comportamenti differenziati in funzione del cliente.

ClientIota Su ClientIota, il cliente con la distribuzione degli incidenti più regolare e il volume maggiore di dati etichettati, tutte e tre le strategie ottengono un AUT paragonabile (0.375-0.404). Il modello RF Statico (AUT 0.381) non viene significativamente superato dall'RF Incrementale (0.375, −0.006), mentre l'MLP Incrementale ottiene il miglior AUT (0.404, +0.022). Nella valutazione settimana per settimana, la media dell'F1 sulle 15 settimane valide è 0.505 (RF Statico), 0.487 (RF Incrementale) e 0.539 (MLP Incrementale), con recall medio rispettivamente di 0.627, 0.599 e 0.737. L'MLP mostra una capacità di recall superiore, catturando più incidenti a fronte di una precision leggermente inferiore.

ClientDelta Su ClientDelta la situazione è nettamente diversa. Il modello RF Statico ottiene un AUT di soli 0.0006 (sostanzialmente zero) in linea con la degradazione catastrofica documentata nello scenario pre-post. L'RF Incrementale porta l'AUT a 0.0191, con un guadagno relativo di oltre 30× rispetto allo statico, ma il valore assoluto rimane molto basso. Il guadagno è concentrato prevalentemente nella settimana 15, dove l'RF Incrementale raggiunge un F1 di 0.151 mentre il modello statico resta a 0. L'MLP Incrementale non mostra vantaggi su questo cliente (AUT 0.0010). Sulle 7 settimane valide, l'F1 medio settimanale è 0.006 (RF Statico), 0.024 (RF Incrementale) e 0.004 (MLP Incrementale): valori coerenti con l'AUT, che confermano prestazioni quasi nulle per tutte le strategie su questo cliente.

ClientTheta ClientTheta presenta il caso più incoraggiante per le strategie adattive. L'RF Incrementale (AUT 0.071) supera il modello statico (0.045) con un guadagno di +0.026, e l'MLP Incrementale ottiene il valore più alto (0.148) con un guadagno di +0.103, pari a un miglioramento relativo di +231%. Nella settimana 16, l'MLP Incrementale raggiunge un F1 di 0.882, dimostrando che in determinate settimane il sistema adattivo può produrre prestazioni elevate anche su un cliente con tasso di incidenza dell'1.78%. Tuttavia, la variabilità è elevatissima: l'F1 oscilla da 0 a 0.88 tra settimane consecutive. Sulle 12 settimane valide, l'F1 medio settimanale è 0.025 (RF Statico), 0.084 (RF Incrementale) e 0.136 (MLP Incrementale), confermando il vantaggio dell'approccio adattivo anche al netto dei picchi isolati.

ClientEta ClientEta è il cliente su cui tutte le strategie essenzialmente falliscono: RF Statico AUT 0.000, RF Incrementale 0.005, MLP Incrementale 0.002. In 7 delle 8 settimane valide, tutti i modelli ottengono F1 = 0. L'unica settimana significativa (settimana 44) produce F1 di 0.241 per l'RF Incrementale e 0.120 per l'MLP, ma si tratta di performance troppo isolate per avere impatto sull'AUT complessivo. L'F1 medio sulle 8 settimane valide è 0.000 (RF Statico), 0.035 (RF Incrementale) e 0.015 (MLP Incrementale): anche con l'aggiornamento continuo, il segnale resta troppo debole per un utilizzo operativo su questo cliente.

6.4 Analisi e interpretazione

6.4.1 Confronto per cliente: eterogeneità dei risultati e correlazione con il drift

La Tabella 6.4 confronta la strategia *single-client* (modello RF addestrato esclusivamente sui dati del cliente in valutazione) con la strategia *mixed-client* (modello addestrato su un mix di dati provenienti da più clienti contemporaneamente), sul test set ottobre-dicembre con split 8-1-3.

Tabella 6.4: Confronto tra esperimento single-client e mixed-client (RF) sul test set ottobre-dicembre (split 8-1-3). F1 e Recall sulla classe positiva.

Cliente	**Single-client**		**Mixed-client**	
	F1	Recall	F1	Recall
ClientDelta	0.008	0.005	0.019	0.015
ClientIota	0.528	0.578	0.499	0.522
ClientTheta	0.008	0.010	0.014	0.021
ClientEta	0.000	0.000	0.005	0.004

I risultati mostrano che il training misto non porta a miglioramenti sistematici rispetto al single-client: per ClientIota, il modello misto è addirittura leggermente peggiore (F1 0.499 vs 0.528),

mentre per i clienti a basso tasso di incidenza (ClientDelta, ClientTheta, ClientEta) i valori sono talmente ridotti da non consentire conclusioni statisticamente robuste. Questo risultato è coerente con la caratterizzazione del dataset esposta nella §5.3.2: la distribuzione degli alert è fortemente client-specific, e i modelli addestrati su dati di altri clienti non riescono a catturare i pattern peculiari del cliente target. La natura locale del concept drift rafforza ulteriormente questa osservazione: la deriva è guidata da eventi specifici del singolo cliente (aggiornamenti dell'infrastruttura, nuove campagne mirate) che non si generalizzano agli altri clienti del portafoglio.

6.4.2 Effetto isolato del Time Decay sulle performance

La Tabella 6.5 confronta le performance del modello RF con feature standard e feature di Time Decay ($\lambda = 0.05\,\mathrm{s}^{-1}$), sul test set ottobre-dicembre con split 8-1-3.

Tabella 6.5: Confronto tra feature standard e feature di Time Decay ($\lambda = 0.05\,\mathrm{s}^{-1}$) sul test set ottobre-dicembre (split 8-1-3)

Cliente	**Standard**			**Time Decay** ($\lambda = 0.05$)		
	F1	Rec.	Ticket	F1	Rec.	Ticket
ClientDelta	0.012	0.007	865	0.017	0.012	865
ClientZeta	0.000	0.000	0	0.000	0.000	0
ClientTheta	0.040	0.115	96	0.017	0.104	96
ClientIota	0.596	0.719	854	0.531	0.570	854
ClientEta	0.010	0.008	267	0.016	0.034	267

I risultati del confronto diretto (Figura 6.4) mostrano effetti misti: il Time Decay migliora leggermente l'F1 su ClientDelta (+0.005) e ClientEta (+0.006), ma lo riduce su ClientIota (−0.065) e ClientTheta (−0.023 F1, a fronte di recall pressoché invariato). Questa variabilità suggerisce che il beneficio del Time Decay dipende dalla struttura temporale specifica degli attacchi di ciascun cliente e non costituisce un miglioramento universale nel regime del singolo split statico. L'analisi di sensitività al parametro λ (Tabella 6.6) conferma l'assenza di un valore ottimale universale. La Tabella 6.6 riporta anche l'AUT_{F1} dell'MLP Incrementale al variare dell'architettura.

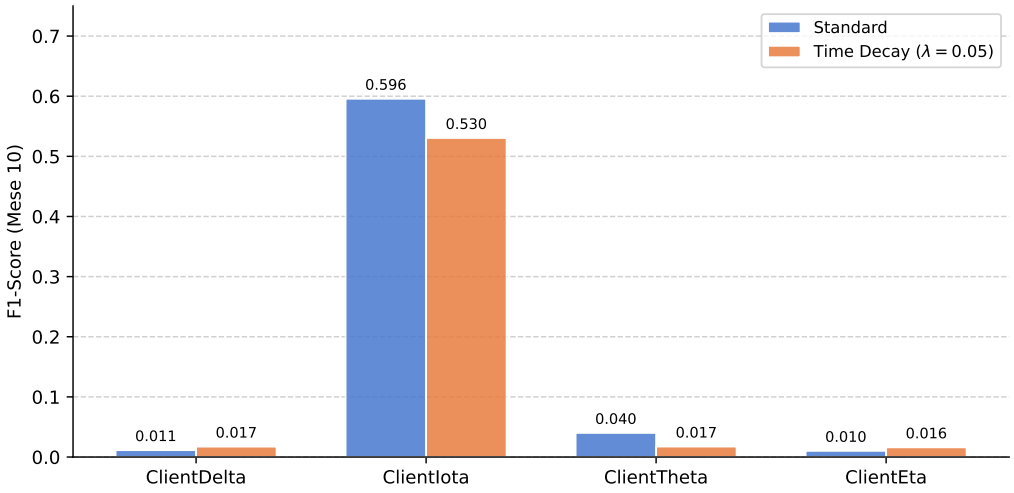

Figura 6.4: Confronto F1-Score tra feature standard e feature di Time Decay ($\lambda = 0.05\,\mathrm{s}^{-1}$) sul mese di test (Ottobre) per i quattro clienti focus. I valori numerici sono riportati sopra ciascuna barra.

Tabella 6.6: Sensitività dell'MLP Incrementale all'architettura: AUT_{F1} computato sulla sequenza settimanale (settimane 11–44).

Architettura	**ClientIota**	**ClientDelta**	**ClientTheta**	**ClientEta**
(50, 25)	0.4035	0.0002	0.1433	0.0038
(100, 50)	0.4016	0.0002	0.1408	0.0030
(100, 50, 25)	0.4043	0.0002	0.1392	0.0066

Le differenze di AUT_{F1} tra le architetture MLP sono marginali su tutti i clienti, il che giustifica la scelta dell'architettura più semplice (50, 25) come default. La Tabella 6.7 riporta l'analisi di sensitività al rapporto di undersampling.

Tabella 6.7: Sensitività al rapporto di undersampling r: AUT_{F1} del modello RF Incrementale (settimane 11–44). Media Recall tra parentesi.

Rapporto r	**ClientIota**	**ClientDelta**	**ClientTheta**	**ClientEta**
3	0.379 (0.623)	0.013 (0.037)	0.090 (0.252)	0.007 (0.067)
5	0.375 (0.599)	0.006 (0.014)	0.051 (0.180)	0.004 (0.035)
10	0.363 (0.549)	0.000 (0.001)	0.064 (0.147)	0.004 (0.024)

Il rapporto $r = 3$ tende a massimizzare l'AUT_{F1} e il recall medio, in particolare per i clienti a basso tasso di incidenza (ClientDelta, ClientEta e ClientTheta) dove aumentare la presenza della classe positiva nel training migliora la sensibilità del classificatore. Tuttavia, $r = 3$ riduce la stabilità della precisione su settimane con pochi ticket. Il rapporto $r = 5$ viene mantenuto come valore di default per la valutazione principale come compromesso tra recall e precisione.

6.4.3 Analisi della generalizzazione cross-client: natura locale del Concept Drift

Come mostrato dalla Tabella 6.4, la distribuzione degli alert è fortemente client-specific: i modelli addestrati su dati di più clienti contemporaneamente non portano a miglioramenti sistematici rispetto ai modelli addestrati sul singolo cliente. Questo risultato ha implicazioni dirette sulla natura del concept drift osservato: la deriva non è un fenomeno globale che interessa uniformemente tutti i clienti del portafoglio (es. un aggiornamento del ruleset Suricata), bensì un fenomeno locale guidato da eventi specifici di ciascun cliente (nuove campagne malevole mirate, cambiamenti nell'infrastruttura, variazioni stagionali del traffico). La strategia di modelli separati per cliente, adottata in questa tesi, è pertanto la scelta metodologicamente corretta: un modello unificato non sarebbe in grado di adattarsi simultaneamente alle derive di tutti i clienti senza degradare le prestazioni su ciascuno.

6.4.4 Discussione critica dei risultati e limitazione del sistema

I risultati presentati in questo capitolo impongono l'ammissione di alcuni limiti insieme ai contributi positivi della tesi.

Limitazione 1: il sistema funziona bene solo su ClientIota L'AUT$_{F1}$ superiore a 0.1 è raggiunto solo su ClientIota (0.40) e parzialmente su ClientTheta con MLP Incrementale (0.148). Su ClientDelta e ClientEta tutte le strategie producono AUT nell'ordine di 10^{-2} o inferiore, il che equivale operativamente a un sistema che non funziona. Attribuire questo fallimento esclusivamente al concept drift sarebbe fuorviante: la target sparsity (ClientDelta: 4.89‰, ClientEta: 3.34‰), la concentrazione degli incidenti in pochi picchi e l'irregolarità del ground truth (derivato da un processo di triage manuale soggetto a latenze) sono fattori strutturali che limitano la capacità di qualunque classificatore, indipendentemente dalla strategia di aggiornamento.

Limitazione 2: l'RF Incrementale non supera l'RF Statico su ClientIota Su ClientIota, il modello statico (AUT 0.381) ottiene AUT paragonabile all'RF Incrementale (0.375). Questo suggerisce che la finestra crescente di dati storici non apporta un vantaggio netto su un cliente con drift graduale: i dati più vecchi contribuiscono rumore che compensa il beneficio dei dati recenti. La rolling window a dimensione fissa (3 mesi), non testata direttamente in questa configurazione settimanale, potrebbe offrire un compromesso migliore.

Limitazione 3: il Time Decay non produce miglioramenti robusti Il parametro λ è stato ricavato analiticamente dai dati ClientIota ($\lambda = 0.05$) e applicato uniformemente a tutti i clienti. L'assenza di un'ottimizzazione per-cliente e la limitata variabilità di λ testata (un solo valore nella valutazione incrementale settimanale) rendono non conclusivi i risultati sul Time Decay. Una valutazione più sistematica, con ottimizzazione di λ su un set di validazione separato per cliente, è necessaria per trarre conclusioni definitive.

Limitazione 4: il protocollo di valutazione settimanale contiene un'anomalia di ordinamento. Il meccanismo di ordinamento per settimana ISO senza disambiguazione dell'anno fa sì che, per clienti con dati a cavallo tra la fine del 2024 e l'inizio del 2025, le settimane 40-44 (ottobre-

novembre 2024) risultino *cronologicamente antecedenti* alle settimane 1-8 (gennaio-febbraio 2025), ma vengano ordinate *dopo* di esse nell'asse temporale del training. Questo introduce un parziale *temporal leak* nelle settimane di test corrispondenti a ottobre-novembre 2024, poiché il modello è stato addestrato su dati di mesi successivi. I risultati delle settimane 40-44 devono pertanto essere interpretati con cautela. Una soluzione robusta, da implementare in sviluppi futuri, consiste nell'usare il timestamp assoluto come chiave di ordinamento invece della settimana ISO.

Sintesi del posizionamento Nonostante le limitazioni descritte, la tesi raggiunge gli obiettivi dichiarati: il concept drift viene documentato empiricamente con misure quantitative su dati operativi reali; le strategie adattive (in particolare l'MLP Incrementale) mostrano un guadagno significativo su ClientTheta (+231% sull'AUT rispetto allo statico) e su ClientDelta (+30×), dimostrando che l'aggiornamento continuo è necessario anche se non sufficiente a garantire prestazioni elevate su tutti i clienti. Il principale contributo è metodologico: l'adozione dell'AUT come metrica di valutazione e il protocollo di simulazione del deployment settimanale forniscono un framework replicabile per la valutazione di sistemi ML-SOC in produzione, indipendentemente dalla specifica architettura del classificatore.

7. Conclusioni

Questa tesi ha affrontato il problema del concept drift nei sistemi di triage automatizzato basati su Machine Learning operanti all'interno di un Security Operations Center. Il punto di partenza era una lacuna precisa nella letteratura e nei sistemi ML-SOC esistenti, tipicamente valutati in modo statico, senza misurare né mitigare la degradazione delle loro prestazioni nel tempo. Il punto di arrivo è un framework sperimentale completo (con metriche, protocolli e strategie adattive) applicato su dati operativi reali provenienti da dieci clienti di un MSSP italiano.

7.1 Sintesi dei traguardi raggiunti

Documentazione empirica del concept drift Il primo e più importante risultato della tesi è la dimostrazione quantitativa che il concept drift è un fenomeno reale, sistematico e di entità rilevante nel dominio ML-SOC. Un modello Random Forest addestrato sui dati di gennaio-marzo 2025 e valutato in ottobre, senza alcun aggiornamento, perde tra il 48.9% (ClientIota) e il 99.3% (ClientTheta) del proprio F1-score. Sei degli otto clienti osservabili mostrano un collasso quasi totale delle prestazioni (F1 inferiore a 0.02) già a pochi mesi dal deployment. Questa evidenza risponde in modo definitivo alla prima domanda di ricerca: il concept drift non è trascurabile e la valutazione statica sovrastima sistematicamente le prestazioni di un sistema ML-SOC in produzione.

Framework di valutazione temporalmente corretto Il secondo contributo è metodologico: l'adozione della metrica Area Under Time, calcolata come integrale normalizzato della curva F1-score settimanale, fornisce uno strumento comparativo robusto che cattura sia il livello medio delle prestazioni sia la loro variabilità nel tempo. Il protocollo di simulazione del deployment (training sulle prime 8 settimane, valutazione settimana per settimana senza look-ahead) costituisce un riferimento replicabile per chiunque voglia valutare sistemi ML-SOC su dati con struttura temporale, ed è allineato alle raccomandazioni del framework TESSERACT [14].

Strategie di mitigazione adattiva Il terzo contributo riguarda il confronto tra tre strategie di aggiornamento del modello: il modello statico (baseline), il modello incrementale a finestra cre-

scente (RF e MLP) e, come variante esplorata, il modello con rolling window trimestrale. I risultati mostrano che le strategie adattive apportano miglioramenti significativi rispetto alla baseline su tutti i clienti in cui il sistema produce segnale misurabile: su ClientTheta, l'MLP Incrementale ottiene un AUT di 0.148 contro 0.045 dello statico (+231%); su ClientDelta, l'RF Incrementale porta l'AUT da 0.0006 a 0.0191 (+30×); su ClientIota, l'MLP Incrementale supera il modello statico di +0.022 AUT con un recall medio settimanale del 73.7%. Questi risultati rispondono alla seconda domanda di ricerca: l'aggiornamento continuo è necessario e produce guadagni misurabili, anche se non sufficiente a garantire prestazioni elevate su clienti con distribuzione sparsa degli incidenti.

Time Decay e feature engineering temporale L'introduzione delle feature di Time Decay, che ponderano gli eventi nella finestra temporale in modo esponenzialmente decrescente rispetto alla loro età, non ha prodotto miglioramenti uniformi rispetto alle feature standard. Su ClientDelta e ClientEta si osserva un miglioramento marginale dell'F1 (+0.005 e +0.006 rispettivamente) nello scenario di split statico 8-1-3, mentre su ClientIota e ClientTheta il Time Decay riduce leggermente l'F1. Questo risultato suggerisce che la ponderazione temporale è sensibile alla struttura specifica dei pattern di attacco del cliente: la sua efficacia richiede una calibrazione del parametro λ per-cliente che non è stata eseguita in questa versione.

Natura client-specific del drift Infine, gli esperimenti di generalizzazione cross-client confermano che il concept drift nel dominio ML-SOC è fortemente locale: i modelli addestrati su pool di clienti eterogenei non superano sistematicamente i modelli addestrati sul singolo cliente. Questo risultato, coerente con quanto osservato nei lavori precedenti, ha implicazioni dirette per l'architettura operativa del SOC: sono necessari modelli per-cliente e strategie di adattamento per-cliente, non un unico modello universale.

7.2 Considerazioni sull'integrazione reale in un SOC di produzione

Prerequisiti di qualità del ground truth La condizione necessaria affinché qualsiasi strategia di retraining funzioni è la disponibilità di un ground truth tempestivo e affidabile. Il modello

incrementale apprende dalla chiusura dei ticket da parte degli analisti: se la latenza tra l'evento e la decisione dell'analista è elevata (giorni, non ore), il segnale di aggiornamento arriva troppo tardi per inseguire il drift. L'integrazione operativa richiede pertanto processi di triage che producano etichette con latenza inferiore alla settimana, ovvero compatibile con la granularità del ciclo di retraining.

Ciclo settimanale di retraining Il protocollo implementato è operativamente fattibile: l'addestramento di un Random Forest o MLP su qualche decina di migliaia di campioni sparsi richiede tempi nell'ordine dei secondi su hardware commodity. Un ciclo settimanale di retraining, che si attiva automaticamente al completamento della settimana di osservazione, introduce un overhead computazionale trascurabile rispetto al volume di operazioni di un SOC reale. Il costo dominante è invece organizzativo: la raccolta e l'etichettatura dei nuovi dati, il monitoraggio delle metriche di performance e il triggering del ciclo quando la performance scende sotto una soglia di allerta.

Gestione dei clienti con target sparsity Per i clienti con tasso di incidenza inferiore al 5% (ClientDelta, ClientEta, ClientTheta), il framework proposto non è sufficiente allo stato attuale. Su questi clienti la frequenza degli incidenti è troppo bassa per addestrare un classificatore stabile su finestre settimanali. Una possibile soluzione operativa è l'adozione di una soglia di attivazione del retraining basata sul numero di nuovi incidenti osservati: il modello viene aggiornato solo quando vengono accumulate almeno k nuove istanze positive (es. $k = 20$), indipendentemente dalla finestra temporale.

Architettura multitenancy L'evidenza della natura client-specific del drift implica che il SOC debba mantenere un modello distinto per ciascun cliente, con vocabolario TF-IDF e parametri del classificatore ri-fittati indipendentemente. Dal punto di vista architetturale, ciò si traduce in un orchestratore di pipeline per-cliente, in cui ogni istanza gestisce il proprio ciclo di raccolta dati, feature engineering, retraining e deployment. La scalabilità di questa architettura è lineare nel numero di clienti e non richiede modifiche all'algoritmo di apprendimento.

7.3 Sviluppi futuri

I risultati e le limitazioni documentate aprono diverse direzioni di ricerca e sviluppo che si ritengono prioritarie.

Ottimizzazione per-cliente del parametro λ Il valore $\lambda = 0.05$ è stato derivato analiticamente dalla struttura del dataset ClientIota e applicato uniformemente. Un'ottimizzazione per-cliente di λ, tramite validazione su un insieme di settimane trattenute, potrebbe rivelare se il Time Decay produce miglioramenti sistematici quando calibrato sulla distribuzione temporale specifica di ciascun cliente. In alternativa, λ potrebbe essere appreso come iperparametro del classificatore attraverso una ricerca a griglia con cross-validation temporale.

Rilevamento automatico del drift Il framework attuale applica il retraining in modo periodico e cieco, indipendentemente dall'entità del drift effettivamente osservato. L'integrazione di un meccanismo di *drift detection* (ispirato a Transcend [11] o basato su test statistici sulla distribuzione degli score del classificatore) consentirebbe di attivare il retraining quando necessario piuttosto che a cadenza fissa, riducendo il numero di aggiornamenti nei periodi stabili e concentrandoli nei periodi di deriva intensa.

Ottimizzazione della rolling window per-cliente Il valore di $N = 3$ mesi è stato scelto euristicamente. Una selezione adattiva della dimensione della finestra (ad esempio calibrata sulla velocità di drift stimata dal classificatore stesso) potrebbe migliorare il compromesso tra stabilità statistica e reattività al cambiamento in modo dipendente dal profilo del cliente.

Miglioramento del ground truth tramite semi-supervised learning Su clienti con target sparsity elevata, la scarsità di etichette positive è il principale collo di bottiglia. Tecniche di apprendimento semi-supervisionato (come il pseudo-labeling usato da INSOMNIA [2]) potrebbero permettere di sfruttare anche le sessioni non etichettate per aggiornare la rappresentazione interna del classificatore, riducendo la dipendenza da un ground truth tempestivo.

Federated Learning per la condivisione controllata della conoscenza L'addestramento separato per cliente non permette di trasferire conoscenza tra clienti che subiscono drift simili in periodi comparabili (es. stessa campagna di attacco che colpisce più clienti simultaneamente). Un approccio federato, in cui ogni cliente contribuisce un aggiornamento locale a un modello globale senza condividere i dati grezzi, potrebbe migliorare la robustezza sui clienti con pochi incidenti, preservando al contempo la riservatezza dei dati.

Correzione dell'ordinamento temporale inter-annuale Come evidenziato nella §6.4.4, il protocollo di valutazione basato sul numero di settimana ISO senza disambiguazione dell'anno introduce un'anomalia per i dataset che coprono due anni solari consecutivi. Una riformulazione del protocollo basata su timestamp assoluti (espressi come settimane dall'inizio della raccolta dati piuttosto che come settimane dell'anno) eliminerebbe questa anomalia garantendo un ordinamento temporale rigoroso.

Bibliografia

[1] Bushra A. Alahmadi, Louise Axon e Ivan Martinovic. «99% False Positives: A Qualitative Study of SOC Analysts' Perspectives on Security Alarms». In: *Proceedings of the 31st USENIX Security Symposium*. Boston, MA, USA: USENIX Association, 2022, pp. 2783–2800. ISBN: 978-1-939133-31-1.

[2] Giuseppina Andresini et al. «INSOMNIA: Towards Concept-Drift Robustness in Network Intrusion Detection». In: *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security (AISec '21)*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 87–98. DOI: 10.1145/3474369.3486864.

[3] Giovanni Apruzzese, Pavel Laskov e Johannes Schneider. «The Role of Machine Learning in Cybersecurity». In: *ACM Digital Threats: Research and Practice* 4.1 (2023), pp. 1–38. DOI: 10.1145/3545574.

[4] Tao Ban et al. «Combat Security Alert Fatigue with AI-Assisted Techniques». In: *Proceedings of the Workshop on Cyber Security Experimentation and Test (CSET '21)*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 9–16. DOI: 10.1145/3474718.3474723.

[5] Leo Breiman. «Random Forests». In: *Machine Learning* 45.1 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324.

[6] Fabrício Ceschin et al. «Fast & Furious: On the Modelling of Malware Detection as an Evolving Data Stream». In: *Expert Systems with Applications* 212 (2023), p. 118590. DOI: 10.1016/j.eswa.2022.118590.

[7] Jerome H. Friedman. «Greedy Function Approximation: A Gradient Boosting Machine». In: *The Annals of Statistics* 29.5 (2001), pp. 1189–1232. DOI: 10.1214/aos/1013203451.

[8] João Gama et al. «A Survey on Concept Drift Adaptation». In: *ACM Computing Surveys* 46.4 (2014), pp. 1–37. DOI: 10.1145/2523813.

[9] Ben Gelman et al. *That Escalated Quickly: An ML Framework for Alert Prioritization*. Sophos AI Research. 2023. arXiv: 2302.06648 [cs.CR].

[10] Jalal Ghadermazi, Anoop Shah e Sushil Jajodia. «A Machine Learning and Optimization Framework for Efficient Alert Management in a Cybersecurity Operations Center». In: *ACM Digital Threats: Research and Practice* 5.2 (2024). DOI: 10.1145/3644393.

[11] Roberto Jordaney et al. «Transcend: Detecting Concept Drift in Malware Classification Models». In: *Proceedings of the 26th USENIX Security Symposium*. Vancouver, BC, Canada: USENIX Association, 2017, pp. 625–642. ISBN: 978-1-931971-40-9.

[12] Open Information Security Foundation. *Suricata Network IDS/IPS Engine*. https://suricata.io. Versione 7.x. Ultimo accesso: aprile 2026. 2023.

[13] Fabian Pedregosa et al. «Scikit-learn: Machine Learning in Python». In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[14] Feargus Pendlebury et al. «TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time». In: *Proceedings of the 28th USENIX Security Symposium*. Santa Clara, CA, USA: USENIX Association, 2019, pp. 729–746. ISBN: 978-1-939133-06-9.

[15] Gerard Salton e Christopher Buckley. «Term-Weighting Approaches in Automatic Text Retrieval». In: *Information Processing & Management* 24.5 (1988), pp. 513–523. DOI: 10.1016/0306-4573(88)90021-0.

[16] Robin Sommer e Vern Paxson. «Outside the Closed World: On Using Machine Learning for Network Intrusion Detection». In: *Proceedings of the 2010 IEEE Symposium on Security and Privacy (S&P)*. Oakland, CA, USA: IEEE, 2010, pp. 305–316. DOI: 10.1109/SP.2010.25.

[17] Kalyan Veeramachaneni et al. «AI^2: Training a Big Data Machine to Defend». In: *Proceedings of the 2nd IEEE International Conference on Big Data Security on Cloud (BigDataSecurity)*. IEEE, 2016, pp. 49–54. DOI: 10.1109/BigDataSecurity-HPSC-IDS.2016.79.

[18] Manfred Vielberth et al. «Security Operations Center: A Systematic Study and Open Challenges». In: *IEEE Access* 8 (2020), pp. 227756–227779. DOI: 10.1109/ACCESS.2020.3045514.

Ringraziamenti

Alla mia famiglia

Grazie per il sostegno continuo e per avermi incoraggiato in ogni scelta, anche quando sembrava difficile o impossibile. Grazie per la vostra pazienza, il vostro amore e il vostro supporto incondizionato.

Ai miei amici

Grazie per aver condiviso con me momenti di gioia e di difficoltà e per farmi sentire sempre parte di qualcosa di speciale.

Al prof. Mirco Marchetti e all'ing. Dimitri Galli

Grazie per la guida, il supporto e i preziosi consigli durante la stesura della tesi: la vostra esperienza e competenza sono state fondamentali per il completamento di questo lavoro.

A Clelia

Devo trovare il modo giusto di ringraziarti per tutto ciò che hai fatto per me senza essere bannale, scontato o ripetitivo. Grazie per esserci in ogni momento, anche quando sei stanca e magari non hai voglia di ascoltarmi, ma lo fai comunque. Grazie per il supporto che mi dai ogni giorno, per dirmi sempre che sono bravo e che ce la posso fare: io sono sicuro che con te posso fare ed affrontare qualsiasi cosa, perchè in due è tutto più bello e semplice. Grazie per tutto l'amore incondizionato che mi dai e per tutto ciò che facciamo insieme, anche le cose più semplici. Grazie per essere la mia compagna di vita e per avermi fatto capire che l'amore vero non esiste solo nelle fiabe, ma anche nella vita reale. E come qualcuno che entrambi conosciamo disse: "la felicità la si può trovare anche negli attimi più tenebrosi, se solo ci si ricorda di accendere la luce"; tu sarai la mia luce, e io sarò la tua per sempre.

Per me sei tutto.

Ti amo, amore mio.