



UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

DIPARTIMENTO DI INFORMATICA

CORSO DI LAUREA IN INFORMATICA

TESI DI LAUREA
IN
MODELLO E METODI PER LA SICUREZZA DELLE APPLICAZIONI

Metodi e Strumenti per la Supervisione di Reti per le PMI

RELATORE:

Prof. Donato Impedovo

LAUREANDO:

Tommaso Orlando

CORRELATORI:

Dott. Danilo Caivano

Dott. Vincenzo Gattulli

Dott. Michele Crudele

ANNO ACCADEMICO 2024 - 2025

“Un uomo che sia padrone di se stesso può porre fine ad un dolore con tanta facilità quanto può inventarsi un piacere.

Non voglio essere in balia delle mie emozioni. Voglio solo servirmene, goderle e dominarle.”

— Oscar Wilde

Questo traguardo è innanzitutto dedicato a me stesso e a tutti coloro che hanno creduto in me. In misura più importante però, questo lavoro va ai miei nonni, nonno Masino e nonna Lina. Spero che possano continuare a guidare i miei passi, vegliando su di me da quell’infinito che li accoglie.

— Tommaso Orlando

“Nothing Special”

Disclaimer

Tutti i marchi, nomi commerciali, prodotti, loghi e framework menzionati in questa tesi sono di proprietà dei rispettivi titolari. L'autore non rivendica alcun diritto di proprietà su tali elementi e li utilizza esclusivamente a scopo informativo, descrittivo e accademico, senza alcun intento di violazione o uso commerciale, in linea con la normativa sul diritto d'autore e la proprietà intellettuale.

Abstract

Le Piccole e Medie Imprese (PMI) affrontano notevoli sfide nella gestione di infrastrutture IT ibride, principalmente a causa della **mancanza di visibilità unificata** su sistemi eterogenei. Questa tesi, in collaborazione con l'azienda Evolumia, propone la progettazione e lo sviluppo parziale di **EPS+**, un sistema di monitoraggio open source.

L'obiettivo di EPS+ è integrare dati frammentati da diverse fonti (come **Proxmox VE**, **Acronis Cyber Protect** e dispositivi di rete monitorati via **SNMP**), per rilevare proattivamente le anomalie tramite **machine learning** e calcolare un “**health score**” aggregato. Questo punteggio, basato su metriche di performance, availability e resilience, offre una valutazione immediata e intuitiva dello stato dell'infrastruttura.

Dato che gli ambienti PMI sono eterogenei e spesso privi di dati etichettati, il progetto si concentra su un approccio non supervisionato. Il nucleo sperimentale si basa sull'algoritmo **Isolation Forest**, convalidato su dataset SNMP-MIB e su dati generati. Il confronto con benchmark supervisionati (come Random Forest e kNN) ha dimostrato l'efficienza di Isolation Forest nel modellare la “normalità” senza la necessità di etichette, mantenendo **performance** in termini di accuratezza e velocità di elaborazione.¹

Il sistema esplora anche un approccio più avanzato con **autoencoder basati su GRU** per l'analisi di pattern sequenziali, dimostrando potenzialità di sviluppo future. In questo contesto, Isolation Forest agisce come un **passo strategico iniziale**: identifica le anomalie in tempo reale e accumula dati validati che possono essere usati per addestrare modelli più complessi.

Parole Chiave: Anomaly Detection, Machine Learning, Monitoraggio Infrastruturale, PMI, Isolation Forest, Health Score, Visibilità Unificata, SNMP, Open Source, GRU Autoencoder.

¹Il codice sorgente è disponibile su GitHub:https://github.com/Strix89/PMI_DashboardEPS

Indice

1	Introduzione	1
1.1	Contesto e Motivazioni	1
1.2	Limiti delle Soluzioni Esistenti	2
1.2.1	Elastic Stack (ELK Stack)	2
1.2.2	Prometheus & Grafana	3
1.2.3	Splunk	3
1.2.4	Zabbix	3
1.3	Tecnologie e Strumenti di Riferimento	4
1.3.1	Acronis Cyber Protect	4
1.3.2	Address Resolution Protocol (ARP)	5
1.3.3	AJAX (Asynchronous JavaScript and XML)	5
1.3.4	Chart.js	5
1.3.5	Flask	6
1.3.6	HTML, CSS e JavaScript	6
1.3.7	MongoDB	6
1.3.8	Nmap (Network Mapper)	7
1.3.9	Proxmox Virtual Environment (PVE)	7
1.3.10	Python	7
1.3.11	Simple Network Management Protocol (SNMP)	8
1.3.12	Socket.IO	8
1.4	Obiettivi della Tesi	8
1.5	Struttura del documento	9
2	Stato dell'Arte	10
2.1	Monitoraggio delle infrastrutture IT	10
2.2	Tecniche di Anomaly Detection	13
2.2.1	Definizioni e Tipologie di Anomalie	13
2.2.2	Paradigmi di Apprendimento per il Rilevamento	13
2.2.3	Algoritmi di Riferimento nello Stato dell'Arte	14
2.2.4	L'Approccio Statistico e la Soluzione ai Problemi del Monitoraggio	16
2.2.5	La Sfida del Bilanciamento del Dataset e la Tecnica SMOTE	18

3	Dataset	20
3.1	Dataset SNMP-MIB	20
3.1.1	Metodologia di Raccolta e Generazione	21
3.2	Dataset Generato	24
3.2.1	Motivazioni e Giustificazione Metodologica	24
3.2.2	Modellazione dei Dati e Corrispondenza con API Reali	25
3.2.3	Metodologia di Generazione e Implementazione	27
4	Proposizioni di Metodi e Tecniche	31
4.1	Metodologie per Dataset SNMP-MIB	31
4.1.1	Tecniche di Preprocessing dei Dati	31
4.1.2	Gestione dello Sbilanciamento delle Classi	32
4.1.3	Modelli di Rilevamento a Confronto	33
4.1.4	Metodologia di Validazione	35
4.1.5	Metriche di Valutazione	37
4.2	Simulazione con Dataset Generato e Calcolo degli Health Score	38
4.2.1	Metodologia per lo Score di Performance (SPC)	39
4.2.2	Implementazione del Generatore di Dati di Performance	42
4.2.3	Implementazione della Simulazione Interattiva	44
4.2.4	Metodologia per lo Score di Availability	48
4.2.5	Implementazione del Generatore di Dati di Availability	49
4.2.6	Implementazione della Simulazione Interattiva	50
4.2.7	Metodologia e Implementazione per lo Score di Resilience	53
4.2.8	Implementazione del Generatore di Dati di Resilience	54
4.2.9	Implementazione della Simulazione Interattiva	55
5	Sperimentazione	58
5.1	Setup Sperimentale e Ambiente di Sviluppo	58
5.1.1	Preprocessing dei Dati e Preparazione degli Esperimenti	59
5.2	Esperimento 1: Confronto Supervisionato vs. Non Supervisionato	60
5.2.1	Obiettivo e Protocollo Sperimentale	60
5.2.2	Dettagli Implementativi	61
5.3	Esperimento 2: Ottimizzazione di Isolation Forest	61
5.3.1	Obiettivo e Protocollo Sperimentale	61
5.3.2	Dettagli Implementativi	62
5.4	Esperimento 3: Analisi Temporale con GRU-Autoencoder	62
5.4.1	Obiettivo e Protocollo Sperimentale	62
5.4.2	Dettagli Implementativi	62
6	Risultati	65
6.1	Esperimento 1	65
6.1.1	Risultati su Dataset Sottocampionato	65

6.1.2	Risultati su Dataset Sovracampionato (SMOTE)	67
6.2	Esperimento 2	67
6.3	Esperimento 3	68
6.3.1	Fallimento del primo tentativo (8 feature)	68
6.3.2	Analisi del fallimento e ipotesi risolutiva	69
6.3.3	Risultati con set di feature esteso (34 feature)	69
7	Proposta progettuale: Dashboard EPS+	71
7.1	Architettura Generale del Sistema EPS+	71
7.1.1	Data Sources Layer	72
7.1.2	Storage Layer	73
7.1.3	Processing Layer	73
7.1.4	Presentation Layer	73
7.2	Modulo di Network Discovery	74
7.2.1	Pipeline	74
7.2.2	Componenti Chiave	75
7.2.3	Profili di Scansione	76
7.3	Il Prototipo Dimostrativo: Funzionalità	77
7.3.1	Integrazione con Proxmox VE e Acronis Cyber Protect	77
7.3.2	Le Dashboard di Scoring	79
8	Conclusioni e Sviluppi Futuri	87
8.1	Conclusioni	87
8.2	Sviluppi Futuri	88
8.2.1	Completamento Architetturale e Integrazione dei Moduli	88
8.2.2	Potenziamento delle Capacità Analitiche	88
8.2.3	Verso un Sistema Multimodale e Interpretabile	89
Bibliografia		90
Ringraziamenti		93

Capitolo 1

Introduzione

Il presente capitolo introduce il lavoro di tesi, delineando il percorso logico che ne ha guidato lo sviluppo. Si partirà dalla definizione del **contesto** e delle **motivazioni** alla base della ricerca, analizzando poi i limiti delle **soluzioni di monitoraggio esistenti**. Successivamente, verranno presentate le **tecnologie di riferimento** e definiti con chiarezza gli **obiettivi specifici** del progetto, per concludere con una descrizione della **struttura del documento**.

1.1 Contesto e Motivazioni

La gestione efficace delle infrastrutture IT rappresenta una sfida crescente per le Piccole e Medie Imprese (**PMI**). A differenza delle realtà di grandi dimensioni, le PMI operano tipicamente con vincoli di budget e con personale non sempre specializzato in tutte le aree del *governo IT*. Questa situazione le espone a un rischio maggiore di interruzioni di servizio e di incidenti di sicurezza, poiché la supervisione dell'infrastruttura è spesso gestita con un approccio reattivo, intervenendo solo a seguito di un problema, piuttosto che attraverso un monitoraggio avanzato e proattivo.

Questa complessità gestionale è aggravata dalla diffusa adozione di architetture **IT ibride**, che affiancano risorse locali (*on-premise*) a servizi cloud. Tale approccio, sebbene flessibile, introduce una frammentazione dei dati che ostacola una supervisione centralizzata. Nello specifico, le **PMI** clienti dell'azienda Evolumia implementano uno stack tecnologico standardizzato che, pur essendo efficiente, è composto da sistemi eterogenei che richiedono un monitoraggio integrato. Questo ecosistema include tipicamente:

- **sistemi di virtualizzazione**, come Proxmox Virtual Environment (*PVE*), che ospitano le macchine virtuali e container LXC per le operazioni aziendali;

- **soluzioni di backup e disaster recovery**, sia locali con Proxmox Backup Server (PBS) sia in cloud tramite servizi come *Acronis Cyber Protect*, fondamentali per garantire la continuità operativa;
- **dispositivi di rete** di varia natura (router, switch, firewall) che, pur in contesti dimensionalmente contenuti, generano un flusso costante di dati sul proprio stato operativo;

È in questo scenario che si colloca il presente lavoro di tesi. La ricerca quindi nasce da un'esigenza concreta espressa dall'azienda Evolumia, specializzata in sicurezza informatica e gestione di infrastrutture per le **PMI**. L'obiettivo dell'azienda, formalizzato nel **progetto EPS+**, è quello di "*creare un ecosistema ibrido per la gestione integrata di infrastruttura e security delle piccole realtà, che dia una immediata percezione dello stato di sicurezza della propria infrastruttura*". Questa tesi si propone di affrontare il nucleo di questa sfida, sviluppando e validando i componenti di monitoraggio e analisi.

1.2 Limiti delle Soluzioni Esistenti

Per affrontare la sfida proposta da Evolumia, il mercato offre soluzioni open source potenti e mature. Tuttavia, un'analisi rivela come la loro applicazione standard non risponda pienamente alle esigenze di un monitoraggio proattivo e integrato per le PMI. La presente tesi si posiziona come un'alternativa che mira a superare questi limiti specifici.

1.2.1 Elastic Stack (ELK Stack)

Precedentemente noto come ELK Stack, è una suite di prodotti progettata per la ricerca, l'analisi e la visualizzazione di grandi volumi di dati, con un focus primario sull'analisi dei log¹. La sua potenza è innegabile: **Elasticsearch** è un motore di ricerca e analisi performante e scalabile; **Logstash** e **Beats** permettono di ingerire dati da quasi qualsiasi fonte; e **Kibana** offre capacità di visualizzazione avanzate e interattive, rendendola la soluzione de facto per la centralizzazione dei log. Tuttavia, i suoi svantaggi emergono quando la si vuole adattare agli scopi del progetto EPS+. Le potenti funzionalità di **machine learning per l'anomaly detection sono incluse solo nelle licenze commerciali a pagamento**, rendendole inaccessibili per una soluzione puramente open source. Inoltre, la gestione di metriche da protocolli come SNMP non è nativa, richiedendo configurazioni complesse, ed è una soluzione nota per essere **resource intensive**, un fattore non trascurabile per le PMI.

¹Il sito ufficiale di Elastic è disponibile all'indirizzo: <https://www.elastic.co/>

1.2.2 Prometheus & Grafana

Prometheus, spesso utilizzato in coppia con Grafana, è un sistema di monitoraggio e alerting open source diventato lo standard de facto per il monitoraggio di ambienti dinamici, containerizzati e cloud-native². La sua architettura è ottimizzata per la gestione di **dati time-series** e il suo modello dati basato su etichette (labels) offre una flessibilità senza pari nell'interrogazione tramite il linguaggio PromQL. **Grafana**, d'altro canto, fornisce un'interfaccia utente moderna, reattiva e personalizzabile. Nonostante l'eccellenza, questa combinazione presenta delle lacune rispetto agli obiettivi della tesi: Prometheus è progettato per le metriche, ma **non gestisce i log**, creando di fatto un silo separato. Il suo modello di raccolta dati è esclusivamente **pull-based**, il che può complicare il monitoraggio di dispositivi dietro firewall. Infine, né Prometheus né Grafana includono funzionalità native di **anomaly detection basata su machine learning**, lasciando l'analisi manuale o basata su soglie.

1.2.3 Splunk

È una piattaforma software leader di mercato per la ricerca, il monitoraggio e l'analisi di dati macchina (machine-generated data) su larga scala, ampiamente utilizzata per la sicurezza (SIEM), l'osservabilità e le operation IT³. Essendo una soluzione enterprise, offre una suite di funzionalità estremamente ricca e matura, inclusi potenti strumenti di **machine learning integrati** per l'analisi predittiva e il rilevamento di anomalie, con un'interfaccia considerata tra le migliori del settore. Il suo limite, tuttavia, è insormontabile per il target di questo progetto: il **costo**. Il modello di licenza, basato sul volume di dati ingeriti giornalmente, è **proibitivo per la quasi totalità delle PMI**, ponendosi in diretto contrasto con l'obiettivo di questa tesi di fornire una soluzione 100% open source e dai costi di gestione contenuti.

1.2.4 Zabbix

È una piattaforma di monitoraggio di livello enterprise, open source e all-in-one, progettata per la supervisione completa di infrastrutture IT, dai server fisici alle applicazioni⁴. La sua forza risiede nella maturità e completezza, offrendo un monitoraggio estremamente granulare sia in modalità **agent-based** sia **agentless** (tramite protocolli come SNMP, ICMP, IPMI), supportato da una vasta community. Il principale limite di Zabbix, nel contesto di questo progetto, è il suo paradigma

²I siti ufficiali sono: <https://prometheus.io/> e <https://grafana.com/>

³Il sito ufficiale di Splunk è disponibile all'indirizzo: <https://www.splunk.com/>

⁴Il sito ufficiale di Zabbix è disponibile all'indirizzo: <https://www.zabbix.com/>

di alerting intrinsecamente **reattivo e basato su soglie (threshold-based)**. Non include un motore di machine learning nativo per l'anomaly detection, rendendolo cieco a problemi subdoli che non violano una soglia predefinita. Inoltre, la sua configurazione iniziale può risultare complessa e dispendiosa in termini di tempo, specialmente in ambienti eterogenei.

Tabella 1.1: Tabella Comparativa delle Soluzioni di Monitoraggio. I punti contrassegnati con (*) indicano funzionalità previste nella visione a lungo termine del progetto *EPS+*, non implementate nel presente lavoro di tesi.

Soluzione	Punti di Forza	Limitazioni nel Contesto della Tesi	Valore Aggiunto del Progetto EPS+
Zabbix	<ul style="list-style-type: none"> • Monitoraggio SNMP maturo • Architettura Agent/Agentless • Ampia community 	<ul style="list-style-type: none"> • Alerting basato su soglie statiche • Assenza di ML nativo per anomaly detection • Complessità di configurazione 	<ul style="list-style-type: none"> • Anomaly detection basata su ML • Health score unificato • UI moderna e intuitiva
ELK Stack	<ul style="list-style-type: none"> • Potente per analisi dei log • Performance di Elasticsearch • Visualizzazioni avanzate con Kibana 	<ul style="list-style-type: none"> • ML per anomalie solo con licenza a pagamento • Gestione SNMP non nativa • Intensivo in termini di risorse 	<ul style="list-style-type: none"> • Integrazione nativa delle metriche (SNMP) • Soluzione 100% open source • * Pre-configurato
Splunk	<ul style="list-style-type: none"> • Suite di funzionalità enterprise • ML e analisi predittiva integrati 	<ul style="list-style-type: none"> • Costo di licenza proibitivo per le PMI 	<ul style="list-style-type: none"> • Alternativa open source e a costo zero • *Deploy semplice tramite Docker
Prometheus & Grafana	<ul style="list-style-type: none"> • Ottimizzato per time-series • Ideale per ambienti container-native • UI moderna e flessibile con Grafana 	<ul style="list-style-type: none"> • Non gestisce i log • Modello di raccolta solo "pull" • Assenza di anomaly detection nativa 	<ul style="list-style-type: none"> • Analisi integrata di *log* e metriche • Anomaly detection built-in

1.3 Tecnologie e Strumenti di Riferimento

La realizzazione del progetto e l'analisi condotta in questo lavoro si basano su un insieme di tecnologie e standard consolidati nel mondo dell'IT e dello sviluppo software. Di seguito viene fornita una descrizione approfondita di ciascuno strumento, presentato in ordine alfabetico.

1.3.1 Acronis Cyber Protect

È una piattaforma integrata di "cyber protection" che unifica diverse discipline della sicurezza informatica. Supera il concetto di backup tradizionale integrando,

in un'unica soluzione, funzionalità di **backup e disaster recovery**, **protezione anti-malware** basata su intelligenza artificiale, **vulnerability assessment** e gestione della protezione degli endpoint. Questo approccio olistico mira a ridurre la complessità e a migliorare la postura di sicurezza, proteggendo i dati da un'ampia gamma di minacce, dai guasti hardware ai ransomware.⁵

1.3.2 Address Resolution Protocol (ARP)

È un protocollo di comunicazione essenziale che opera al confine tra il Livello 2 (Data Link) e il Livello 3 (Network) del modello OSI. Il suo scopo fondamentale è quello di risolvere, ovvero mappare, un indirizzo logico di Livello 3 (come un **indirizzo IP**) con il corrispondente indirizzo fisico di Livello 2 (**indirizzo MAC**) all'interno di una rete locale. Quando un host deve comunicare con un altro host sulla stessa sottorete, invia una richiesta ARP in broadcast chiedendo "Chi possiede questo indirizzo IP?". L'host di destinazione risponde con il proprio indirizzo MAC. Questa mappatura viene quindi memorizzata in una **ARP cache** locale per ottimizzare le comunicazioni future. Dal punto di vista della sicurezza, il monitoraggio del traffico ARP è cruciale per rilevare attacchi come l'**ARP spoofing**, una tecnica usata per attacchi Man-in-the-Middle (MitM).⁶

1.3.3 AJAX (Asynchronous JavaScript and XML)

È una tecnica di sviluppo per la creazione di applicazioni web asincrone. Utilizzando oggetti nativi del browser (come 'XMLHttpRequest' o la più moderna 'Fetch API'), permette a JavaScript di scambiare dati con un server in background, senza la necessità di ricaricare l'intera pagina. Questa tecnica è fondamentale per creare interfacce utente fluide e reattive, come una dashboard di monitoraggio che aggiorna i grafici in tempo reale.

1.3.4 Chart.js

È una libreria JavaScript open source, semplice ma flessibile, per la **visualizzazione di dati**. Permette agli sviluppatori di creare facilmente grafici interattivi e animati all'interno di un'applicazione web. Utilizzando l'elemento **HTML5 <canvas>**, offre una vasta gamma di tipi di grafici (a linee, a barre, a torta, radar, etc.), è completamente personalizzabile e **responsive**, ovvero in grado di adattarsi

⁵Sito ufficiale di Acronis: <https://www.acronis.com/it-it/products/cyber-protect/>

⁶Lo standard originale per ARP è definito dalla IETF nella RFC 826.

dinamicamente alle dimensioni dello schermo. La sua leggerezza e facilità d'uso la rendono una scelta ideale per la creazione di dashboard di monitoraggio.⁷

1.3.5 Flask

È un "micro-framework" per lo sviluppo di applicazioni web in Python. Viene definito "micro" perché non impone una struttura rigida o dipendenze esterne, fornendo solo gli strumenti essenziali come il routing delle URL e la gestione delle richieste/risposte. La sua leggerezza e flessibilità lo rendono ideale per la creazione di **API RESTful**, che fungono da backend per le moderne applicazioni front-end basate su JavaScript.⁸

1.3.6 HTML, CSS e JavaScript

Questo trio di tecnologie costituisce il fondamento del front-end di ogni applicazione web moderna, operando in sinergia per definire l'intera esperienza utente. Alla base si trova l'**HTML (HyperText Markup Language)**, il linguaggio di markup standard utilizzato per creare la **struttura** e il contenuto semantico di una pagina web⁹. A questo si sovrappone il **CSS (Cascading Style Sheets)**, il linguaggio che ne definisce lo **stile** e la presentazione, controllando layout, colori e l'aspetto visivo complessivo¹⁰. Infine, **JavaScript (JS)** è il linguaggio di programmazione che aggiunge **interattività** e comportamento dinamico; eseguito nel browser dell'utente, gestisce eventi, manipola il DOM (Document Object Model) e comunica con i server¹¹.

1.3.7 MongoDB

È un programma database **NoSQL**, **document-oriented** e **source-available** di primo piano. A differenza dei database relazionali tradizionali che utilizzano tavole con righe e colonne, memorizza i dati in documenti flessibili, simili a JSON, utilizzando un formato chiamato **BSON (Binary JSON)**. Questo modello basato su documenti permette uno **schema flessibile**, il che significa che la struttura dei dati può essere facilmente modificata nel tempo senza interruzioni di servizio. È progettato per la **scalabilità orizzontale**, che gli permette di gestire enormi quantità di dati distribuendoli su più server (una tecnica nota come sharding).

⁷Sito ufficiale di Chart.js: <https://www.chartjs.org/>

⁸Documentazione ufficiale di Flask: <https://flask.palletsprojects.com/>

⁹Standard HTML gestito dal WHATWG: <https://html.spec.whatwg.org/>

¹⁰Specifiche CSS mantenute dal W3C: <https://www.w3.org/Style/CSS/>

¹¹Standard ECMAScript, su cui si basa JavaScript: <https://www.ecma-international.org/>

Queste caratteristiche lo rendono una scelta popolare per applicazioni moderne, big data e sistemi real-time, dove l’agilità e la scalabilità sono requisiti critici.¹²

1.3.8 Nmap (Network Mapper)

È una potente utility open source per l’esplorazione di reti e l’auditing della sicurezza. Il suo scopo principale è la **scansione di host e servizi** su una rete, permettendo di identificare quali indirizzi IP sono attivi, quali porte sono aperte, quali servizi (e le loro versioni) sono in esecuzione e quale sistema operativo stanno utilizzando. È uno strumento diagnostico fondamentale per la mappatura e l’inventario di una rete.¹³

1.3.9 Proxmox Virtual Environment (PVE)

È una piattaforma di virtualizzazione server open source basata su Debian GNU/-Linux. La sua architettura integra due tecnologie di virtualizzazione: **KVM (Kernel-based Virtual Machine)** per le macchine virtuali, che offre una virtualizzazione completa via hardware, e **LXC (Linux Containers)** per i container, un approccio più leggero di virtualizzazione a livello di sistema operativo. Si distingue per la sua interfaccia di gestione web centralizzata, che permette di amministrare VM, container, storage e networking da un unico punto, rendendolo una soluzione molto popolare in ambienti PMI e datacenter.¹⁴

1.3.10 Python

È un linguaggio di programmazione interpretato, di alto livello e multi-paradigma, noto per la sua sintassi semplice e leggibile. È diventato il linguaggio dominante nel campo della data science e del machine learning grazie al suo vasto ecosistema di librerie specializzate. Le principali utilizzate in questo lavoro sono: **Pandas** per la manipolazione e l’analisi di dati tabellari; **Scikit-learn**, una libreria completa per il machine learning classico che fornisce implementazioni efficienti di algoritmi di classificazione, regressione, clustering e anomaly detection.¹⁵

¹²Sito ufficiale di MongoDB: <https://www.mongodb.com/>

¹³Sito ufficiale di Nmap: <https://nmap.org/>

¹⁴Documentazione ufficiale di Proxmox VE: <https://www.proxmox.com/en/proxmox-virtual-environment>

¹⁵Sito ufficiale di Python: <https://www.python.org/>

1.3.11 Simple Network Management Protocol (SNMP)

È un protocollo standard del livello applicativo, definito dall'IETF, per la gestione e il monitoraggio di dispositivi connessi a una rete IP. La sua architettura si basa su tre componenti chiave: il **Manager** (il sistema che monitora), l' **Agent** (il software in esecuzione sul dispositivo monitorato) e la **MIB (Management Information Base)**, una struttura dati gerarchica che definisce le variabili (metriche, stati, configurazioni) che l'agent può esporre. Tramite comandi come **GET**, **SET** e notifiche asincrone chiamate **TRAP**, permette di raccogliere dati e configurare dispositivi di rete in modo standardizzato.¹⁶

1.3.12 Socket.IO

È una libreria JavaScript che abilita una comunicazione **real-time**, **bidirezionale** e **basata su eventi** tra un client (tipicamente un browser web) e un server. Il suo scopo principale è superare i limiti del modello richiesta-risposta di HTTP, permettendo al server di **inviare dati ("push") al client** in modo proattivo, senza che il client debba richiederli esplicitamente. Astrae la complessità della comunicazione real-time: tenta di stabilire una connessione tramite **WebSockets** e, in caso di fallimento (dovuto a firewall o proxy), effettua automaticamente un "fallback" a tecniche alternative come il long-polling. Questa robustezza la rende una tecnologia chiave per applicazioni live come chat, notifiche e dashboard che si aggiornano istantaneamente.¹⁷

1.4 Obiettivi della Tesi

Sulla base del contesto e delle problematiche analizzate, il presente lavoro di tesi quindi si propone di raggiungere i seguenti obiettivi:

1. **Progettare un'architettura per la gestione unificata.** L'obiettivo è definire un'architettura software modulare per la raccolta e l'integrazione di dati provenienti da fonti eterogenee, quali i sistemi di virtualizzazione Proxmox, soluzioni di backup Acronis e dati raccolti tramite SNMP dai dispositivi di rete. Tale architettura deve prevedere una dashboard centralizzata non solo per la visualizzazione dei dati, ma anche per l'esecuzione di operazioni di gestione basilari, semplificando l'interazione dell'operatore IT con l'infrastruttura.

¹⁶La documentazione di riferimento per SNMP è mantenuta dall'IETF (Internet Engineering Task Force), ad esempio RFC 1157.

¹⁷Sito ufficiale di Socket.IO: <https://socket.io/>

2. **Implementare e validare un modulo di anomaly detection non supervisionato.** L'obiettivo è sviluppare un componente software per l' analisi dei dati SNMP raccolti dai dispositivi di rete. Si intende dimostrare e validare l'efficacia di un approccio non supervisionato per superare la sfida della mancanza di dati etichettati in ambienti reali. Questo modulo fornirà un rilevamento e, al contempo, fungerà da meccanismo per la raccolta di dati che potranno essere analizzati in futuro per testare nuovi modelli.
3. **Progettare e implementare un "health score" aggregato.** L'obiettivo è creare un prototipo di "health score" per fornire una valutazione sintetica e immediata dello stato di salute dell'infrastruttura. Tale indicatore si baserà sul monitoraggio di tre dimensioni fondamentali:
 - la **Performance** operativa dei sistemi;
 - l' **Availability** dei servizi critici;
 - la **Resilience**, misurata tramite le metriche di backup per valutare la capacità di recupero da un incidente;

1.5 Struttura del documento

Il presente documento è organizzato come segue. Nel **Capitolo 2**, viene presentato lo stato dell'arte sul monitoraggio delle infrastrutture IT e sulle tecniche di anomaly detection. Il **Capitolo 3** descrive i dataset utilizzati, inclusi SNMP-MIB e quello generato appositamente. Nel **Capitolo 4** sono illustrate le metodologie proposte per il preprocessing, il rilevamento di anomalie e il calcolo degli health score. Il **Capitolo 5** dettaglia il setup sperimentale e gli esperimenti condotti, mentre il **Capitolo 6** riporta i risultati ottenuti. Il **Capitolo 7** presenta l'architettura e il prototipo della dashboard EPS+. Infine, il **Capitolo 8** riassume le conclusioni e propone sviluppi futuri, seguito dalla bibliografia e dai ringraziamenti.

Capitolo 2

Stato dell'Arte

Il presente capitolo analizza lo stato dell'arte del monitoraggio IT e del rilevamento di anomalie. L'analisi partirà dai framework di riferimento per la gestione dei servizi, come **ITIL** e **NIST**, e dalle metriche operative chiave (**SLO**, **RTO**). Si approfondiranno poi le diverse metodologie per l'identificazione di anomalie, confrontando l'approccio statistico del **Controllo di Processo** con le tecniche di **Machine Learning** classico e **Deep Learning**. Infine, verrà contestualizzato l'uso del protocollo **SNMP** come fonte dati per la sicurezza. L'obiettivo è fornire il quadro teorico necessario a motivare le scelte progettuali di questa tesi.

2.1 Monitoraggio delle infrastrutture IT

Il monitoraggio di un'infrastruttura informatica è un processo ciclico e proattivo, essenziale per la gestione operativa e la sicurezza di qualsiasi organizzazione. Il National Institute of Standards and Technology (NIST), nella sua Special Publication 800-137, definisce il monitoraggio continuo come un processo volto a "mantenere una consapevolezza costante della sicurezza delle informazioni, delle vulnerabilità e delle minacce per supportare le decisioni di gestione del rischio" [22]. Sebbene il focus del NIST sia primariamente sulla sicurezza, il principio di "consapevolezza costante" basata su dati è il fondamento di ogni moderna strategia di monitoraggio, estendendosi alla supervisione delle performance, della disponibilità e della resilienza dei sistemi.

Per tradurre questo principio in pratiche operative strutturate, si fa riferimento a framework consolidati per la gestione dei servizi IT come **ITIL (Information Technology Infrastructure Library)** [5]. ITIL scomponete la gestione dell'infrastruttura in una serie di processi chiave che mirano a garantire e migliorare la qualità del servizio. Adottando questa prospettiva è possibile identificare le tre dimensioni fondamentali del monitoraggio, che costituiscono i pilastri di questo lavoro di tesi:

1. **La Gestione della Capacità e delle Performance (Capacity and Performance Management):** Questo processo si focalizza sul monitoraggio delle **risorse** (CPU, RAM, rete, disco) per garantire che i servizi operino in modo efficiente e che la capacità dell'infrastruttura sia adeguata a soddisfare la domanda.
2. **La Gestione della Disponibilità (Availability Management):** Questo processo è incentrato sul monitoraggio della **disponibilità** dei servizi, assicurando che siano operativi e accessibili agli utenti secondo quanto concordato.
3. **La Gestione della Continuità Operativa (IT Service Continuity Management):** Quest'area si occupa di garantire la **resilienza** del servizio, supervisionando gli elementi necessari alla ripartenza in caso di disastro (**Disaster Recovery**).

Per misurare e gestire gli obiettivi di questi processi, è necessario definire un insieme preciso di metriche. Per l'ambito delle performance e della disponibilità, pratiche come la **Site Reliability Engineering (SRE)** hanno reso popolare una gerarchia di indicatori [6]:

- **Service Level Indicator (SLI):** Una misurazione quantitativa di un aspetto del servizio (es. la latenza);
- **Service Level Objective (SLO):** Un obiettivo interno che l'organizzazione si pone per un SLI (es. 99.9% di disponibilità);
- **Service Level Agreement (SLA):** Un contratto formale con il cliente che definisce le garanzie sul livello di servizio;

Per la Gestione della Continuità Operativa, invece, le metriche chiave misurano la capacità di recupero da un disastro. Come definito dal NIST nella Special Publication 800-34, queste sono [27]:

- **Recovery Time Objective (RTO):** Il **tempo massimo** accettabile per ripristinare un servizio dopo un'interruzione;
- **Recovery Point Objective (RPO):** La **quantità massima di dati** che si è disposti a perdere, misurata in tempo;

Queste attività di monitoraggio non sono solo cruciali per la stabilità operativa, ma costituiscono anche un pilastro di qualsiasi strategia di cybersecurity. Il **NIST Cybersecurity Framework (CSF) 2.0** fornisce un modello per la gestione del rischio informatico articolato in sei funzioni principali: *Govern, Identify, Protect, Detect, Respond e Recover* [21].

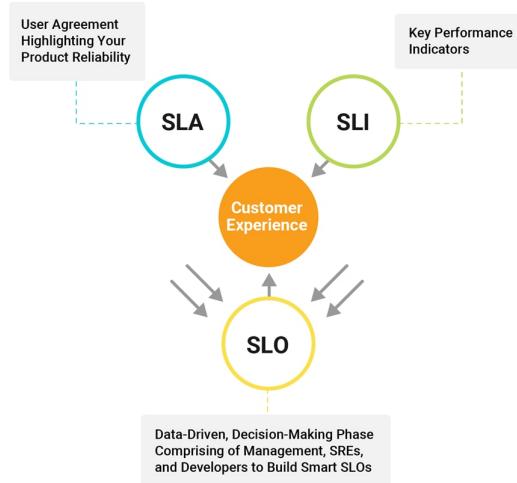


Figura 2.1: Rappresentazione della relazione gerarchica tra SLI (Service Level Indicator), SLO (Service Level Objective) e SLA (Service Level Agreement).



Figura 2.2: Il diagramma del NIST Cybersecurity Framework 2.0 che illustra le sei funzioni.

L'intero sforzo di monitoraggio descritto in questa tesi è progettato per contribuire attivamente o in parte agli obiettivi di: **Govern**, **Identify**, **Protect** e **Detect**. In sintesi, un monitoraggio moderno ed efficace deve integrare la supervisione di performance, disponibilità e capacità di disaster recovery, abilitando al contempo una funzione di rilevamento proattiva per la sicurezza. Implementare un sistema così completo rappresenta una sfida significativa, specialmente per le **Piccole e Medie Imprese (PMI)**. Emerge quindi un "gap" per una soluzione unificata, intelligente e accessibile, obiettivo che questa tesi si prefigge di indirizzare.

2.2 Tecniche di Anomaly Detection

Dopo aver stabilito la necessità di una funzione di "Detect" proattiva, questa sezione analizza lo stato dell'arte delle tecniche di Machine Learning per il rilevamento di anomalie (Anomaly Detection). L'obiettivo è fornire le basi teoriche per gli algoritmi che verranno poi implementati e testati nel corso di questo lavoro.

2.2.1 Definizioni e Tipologie di Anomalie

In letteratura, un'**anomalia** (o *outlier*) è definita come un'osservazione o un pattern di dati che si discosta in modo significativo dal comportamento atteso o considerato "normale". Come sottolineato da survey comprehensive del settore, la presenza di tali anomalie è spesso indicativa di eventi di grande interesse, come guasti di sistema, violazioni della sicurezza o frodi [15].

Le anomalie non sono tutte uguali. A seconda della loro natura, possono essere classificate in tre categorie principali:

- **Anomalie Puntuali (Point Anomalies):** Si tratta di una singola istanza di dati che risulta anomala rispetto al resto del dataset. Un esempio classico nel monitoraggio IT è un picco improvviso e isolato nell'utilizzo della CPU che supera di gran lunga i valori normali;
- **Anomalie Contestuali (Contextual Anomalies):** In questo caso, un'istanza di dati è considerata anomala solo all'interno di uno specifico contesto. Ad esempio, un elevato numero di accessi al server di backup alle 3 del pomeriggio potrebbe essere normale, ma lo stesso numero di accessi alle 3 di notte potrebbe essere un'anomalia sospetta;
- **Anomalie Collettive (Collective Anomalies):** Si verificano quando un insieme di osservazioni correlate risulta anomalo rispetto all'intero dataset, anche se le singole istanze, prese isolatamente, potrebbero non essere considerate anomalie. Un esempio è un attacco di tipo *Distributed Denial of Service (DDoS)* a basso volume, in cui migliaia di richieste singolarmente legittime, se osservate collettivamente, formano un pattern di attacco;

2.2.2 Paradigmi di Apprendimento per il Rilevamento

La capacità di un sistema di identificare le diverse tipologie di anomalie dipende fortemente dal paradigma di apprendimento impiegato. Nello stato dell'arte, si distinguono due approcci principali:

- **Apprendimento Supervisionato (Supervised Learning):** In questo paradigma, l'algoritmo viene addestrato su un dataset in cui ogni singola osservazione è stata preventivamente etichettata come "normale" o "anomala". Il

modello impara a riconoscere le caratteristiche che distinguono le due classi. Questo approccio può raggiungere performance molto elevate, ma richiede la disponibilità, spesso rara e costosa, di un dataset di addestramento completo ed etichettato. È un approccio valido in scenari specifici in cui le tipologie di anomalie sono ben note, come dimostrato in lavori che utilizzano i dati SNMP per classificare attacchi di rete noti [13];

- **Apprendimento Non Supervisionato (Unsupervised Learning):** Questo approccio rappresenta lo scenario più comune e sfidante. L'algoritmo viene addestrato su un dataset completamente privo di etichette, partendo dall'assunto che la maggior parte dei dati rappresenti il comportamento normale. Il compito del modello è quello di identificare autonomamente le osservazioni che si discostano maggiormente da tale normalità. Sebbene possa essere meno accurato su minacce specifiche, questo paradigma è l'unico applicabile quando non si conosce a priori la natura delle possibili anomalie, rendendolo ideale per il rilevamento di minacce nuove o impreviste ("zero-day"). L'algoritmo **Isolation Forest**, ad esempio, è un metodo basato su questo paradigma che, invece di profilare i dati normali, isola esplicitamente le anomalie [19];

2.2.3 Algoritmi di Riferimento nello Stato dell'Arte

Una volta definiti i paradigmi di apprendimento, è possibile analizzare specifici algoritmi che li implementano. Per questo studio sono stati selezionati tre modelli rappresentativi, scelti per la loro efficacia e per la loro diffusione in letteratura: Isolation Forest come esponente dell'approccio non supervisionato, e Random Forest e k-Nearest Neighbors come benchmark per l'approccio supervisionato.

La Frontiera del Deep Learning per l'Anomaly Detection

Negli ultimi anni, lo stato dell'arte nel rilevamento di anomalie, specialmente in contesti con dati complessi, non strutturati o sequenziali (come le serie temporali), è stato dominato da approcci basati sul **Deep Learning (DL)**. Come evidenziato da survey comprehensive del settore, modelli come gli **Autoencoder**, le **Reti Neurali Ricorrenti (RNN)** come LSTM e GRU, e più recentemente i **Transformer**, hanno dimostrato capacità superiori nel modellare la "normalità" in dati ad alta dimensionalità [15]. Questi modelli eccellono nell'apprendere automaticamente feature gerarchiche e complesse, superando spesso le performance degli algoritmi di Machine Learning tradizionali.

Tuttavia, questa elevata capacità di rappresentazione ha un costo significativo. I modelli di Deep Learning sono notoriamente "data-hungry", ovvero richiedono **enormi quantità di dati** per essere addestrati efficacemente, evitando l'overfitting. Inoltre, il loro addestramento comporta un **elevato costo computazionale**,

necessitando di hardware specializzato (GPU) e lunghi tempi di elaborazione. Infine, la loro natura di "scatole nere" (*black box*) rende spesso difficile interpretare il motivo per cui una specifica anomalia è stata rilevata, un aspetto critico in contesti operativi.

Per queste ragioni, sebbene il Deep Learning rappresenti la frontiera della ricerca, per contesti come quelli delle PMI — caratterizzati da una probabile scarsità di dati storici e da una limitata infrastruttura computazionale — gli approcci di Machine Learning classico, come quelli analizzati in seguito, rappresentano una soluzione più pragmatica, robusta e immediatamente applicabile.

Isolation Forest

L'algoritmo **Isolation Forest (IF)** rappresenta un approccio innovativo e particolarmente efficiente all'anomaly detection non supervisionata [19]. A differenza della maggior parte dei metodi che tentano di costruire un modello complesso della "normalità" per poi identificare ciò che non vi rientra, l'Isolation Forest si basa su un principio più semplice e diretto: **le anomalie sono più facili da isolare rispetto ai punti normali**. L'idea fondamentale è che le anomalie, essendo "poche e diverse", richiederanno in media un numero inferiore di partizioni casuali dello spazio delle feature per essere isolate in un albero decisionale. L'algoritmo costruisce una foresta di alberi casuali (iTrees) e l'anomaly score di un punto è calcolato in base alla lunghezza media del percorso necessario per isolarlo. Percorsi più brevi indicano una maggiore probabilità che il punto sia un'anomalia.

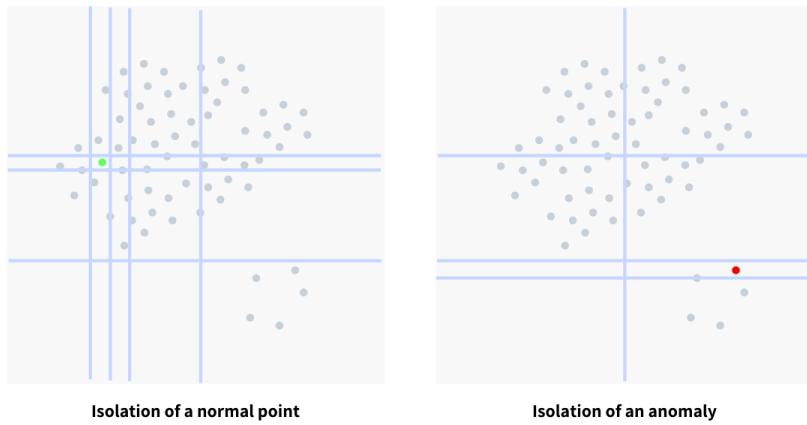


Figura 2.3: Illustrazione grafica di come l'Isolation Forest isola le anomalie

Random Forest e k-Nearest Neighbors come Benchmark Supervisionati

Quando si dispone di un dataset etichettato, è possibile utilizzare algoritmi di classificazione supervisionata, che spesso fungono da benchmark per valutare le

performance massime ottenibili. In questo contesto, due dei modelli più utilizzati in letteratura per la loro robustezza ed efficacia sono Random Forest e k-Nearest Neighbors.

- **Random Forest (RF)** è un metodo di *ensemble learning* che costruisce una moltitudine di alberi decisionali in fase di addestramento. La classificazione di una nuova istanza avviene tramite un "voto di maggioranza" tra tutti gli alberi. Questo approccio riduce il rischio di overfitting e aumenta la generalizzazione del modello, rendendolo molto efficace in compiti di classificazione complessi, come il rilevamento di intrusioni di rete [13];
- **k-Nearest Neighbors (k-NN)** è un algoritmo *instance-based* la cui logica si basa sul principio che osservazioni simili tendono ad appartenere alla stessa classe. Per classificare un nuovo punto, l'algoritmo individua i suoi 'k' vicini più prossimi nel dataset di addestramento e assegna la classe che risulta maggioritaria tra di essi. Nonostante la sua semplicità, k-NN è spesso utilizzato come una solida baseline in problemi di classificazione;

2.2.4 L'Approccio Statistico e la Soluzione ai Problemi del Monitoraggio

Un approccio consolidato, che si affianca alle tecniche di Machine Learning, affronta il rilevamento di anomalie da una prospettiva statistica. Il monitoraggio basato su soglie statiche e predefinite soffre di tre problemi fondamentali [7]:

1. **l'assenza di soglie universalmente valide**, data l'eterogeneità degli ambienti e dei processi da monitorare;
2. **la difficoltà nel definire oggettivamente un comportamento anomalo**, andando oltre la semplice violazione di un limite;
3. **la mancanza di adattabilità del monitoraggio**, che non tiene conto dei cambiamenti naturali nelle performance di un processo nel tempo;

La metodologia **Six Sigma**, attraverso il **Controllo Statistico di Processo (Statistical Process Control - SPC)**, offre una soluzione strutturata a queste problematiche. L'SPC non si basa su limiti arbitrari, ma sul comportamento osservato del processo stesso per determinare la sua variazione naturale e attesa.

Lo strumento operativo principale dell'SPC sono le **Carte di Controllo (Control Charts)**. Queste visualizzano l'andamento di una metrica nel tempo all'interno di un "corridoio di normalità", definito da una linea centrale (la media) e da limiti di controllo superiore (UCL) e inferiore (LCL), calcolati statisticamente dai dati (solitamente a $\pm 3\sigma$). Un processo è considerato stabile o "sotto controllo" se le sue osservazioni si disperdonano casualmente entro questi limiti.

Questo approccio risponde direttamente ai problemi identificati:

- **risolve il problema delle soglie**, poiché i limiti di controllo non sono predefiniti ma sono espressione del particolare fenomeno osservato e possono essere ricalcolati per adeguare la sensibilità del monitoraggio ai cambiamenti;
- **definisce cos'è un'anomalia** attraverso una serie di test di stabilità. Qualsiasi violazione di questi test indica un comportamento anomalo. Tali test includono non solo le **anomalie puntuali** (un'osservazione fuori dai limiti di controllo), ma anche le **anomalie collettive**, identificate tramite regole specifiche (note in letteratura come *Western Electric Rules*) che individuano pattern non casuali, come una sequenza di sette o più punti consecutivi tutti al di sopra della media, che indicano un cambiamento (*shift*) nel processo, con conseguente ricalcolo della "baseline";

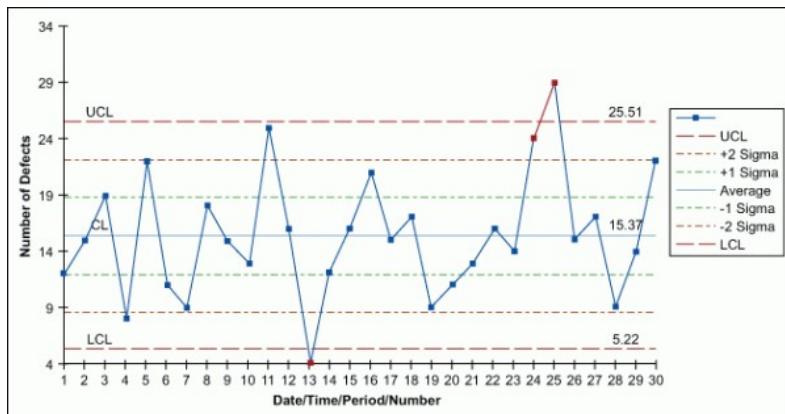


Figura 2.4: Esempio di Carta di Controllo (Control Chart) utilizzata nel Controllo Statistico di Processo.

Un processo è considerato "sotto controllo" se le sue misurazioni si disperdonano casualmente all'interno di limiti statistici. Un'anomalia viene identificata quando vengono violati specifici **test di stabilità**, che, come descritto nelle slide di riferimento [7], possono essere raggruppati in tre macro-categorie:

Sigma Tests Forniscono un primo avvertimento su possibili cause eccezionali. Sono progettati per rilevare le **anomalie puntuali** più evidenti. Il test principale di questa categoria identifica un singolo punto che cade al di fuori dei limiti di controllo superiore o inferiore (tipicamente $\pm 3\sigma$), segnalando un evento anomalo statisticamente improbabile.

Limit Tests Evidenziano un cambiamento avvenuto nelle prestazioni di un processo, suggerendo che i limiti di controllo in uso potrebbero non essere più rappresentativi. Rilevano **anomalie collettive** che indicano uno *shift* nella media del processo, come nel caso di nove punti consecutivi che si trovano tutti dallo stesso lato della linea centrale. Un fallimento di questi test esprime la necessità di ricalcolare i limiti.

Trend Tests Identificano fenomeni di cambiamento graduale, in corso o appena avvenuti. Sono utili per rilevare **anomalie collettive** che si manifestano come tendenze. Un esempio tipico è una sequenza di sei punti consecutivi in continua crescita o decrescita, che può essere il risultato di un degrado progressivo o, al contrario, di un'iniziativa di miglioramento non ancora stabilizzata.

Sebbene molto robusto per l'analisi di singole metriche con andamento prevedibile, questo metodo statistico mostra i suoi limiti in scenari ad alta dimensionalità. In tali contesti, le anomalie emergono non da deviazioni di una singola variabile, ma dalle complesse interazioni tra più metriche, un ambito in cui gli approcci di Machine Learning discussi in precedenza offrono una maggiore flessibilità.

2.2.5 La Sfida del Bilanciamento del Dataset e la Tecnica SMOTE

Un prerequisito fondamentale per l'addestramento di modelli di classificazione supervisionata efficaci è la disponibilità di un dataset di training bilanciato, in cui le classi da predire siano rappresentate in modo equo. Spesso, nei problemi di anomaly detection, si affronta il problema dello *sbilanciamento delle classi* (class imbalance), dove una classe (solitamente quella normale) è molto più numerosa dell'altra.

Tuttavia, a seconda della natura del dataset a disposizione, può verificarsi anche lo scenario inverso. Nel contesto di questo studio, per alcune fasi sperimentali, il numero di campioni rappresentanti il comportamento "normale" era limitato rispetto al volume di esempi di "anomalie" note. Addestrare un modello in queste condizioni può portare a un classificatore con scarse capacità di generalizzazione, che impara a riconoscere le anomalie ma non ha una comprensione sufficientemente robusta della normalità.

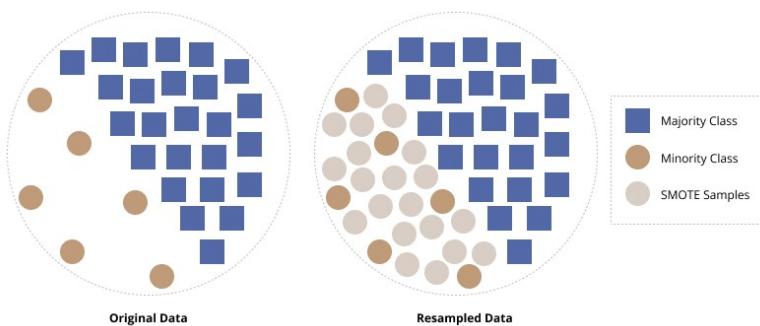


Figura 2.5: Rappresentazione grafica del funzionamento di SMOTE

Per risolvere questa sfida e creare un dataset di addestramento più equilibrato, è stata utilizzata la tecnica di ricampionamento **Synthetic Minority Over-sampling Technique (SMOTE)** [8]. Sebbene SMOTE sia classicamente applicata per aumentare la classe minoritaria, la sua logica è applicabile a qualsiasi classe si desideri popolare.

A differenza di semplici tecniche di *over-sampling* che duplicano casualmente i campioni, SMOTE opera in modo più intelligente: per ogni istanza della classe da aumentare (nel nostro caso, la classe "normale"), seleziona alcuni dei suoi vicini più prossimi e genera nuovi campioni **sintetici** lungo i segmenti di linea che congiungono l'istanza originale ai suoi vicini. Questo approccio permette di creare nuove informazioni plausibili, arricchendo il dataset e aiutando il classificatore a costruire un modello della normalità più completo e affidabile.

L'applicazione di tecniche come SMOTE è quindi un passaggio metodologico cruciale per garantire l'equilibrio del dataset, una condizione necessaria per l'addestramento di modelli di classificazione robusti.

Capitolo 3

Dataset

Inizialmente, il capitolo presenterà il dataset pubblico di benchmark, **SNMP-MIB**, dettagliandone le caratteristiche e il suo ruolo nello sviluppo del modulo di Machine Learning per il rilevamento di anomalie di rete. Verrà inoltre condotta un'analisi critica dei suoi limiti intrinseci, i quali motivano la necessità di un approccio complementare. Successivamente, il capitolo si concentrerà sulla progettazione, l'implementazione e la giustificazione metodologica di un **dataset sintetico su misura**. Questo secondo dataset è stato creato per permettere una validazione degli algoritmi di scoring specifici per le metriche di Performance, Availability e Resilience.

3.1 Dataset SNMP-MIB

Per gli scopi di questa tesi, nello specifico per l'implementazione del modulo di Anomaly Detection su dati SNMP (Simple Network Management Protocol), la ricerca è stata orientata verso un dataset che utilizzasse le variabili definite nello standard MIB-II. Come evidenziato in letteratura, l'analisi dei parametri MIB raccolti dalle interfacce di rete costituisce un approccio efficiente per il rilevamento di anomalie, in quanto sfrutta un flusso di dati già disponibile nella maggior parte dei dispositivi di rete, con un basso impatto sulle risorse hardware [13].

Nonostante la centralità di SNMP nella gestione delle reti, esiste una notevole carenza di dataset pubblici, realistici e correttamente etichettati, indispensabili per addestrare e testare in modo affidabile i sistemi di *anomaly detection*. Questa lacuna rende difficile per ricercatori e professionisti sviluppare e confrontare nuove tecniche basate su dati che riflettano le condizioni operative e le minacce informatiche contemporanee.

Tra i pochi disponibili è stato selezionato il dataset sviluppato da Al-Kasassbeh, Al-Naymat e Al-Hawari [2]. Tale dataset è stato appositamente creato per colmare la suddetta lacuna, fornendo una raccolta di dati SNMP MIB-II generati in un

ambiente di rete fisico e realistico, anziché simulato. Il suo valore fondamentale risiede nella capacità di rappresentare sia il comportamento di una rete in condizioni operative normali sia le alterazioni indotte da una varietà di attacchi informatici comuni, come diverse forme di DoS e Brute Force, fornendo etichette precise per ogni istanza registrata.

3.1.1 Metodologia di Raccolta e Generazione

Il processo di creazione del dataset, descritto in dettaglio in [2], può essere suddiviso nella configurazione di un’infrastruttura fisica di test (test-bed) e nella successiva fase di generazione del traffico e raccolta dei dati.

Configurazione del Test-Bed

Per garantire il pieno controllo sull’esperimento, gli autori hanno allestito un’infrastruttura di rete fisica e isolata presso i laboratori dell’Università di Mu’tah. L’architettura emulava una piccola rete aziendale ed era composta da hardware reale. La topologia, schematizzata in Figura 3.1, includeva i seguenti componenti chiave:

- **un PC-Router:** Un computer standard con sistema operativo Windows 7 è stato configurato per agire come router centrale tra le due sottoreti;
- **due switch Cisco Catalyst 2950:** utilizzati per la gestione del traffico all’interno delle reti locali;
- **due sottoreti fisiche:** una contenente l’host designato come **attaccante** e l’altra contenente il server designato come **vittima** e altri host per la generazione di traffico legittimo;

Questa configurazione ha permesso di isolare l’ambiente, monitorare con precisione il traffico e lanciare attacchi controllati senza impattare reti di produzione [2].

Generazione del Traffico e Raccolta Dati

Il processo di popolamento del dataset si è svolto in due attività parallele: la generazione di traffico di rete e il polling dei contatori SNMP del router.

Per simulare un utilizzo realistico della rete, è stato impiegato lo strumento **LanTrafficV2** per generare un flusso di traffico di background legittimo (TCP e UDP). Contemporaneamente, un host designato come attaccante ha sferrato una serie di attacchi mirati verso il server vittima. Gli attacchi, eterogenei per tipologia, includevano [2]:

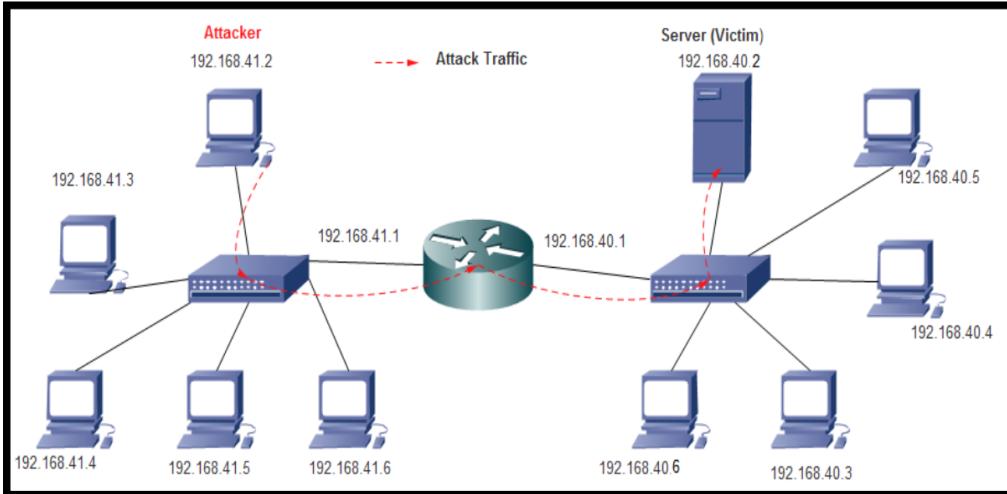


Figura 3.1: Schema della topologia di rete fisica utilizzata per la generazione del dataset [2]

- **denial of service (DoS) flooding:** nelle varianti TCP-SYN, UDP Flood e ICMP-ECHO, generate con lo strumento **HyenaeFE**;
- **attacchi a livello applicativo:** HTTP Flood (con **DOSHTTP**), Slowloris (tramite uno script Perl) e Slowpost (con **HttpDosTool4.0**);
- **brute force attack:** un attacco a dizionario contro un servizio web Apache, eseguito tramite **THC-Hydra**;

Mentre la rete era operativa, un **programma Java custom**, sviluppato dagli autori utilizzando il toolkit webNMS, è stato configurato per interrogare l'agente SNMP in esecuzione sul router. Il polling veniva eseguito a un intervallo regolare di **15 secondi**. I dati raccolti corrispondevano a **34 diverse variabili MIB-II**, appartenenti a cinque gruppi (Interface, IP, ICMP, TCP e UDP). Ogni interrogazione ha prodotto una riga del dataset, successivamente etichettata come **Normal** o con la specifica tipologia di attacco in corso in quel lasso di tempo [2].

Limiti del Dataset

Nonostante il suo valore e l'accurata metodologia di generazione, è fondamentale riconoscere alcuni limiti intrinseci del dataset utilizzato, che derivano dalla natura specifica dell'ambiente sperimentale in cui è stato creato [2]. La consapevolezza di tali limiti è cruciale per contestualizzare i risultati ottenuti e per delineare i futuri sviluppi.

- **scala e complessità della rete:** L'infrastruttura di test, sebbene realistica, è di dimensioni molto contenute (un router, due switch e una decina di host)

[2]. Le reti delle PMI reali, pur essendo piccole, possono presentare una complessità maggiore, con più dispositivi, segmenti di rete (VLAN), firewall e policy di sicurezza. Un modello addestrato su questo dataset potrebbe quindi avere difficoltà a generalizzare le sue performance in ambienti più grandi e "rumorosi";

- **scarsa varietà del traffico legittimo:** Il traffico di background normale è stato generato interamente dallo strumento LanTrafficV2 [2]. Sebbene efficace per creare un carico di rete controllato, questo traffico sintetico potrebbe non catturare la piena eterogeneità e imprevedibilità del comportamento umano e delle applicazioni moderne (es. streaming video, VoIP, traffico da servizi cloud). Il modello potrebbe quindi apprendere una definizione di "normalità" troppo ristretta, rischiando di generare falsi positivi di fronte a traffico legittimo ma anomalo rispetto a quello visto in fase di addestramento;

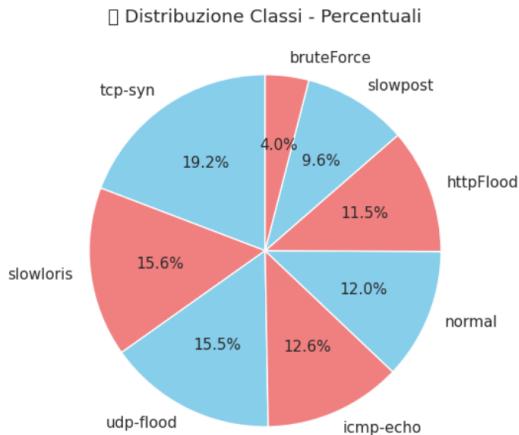


Figura 3.2: Distribuzione percentuale delle classi nel dataset SNMP-MIB. Si evidenzia come il traffico normale rappresenti solo il 12% del totale, a fronte di sette diverse classi di attacco.

- **focus su attacchi di tipo DoS:** Il dataset è fortemente orientato al rilevamento di attacchi Denial of Service (DoS) e Brute Force, che sono caratterizzati da un'alterazione evidente del volume e della tipologia del traffico [2]. Mancano tuttavia esempi di altre categorie di minacce altrettanto critiche per le PMI, come attacchi volti all'esfiltrazione di dati, infezioni da malware (es. ransomware) o attività di scansione e ricognizione. Un modello addestrato su questi dati risulterà quindi specializzato nel rilevare anomalie "rumorose", ma potrebbe essere cieco a minacce più silenziose e mirate;
- **granularità del campionamento:** L'intervallo di polling di **15 secondi**, pur essendo un buon compromesso tra dettaglio e carico, potrebbe non essere

sufficientemente fine per rilevare attacchi estremamente rapidi e di breve durata (burst attacks), che potrebbero iniziare e terminare tra due campionamenti consecutivi, risultando invisibili al sistema di monitoraggio;

3.2 Dataset Generato

Mentre il dataset SNMP-MIB si è rivelato fondamentale per lo sviluppo e il test del modulo di *Anomaly Detection* basato su Machine Learning, la validazione degli altri moduli algoritmici del sistema — specificamente quelli per l’analisi e il calcolo degli Score di Performance (SPC), Availability e Resilience — richiedeva un approccio differente. L’assenza di dataset pubblici adatti a questo scopo e l’impossibilità di allestire un ambiente di raccolta dati reale hanno reso necessaria la creazione di un dataset sintetico su misura, memorizzato e gestito tramite un database **MongoDB**.

Tale dataset è stato progettato per emulare fedelmente l’output di un modulo software di polling, che nella sua controparte reale sarebbe incaricato di interrogare tre fonti dati primarie per alimentare i rispettivi moduli di analisi:

1. **metriche di performance**: Il dataset simula la raccolta periodica di dati sull’utilizzo di CPU, RAM e I/O Wait che verrebbero ottenuti tramite chiamate alle API di Proxmox o interrogazioni SNMP (previa installazione dell’agent SNMP adatto all’SO di riferimento);
2. **dati di resilienza**: Simula i dati sullo stato dei backup (es. successo, fallimento, timestamp) che sarebbero collezionati interrogando le API di Acronis;
3. **dati di disponibilità**: Simula l’esito (UP/DOWN) di un modulo software di controllo che verificherebbe l’effettiva operatività dei servizi tramite controlli a livello applicativo (es. check su porte TCP, query HTTP, query SQL);

3.2.1 Motivazioni e Giustificazione Metodologica

La decisione di sviluppare un generatore di dati sintetici si fonda su pratiche consolidate nell’ingegneria del software e nella ricerca scientifica. L’approccio non è una semplice convenienza, ma una necessità metodologica per garantire la validità, la robustezza e la ripetibilità dei test condotti sui moduli algoritmici del sistema. L’intero processo, infatti, può essere inquadrato nella disciplina della **simulazione**, definita da Law come “**l’imitazione del funzionamento di un processo o sistema del mondo reale nel tempo**” [18]. Generando una “storia artificiale” del comportamento dell’infrastruttura, siamo in grado di trarre conclusioni sulle caratteristiche operative del nostro sistema di monitoraggio.

La validità del motore di analisi delle performance, basato sul Controllo Statisitico di Processo (SPC), dipende dalla capacità di testarlo in condizioni controllate.

A questo proposito, Soltana et al. sostengono che il testing statistico richiede la generazione di dati sintetici che posseggano due caratteristiche fondamentali: devono avere le **stesse caratteristiche statistiche** dei dati reali e devono poter soddisfare (o violare deliberatamente) specifici **vincoli logici** [24]. Questo principio si traduce direttamente nella nostra metodologia di generazione a due fasi: la **Fase 1 (Baseline)** crea dati con le "caratteristiche statistiche" controllate necessarie per validare il calcolo della normalità, mentre la **Fase 2 (Test)** introduce dati che violano i "vincoli logici" delle regole SPC per verificare che il sistema di rilevamento funzioni come atteso.

In modo analogo, le metriche di Disponibilità e Resilienza sono significative solo se misurate in relazione a eventi di fallimento. Attendere disservizi reali in un ambiente di produzione è impraticabile. Questa sfida viene superata attraverso la simulazione di tali eventi, una pratica standard nell'industria come documentato nel manuale di Site Reliability Engineering (SRE) di Google. Gli autori descrivono esplicitamente come, in assenza di fallimenti, venga **sintetizzata un'interruzione controllata** (*synthesized controlled outage*) per testare proattivamente la robustezza del sistema [6]. Il nostro approccio adotta lo stesso principio: generando sequenze di stati UP/DOWN e di esiti di backup, stiamo di fatto "sintetizzando" una cronologia di eventi avversi per validare rigorosamente la logica di calcolo degli SLO e degli score di resilienza.

Questo concetto si estende alla valutazione di proprietà sistemiche complesse come la resilienza. In ambito accademico, è stato dimostrato che la **simulazione** è una componente metodologica chiave in un framework completo per la valutazione della resilienza dei sistemi [26]. L'uso di dati generati per simulare il comportamento del sistema di backup nel tempo rappresenta quindi un approccio metodologicamente valido e scientificamente fondato per poter derivare una metrica quantitativa come il *Resilience Score*.

3.2.2 Modellazione dei Dati e Corrispondenza con API Reali

Per garantire che il dataset generato sia una rappresentazione fedele dei dati reali, la sua struttura è stata modellata direttamente sulle risposte fornite dalle API ufficiali di Proxmox VE e Acronis Cyber Protect Cloud.

Metriche di Performance da Proxmox VE

Le metriche di performance per host fisici, macchine virtuali e container vengono raccolte dall'API RESTful di Proxmox, la quale espone dati storici memorizzati in database RRD (Round-Robin Database).

Chiamata API Reale: La chiamata API principale per ottenere questi dati è:

GET /api2/json/nodes/{node}/rrddata

Questa chiamata, con il parametro `timeframe` opportunamente impostato, restituisce una serie storica di oggetti JSON.

Mapping dei Dati e Struttura su MongoDB: Dalla risposta dell'API, vengono estratti i campi chiave ('cpu', 'iowait', 'memused', 'memtotal') e mappati nella struttura del documento memorizzato in MongoDB. La percentuale di RAM utilizzata viene calcolata come `(memused / memtotal) * 100`. Questo processo porta a un documento finale che rispecchia fedelmente i dati ottenibili, come mostrato nel Listato 3.1.

```
{  
    "timestamp": "2025-01-15T10:30:00Z",  
    "machine_id": "WebServer-01",  
    "metrics": {  
        "cpu_percent": 28.5,  
        "ram_percent": 42.1,  
        "io_wait_percent": 4.8  
    },  
    "phase": "test",  
    "injected_anomaly_type": "Test 1 - Violazione Limite 3-Sigma"  
}
```

Listing 3.1: Struttura di un documento per le metriche di performance, modellata sulla risposta dell'API Proxmox.

Metriche di Resilienza da Acronis Cyber Protect Cloud

Le informazioni necessarie per calcolare la resilienza, come l'esito e il timestamp dei backup, vengono recuperate tramite l'API di Acronis.

Chiamate API Reali: Le due chiamate principali utilizzate per questo scopo sono:

- GET `/api/resource_management/v4/resources`: Per ottenere lo stato aggregato di protezione di un asset, inclusa l'informazione cruciale dell'ultimo backup completato con successo.
- GET `/api/task_manager/v2/tasks`: Per recuperare la cronologia dettagliata di tutte le attività eseguite, inclusi i job di backup, e analizzarne l'esito.

Mapping dei Dati e Struttura su MongoDB: Dai dati restituiti da queste API, vengono estratti i campi `state` (es. "succeeded", "failed") e `progress.finished_at` o `last_success_run` (il timestamp). Queste informazioni vengono poi trasformate e memorizzate in un documento strutturato su MongoDB, come mostrato nel Listato 3.2.

```
{  
    "timestamp": "2025-01-15T02:00:00Z",  
    "meta": {  
        "asset_id": "vm_001",  
        "metric_name": "backup_status"  
    },  
    "value": {  
        "backup_completed": 1.0,  
        "backup_size_gb": 45.2,  
        "duration_minutes": 23,  
        "backup_type": "incremental"  
    }  
}
```

Listing 3.2: Struttura di un documento per le metriche di backup, basata sulle risposte delle API Acronis

Metriche di Disponibilità e Controlli Applicativi

Infine, per alimentare il modulo di analisi della disponibilità, il generatore simula l'esito di un sistema di polling che monitora lo stato operativo dei servizi. Questo tipo di controllo, nella sua controparte reale, non si limita a un semplice ping, ma esegue verifiche a livello applicativo per garantire che il servizio stia effettivamente funzionando (es. una query HTTP a un web server, una connessione a un database).

Mapping dei Dati e Struttura su MongoDB: L'esito di questi controlli viene tradotto in un semplice documento time-series. Valore numerico ('1.0' per UP, '0.0' per DOWN). La struttura memorizzata è mostrata nel Listato 3.3.

```
{  
    "timestamp": "2025-01-15T10:30:00Z",  
    "meta": {  
        "asset_id": "service_001",  
        "metric_name": "availability_status"  
    },  
    "value": 1.0  
}
```

Listing 3.3: Struttura di un documento per le metriche di disponibilità, basata sull'esito di controlli applicativi.

3.2.3 Metodologia di Generazione e Implementazione

Per tradurre in pratica i principi di simulazione descritti, è stato sviluppato uno script di seeding ('seed_database.py') in Python, che si interfaccia con il database

MongoDB per popolarlo con un dataset completo e realistico. Questa sezione illustra gli estratti di codice fondamentali che implementano la logica di generazione per ciascuna delle tre principali categorie di metriche.

Implementazione della Generazione di Dati di Performance (SPC)

Il cuore del generatore di dati di performance è il modulo ‘generate_metrics_spc.py‘, la cui esecuzione è orchestrata dalla funzione ‘_generate_performance_data‘ in ‘seed_database.py‘. Come illustrato nel Listato 3.4, il codice implementa la logica a due fasi discussa in precedenza: una fase iniziale di *Baseline* per la raccolta di dati stabili e una successiva fase di *Test* in cui vengono iniettate programmaticamente diverse tipologie di anomalie.

```
# Ciclo principale di generazione
for machine_config in PROFILI_MACCHINE:
    machine_id = machine_config["machine_id"]
    history = {"cpu": [], "ram": [], "io_wait": []}

    for i in range(TOTAL_DAYS * SAMPLES_PER_DAY):
        current_time = START_DATE + timedelta(minutes=i * (1440 // SAMPLES_PER_DAY))

        # Determina la fase (baseline o test)
        if (current_time - START_DATE).days < BASELINE_DAYS:
            phase = "baseline"
            anomaly_type = None
            new_metrics = generate_normal_data(machine_config["profile"])
        else:
            phase = "test"
            anomaly_type, new_metrics = inject_anomaly_if_needed(
                machine_config["profile"], history
            )

        # Creazione del documento e inserimento in MongoDB
        metric_document = {
            "timestamp": current_time,
            "machine_id": machine_id,
            "metrics": new_metrics,
            "phase": phase,
            "injected_anomaly_type": anomaly_type
        }
        collection.insert_one(metric_document)
```

Listing 3.4: Estratto dal ciclo di generazione dati in ‘generate_metrics_spc.py‘.

Logica di Inserimento: La funzione `inject_anomaly_if_needed` contiene la logica per generare dati che violano le regole SPC, mentre la chiamata finale

`collection.insert_one()` si occupa della persistenza del documento generato nel database MongoDB.

Implementazione della Generazione di Dati di Disponibilità

La simulazione della disponibilità dei servizi è implementata dalla funzione ‘`_generate_availability_data`’ all’interno dello script ‘`seed_database.py`’. Per rendere la simulazione realistica, non si utilizza una semplice probabilità casuale, ma si modella un comportamento più verosimile in cui un servizio, una volta in *downtime*, ha una maggiore probabilità di rimanere in tale stato, simulando così un tempo di ripristino. L’estratto di codice nel Listato 3.5 mostra questa logica.

```
def _generate_availability_data(storage_manager, services,
    start_date, days):
    # ...
    for service in services:
        current_status = 1.0 # Inizia come UP
        failure_prob = 0.02

        for i in range(days * 1440): # Un campione al minuto
            timestamp = start_date + timedelta(minutes=i)

            # Un servizio DOWN ha il 50% di possibilità di
            # rimanere DOWN
            if current_status == 0.0 and random.random() < 0.5:
                pass # Rimane DOWN
            elif random.random() < failure_prob:
                current_status = 0.0 # Va DOWN
            else:
                current_status = 1.0 # E' UP

            # Creazione del documento tramite il modello e
            # inserimento
            metric = MetricDocument(
                timestamp=timestamp,
                meta={"asset_id": service["_id"], "metric_name": "availability_status"},
                value=current_status
            )
            storage_manager.insert_metric(metric.to_dict())
```

Listing 3.5: Estratto dalla funzione ‘`_generate_availability_data`’ in ‘`seed_database.py`’.

Implementazione della Generazione di Dati di Resilienza

Infine, la generazione dei dati di backup, gestita dalla funzione ‘`_generate_resilience_data`’ in ‘`seed_database.py`’, simula l’esecuzione schedulata dei job (in questo caso, uno al giorno alle 2 del mattino). Come mostrato nel Listato

3.6, l'esito di ogni backup (successo o fallimento) viene determinato in base a una probabilità predefinita ('success_rate'), e i metadati rilevanti vengono registrati.

```
def _generate_resilience_data(storage_manager, assets_to_backup,
    start_date, days):
    success_rate = 0.95

    for asset in assets_to_backup:
        for day in range(days):
            backup_time = (start_date + timedelta(days=day)).
replace(hour=2, minute=0)

            # Determina l'esito del backup
            if random.random() < success_rate:
                backup_completed = 1.0
                duration = random.randint(15, 60)
                size = round(random.uniform(20.0, 100.0), 2)
            else:
                backup_completed = 0.0
                duration = random.randint(5, 15) # Fallimento
                rapido
                size = 0.0

            # Creazione del documento e inserimento tramite Storage
            Manager
            metric = MetricDocument(
                timestamp=backup_time,
                meta={"asset_id": asset["_id"], "metric_name": "backup_status"},
                value={
                    "backup_completed": backup_completed,
                    "backup_size_gb": size,
                    "duration_minutes": duration,
                }
            )
            storage_manager.insert_metric(metric.to_dict())
```

Listing 3.6: Estratto dalla funzione '_generate_resilience_data' in 'seed_database.py'.

Capitolo 4

Proposizioni di Metodi e Tecniche

Dopo aver definito il contesto, lo stato dell’arte e i dataset di riferimento nei capitoli precedenti, questo capitolo illustra in dettaglio il framework metodologico adottato proposto per raggiungere gli obiettivi di ricerca. La trattazione si concentra in una prima fase sull’analisi del dataset pubblico **SNMP-MIB**, descrivendo l’insieme di tecniche di *preprocessing*, gestione dei dati e modellazione impiegate per sviluppare e validare il modulo di *Anomaly Detection*. L’approccio sperimentale qui descritto mira a replicare e confrontare le performance di algoritmi di *machine learning* supervisionati e non supervisionati, per poi estendere l’analisi a modelli di *deep learning* capaci di interpretare la natura sequenziale dei dati di rete. Successivamente, verranno introdotte le tecniche di simulazione utilizzate per il **dataset generato**, spiegando la logica implementativa per il calcolo istantaneo delle metriche di *performance*, *resilience* e *availability*, fondamentali per la costruzione dell’*health score* aggregato.

4.1 Metodologie per Dataset SNMP-MIB

Per il rilevamento di anomalie basato sui parametri SNMP (*Simple Network Management Protocol*), la scelta dei metodi è stata guidata dalla necessità di replicare e validare i risultati di studi scientifici esistenti, che hanno dimostrato l’efficacia di specifici classificatori su un sottoinsieme ridotto di variabili SNMP-MIB [13].

4.1.1 Tecniche di Preprocessing dei Dati

Il preprocessing dei dati è un passaggio critico, data l’estrema variabilità nelle scale dei valori presenti nel dataset SNMP-MIB, dove alcune variabili raggiungono valori nell’ordine dei milioni (es. `ifInOctets`) mentre altre rimangono nell’ordine delle unità (es. `tcpOutRsts`) [2]. Per affrontare questa sfida, sono state implementate le seguenti tecniche.

Normalizzazione e Standardizzazione

Le tecniche di scalatura delle feature sono un passaggio standard nel preprocessing per il machine learning, essenziale per garantire che gli algoritmi che si basano su misure di distanza o sull'ottimizzazione del gradiente non siano influenzati in modo sproporzionato dalla scala delle variabili di input [14].

L'applicazione dello **StandardScaler** (o Z-score normalization) è stata motivata dai requisiti di algoritmi basati su distanza come il KNN. Questa tecnica trasforma i dati in modo che abbiano una media (μ) pari a 0 e una deviazione standard (σ) pari a 1. La formula per la standardizzazione di un valore x è la seguente:

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

Senza questa trasformazione, le variabili con valori numerici più grandi dominerebbero il calcolo della distanza euclidea, mascherando il contributo delle variabili con scale minori. La necessità di gestire la scala delle variabili per algoritmi come il KNN è infatti un punto fondamentale discusso in letteratura [14].

Per il modello di *deep learning*, invece, è stata adottata la **normalizzazione Min-Max**, che scala i dati in un intervallo predefinito, tipicamente $[0, 1]$, secondo la formula:

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

Questa scelta previene problemi di gradiente (*vanishing/exploding*) nelle reti neurali.

Creazione di Sequenze con Finestra Scorrevole

Per sfruttare la natura temporale dei dati con il modello GRU-Autoencoder, le osservazioni indipendenti sono state trasformate in sequenze. Questo è un approccio comune nel *time series data mining*, dove una tecnica di **finestra scorrevole** (*sliding window*) viene utilizzata per convertire una serie temporale in un formato adatto a problemi di apprendimento supervisionato [17]. Nel nostro caso, è stata implementata una finestra con sovrapposizione. Una sequenza è stata etichettata come anomala se conteneva almeno un'osservazione anomala, seguendo un approccio conservativo adatto ai contesti di sicurezza.

4.1.2 Gestione dello Sbilanciamento delle Classi

Il dataset SNMP-MIB, come già accennato in precedenza, presenta una distribuzione atipica in cui la classe "normale" costituisce la minoranza (12% del totale), contrariamente alla maggior parte dei problemi di *anomaly detection*. Questa inversione ha richiesto un approccio sperimentale comparativo per identificare la

strategia di gestione ottimale, poiché, come evidenziato in letteratura, la gestione dello sbilanciamento delle classi è una delle sfide principali del settore [15].

Sottocampionamento Strategico e Stratificato

È stato implementato per ricreare una distribuzione più realistica in cui le anomalie rappresentano il 10% del dataset totale. A differenza di un sottocampionamento casuale, quello **stratificato** ha preservato le proporzioni relative di ogni tipo di attacco, garantendo che il modello potesse apprendere la loro completa varietà. Questa tecnica si è rivelata cruciale per rispettare l'assunzione fondamentale dell'algoritmo Isolation Forest, secondo cui le anomalie devono essere "poche e diverse" [19].

Sovracampionamento con SMOTE

Come approccio opposto, è stata utilizzata la tecnica **SMOTE** (*Synthetic Minority Over-sampling Technique*) [8]. Invece di limitarsi a duplicare i dati esistenti, SMOTE genera nuovi campioni sintetici interpolando tra vicini esistenti nello spazio delle feature. Nello specifico, la tecnica è stata applicata esclusivamente alla classe di minoranza ("normale") con l'obiettivo di aumentarne la numerosità in modo controllato. Lo scopo era quello di creare un dataset finale in cui la classe maggioritaria originale ("Anomalia") costituisse una percentuale predefinita del 10% del totale. Questo ha permesso di creare una rappresentazione più densa e continua del comportamento normale della rete, massimizzando l'informazione disponibile per l'addestramento senza scartare dati preziosi sugli attacchi.

4.1.3 Modelli di Rilevamento a Confronto

Per valutare l'efficacia dei diversi paradigmi di apprendimento, sono stati selezionati e implementati quattro modelli distinti: due supervisionati, uno non supervisionato e uno basato sul deep learning per l'analisi di serie temporali.

K-Nearest Neighbors (KNN)

Come primo modello supervisionato è stato scelto il KNN, un classificatore basato su istanza. La sua selezione è motivata dall'eccezionale accuratezza che questo algoritmo ha dimostrato in studi precedenti su questo stesso dataset [13]. La sua natura di *lazy learner*, che non costruisce un modello esplicito ma classifica le nuove istanze basandosi sulla vicinanza con i campioni di training, lo rende particolarmente efficace nel rilevare le deviazioni locali tipiche degli attacchi di rete.

Random Forest

È stato selezionato come secondo modello supervisionato per la sua robustezza e le sue caratteristiche complementari al KNN. Il Random Forest è un metodo di *ensemble* che costruisce una moltitudine di alberi decisionali e ne aggrega le predizioni. Questo lo rende intrinsecamente più robusto al rumore e meno incline all’overfitting, oltre che efficace nel gestire un alto numero di attributi, come dimostrato nello studio di riferimento [13].

Isolation Forest

Per il paradigma non supervisionato, è stato scelto l’Isolation Forest. A differenza degli approcci tradizionali, questo algoritmo non modella il comportamento "normale", ma isola esplicitamente le anomalie sfruttando la loro proprietà intrinseca di essere "poche e diverse" [19]. La sua efficienza, scalabilità e la capacità di operare senza etichette preesistenti lo rendono un candidato ideale per il rilevamento di minacce non note (*zero-day*).

Inoltre, la sua natura non supervisionata e la bassa complessità computazionale lo rendono perfetto per scenari operativi in cui si necessita di una soluzione **rapidamente implementabile**. Non richiedendo una fase di etichettatura dei dati, che è spesso lunga e costosa, può essere messo in funzione immediatamente per fornire un primo livello di monitoraggio. La sua efficacia anche con dataset di dimensioni contenute lo qualifica come uno strumento strategico per ottenere risultati validi fin da subito, anche in assenza di grandi quantità di dati storici.

GRU-Autoencoder

Infine, per sfruttare la natura temporale dei dati, è stato implementato un modello di *deep learning* basato su un’architettura **Autoencoder** con strati GRU (*Gated Recurrent Unit*), come mostrato in Figura 4.1. Come evidenziato in recenti rassegne, i modelli basati sulla ricostruzione con architetture ricorrenti rappresentano lo stato dell’arte per l’anomaly detection non supervisionata su serie temporali [15].

La scelta delle GRU rispetto alle più complesse LSTM è motivata dalla loro maggiore efficienza computazionale, avendo meno parametri da addestrare. Su un dataset di dimensioni contenute come quello in esame, la minore complessità delle GRU riduce anche il rischio di *overfitting*, raggiungendo performance spesso equivalenti a quelle delle LSTM su sequenze di lunghezza moderata [15].

L’architettura implementata segue il paradigma encoder-decoder:

- **Encoder:** Due layer GRU (con 64 e 32 unità) che comprimono progressivamente la sequenza temporale in una rappresentazione latente;
- **Decoder:** Una struttura speculare all’encoder che ricostruisce la sequenza originale a partire dal vettore latente;

- **Output:** Un layer *TimeDistributed(Dense)* che genera una predizione per ogni timestep della sequenza in output;

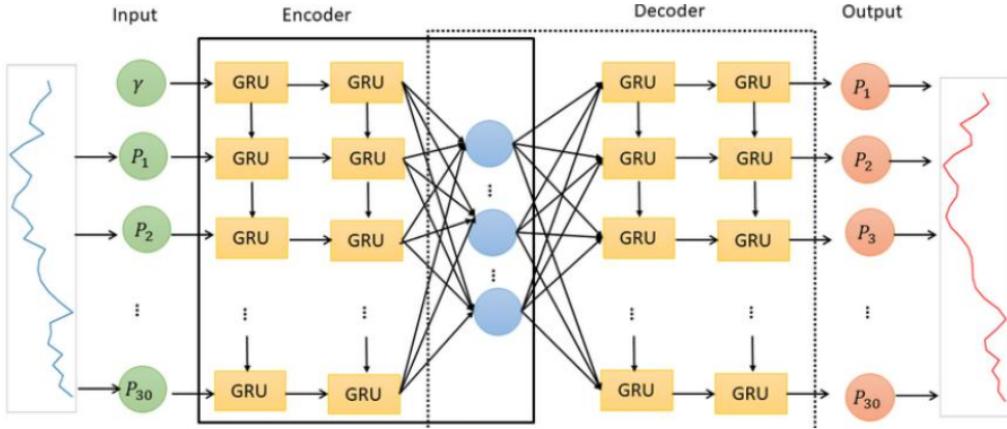


Figura 4.1: Architettura dell'Autoencoder generico basato su GRU per l'analisi di serie temporali.

4.1.4 Metodologia di Validazione

Per ottenere una stima affidabile delle capacità di generalizzazione dei modelli, sono state adottate due diverse strategie di validazione, scelte in base alla natura degli algoritmi e dei dati analizzati.

K-Fold Cross-Validation per Modelli Classici

Per i modelli di *machine learning* classici (KNN, Random Forest e Isolation Forest), è stata utilizzata una validazione incrociata con **K=10 fold**. Questa tecnica, robusta e ampiamente accettata, è essenziale specialmente con dataset di dimensioni limitate. Come discusso in letteratura, la K-Fold Cross-Validation è una tecnica potente per valutare la capacità di un modello di generalizzare su dati nuovi, riducendo la varianza nella stima delle performance [10]. Addestrando e valutando il modello più volte su diverse porzioni del dataset, si ottiene una misura delle performance più stabile e meno dipendente dalla specifica suddivisione dei dati.

Il processo garantisce che ogni osservazione del dataset venga utilizzata sia nel training set che nel test set almeno una volta nel corso delle diverse "fold", massimizzando l'utilizzo delle informazioni disponibili. Ad ogni iterazione, uno dei K sottoinsiemi (o "fold") viene utilizzato come test set, mentre i restanti $K-1$ vengono combinati per formare il training set. Le metriche di performance finali sono calcolate come la media dei risultati ottenuti in ciascuna delle K iterazioni.

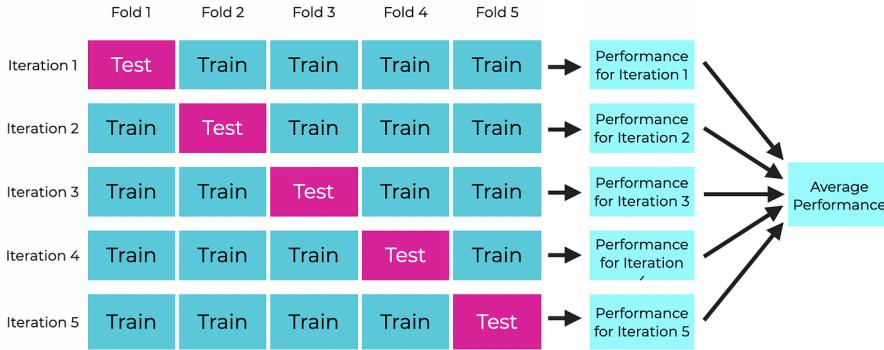


Figura 4.2: Rappresentazione grafica del processo di K-Fold Cross-Validation (con K=5). Il dataset viene suddiviso in 5 fold; il processo viene ripetuto 5 volte, utilizzando a ogni iterazione un fold diverso per il test e i restanti per il training.

Validazione per Serie Temporali

L'uso della K-Fold Cross-Validation tradizionale sui dati temporali comporterebbe l'addestramento su osservazioni future per predire il passato, un fenomeno noto come *data leakage* che produce metriche irrealisticamente ottimistiche. Anche la soluzione canonica, *TimeSeriesSplit*, che preserva l'ordine cronologico, si è rivelata inapplicabile a causa della particolare struttura del dataset.

Per superare questa criticità, è stata implementata una **suddivisione strategica**. Questa scelta segue il paradigma standard per l'addestramento di modelli ricostruttivi per l'anomaly detection, che prevede l'uso esclusivo di dati normali in fase di training per permettere al modello di apprendere una rappresentazione della normalità, per poi valutarlo sulla sua capacità di distinguere le anomalie [23]. I dati sono stati suddivisi come segue:

- **training Set:** composto dal 70% delle sequenze "normali", per addestrare il modello a ricostruire solo il comportamento atteso;
- **validation Set:** composto dal 15% delle sequenze "normali", utilizzato per calibrare la soglia statistica di anomalia;
- **test Set:** composto dal restante 15% di sequenze "normali" e da tutte le sequenze anomale, per valutare le performance finali del modello;

Questa strategia, pur rinunciando alla *Macro-Sequenzialità*, preserva la **micro-sequenzialità**, ovvero l'ordine temporale all'interno di ogni singola finestra.

4.1.5 Metriche di Valutazione

Per valutare e confrontare le performance dei modelli in modo oggettivo, è stato utilizzato un insieme completo di metriche quantitative e qualitative, scelte in accordo con le pratiche standard del settore [4].

Metriche di Classificazione

Sono state utilizzate le metriche standard per la valutazione dei classificatori binari:

- **Accuracy:** la proporzione di predizioni corrette sul totale. Sebbene intuitiva, è noto che possa essere fuorviante in presenza di classi sbilanciate;

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

- **Precision:** la proporzione di vere anomalie identificate tra tutte quelle predette come tali. Misura l'affidabilità degli allarmi generati;

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

- **Recall:** la capacità del modello di identificare tutte le anomalie effettivamente presenti. Come sottolineato in letteratura, il Recall è considerato una metrica critica in contesti di forte sbilanciamento come l'anomaly detection;

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

- **F1-Score:** la media armonica di Precision e Recall, che fornisce una misura bilanciata delle performance. È stata usata la sua variante *weighted* per pesare il contributo di ogni classe in base alla sua frequenza nel dataset;

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Analisi del Comportamento del Modello

Oltre alle metriche puntuali, sono stati usati strumenti grafici per analizzare il comportamento dei modelli durante l'addestramento.

Curve di Apprendimento (*Learning Curves*) Le curve di apprendimento sono state utilizzate per diagnosticare il comportamento dei modelli. L'analisi del divario tra la performance sul set di training e quella sul set di validazione è una tecnica standard per identificare problemi di bias (underfitting) e varianza (overfitting) [1].

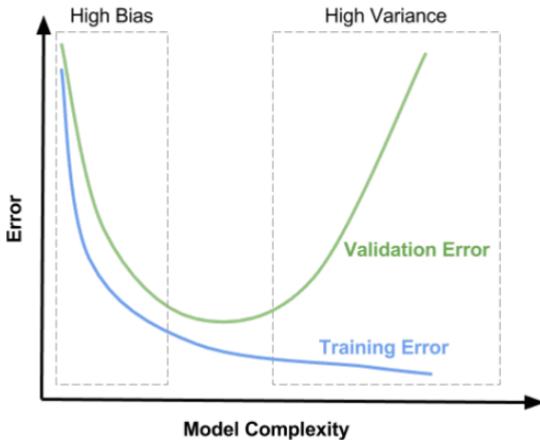


Figura 4.3: Esempio di curve di apprendimento per la diagnosi di bias e varianza. L’analisi del divario tra l’errore di training e l’errore di validazione permette di identificare condizioni di underfitting (High Bias) o overfitting (High Variance).

Metriche Computazionali

Infine, in linea con gli obiettivi del progetto EPS+ di fornire soluzioni efficienti per le PMI, sono stati monitorati il **tempo di addestramento** e la **memoria occupata** dai modelli. La valutazione dell’efficienza e della scalabilità è un aspetto cruciale, poiché spesso esiste un trade-off tra l’accuratezza di un modello e la sua complessità computazionale, un fattore non trascurabile in contesti operativi con risorse limitate [16].

4.2 Simulazione con Dataset Generato e Calcolo degli Health Score

Questo capitolo illustra la metodologia pratica della simulazione implementata per validare il calcolo degli "Health Score". Verrà descritto in dettaglio il funzionamento del prototipo, che utilizza il dataset sintetico, memorizzato su MongoDB e generato come descritto nel capitolo precedente, per calcolare in tempo reale le metriche di performance, disponibilità e resilienza.

Il culmine di questo processo è la produzione di un **Health Score Aggregato** (H_{agg}), un singolo indicatore progettato per fornire una valutazione immediata e comprensibile dello stato di salute complessivo dell’infrastruttura. La creazione di un indice aggregato a partire da molteplici indicatori (KPI) è una pratica consolidata in letteratura, in quanto permette di ridurre la complessità informativa e di fornire ai decisori una metrica sintetica per classificare le performance e identificare rapidamente le criticità [11]. La sua formula teorica è una media pesata dei punteggi

ottenuti dai pilastri fondamentali che rappresentano le aree chiave monitorate:

$$H_{agg} = (w_P \cdot P_{score}) + (w_A \cdot A_{score}) + (w_R \cdot R_{score})$$

Dove:

- $P_{score}, A_{score}, R_{score}$: Sono i punteggi normalizzati (da 0 a 1) rispettivamente dei pilastri di Performance, Availability e Resilience;
- w_P, w_A, w_R : Sono i pesi assegnati a ciascun pilastro, la cui somma è normalizzata a 1, che ne determinano l'importanza relativa;

Una considerazione fondamentale in questo modello è la potenziale soggettività nella scelta dei pesi (w). Per affrontare questa sfida, si propone come futuro sviluppo del sistema l'introduzione del concetto di "**Profilo di Rischio**". L'idea è di offrire all'utente la possibilità di scegliere un profilo che descriva il proprio contesto di business (es. "E-commerce", "Ufficio Standard"), a cui corrisponderebbe un insieme di pesi pre-configurato. Sebbene non implementata nel prototipo attuale, questa funzionalità renderebbe la configurazione più intuitiva e significativa dal punto di vista operativo, trasformando l'apparente debolezza della soggettività in un punto di forza basato sulla contestualizzazione.

4.2.1 Metodologia per lo Score di Performance (SPC)

Come anticipato nei capitoli precedenti, per fornire una valutazione oggettiva e dinamica delle performance di sistema, è stata adottata una metodologia basata sul **Controllo Statistico di Processo (SPC)** [7]. La scelta dell'SPC per il monitoraggio delle singole metriche è stata deliberata, in quanto offre vantaggi pratici fondamentali nel contesto delle PMI. In primo luogo, garantisce una totale **interpretabilità (Explainability)**: ogni allarme è direttamente riconducibile a una regola statistica semplice e verificabile, come "otto punti sopra la media", un aspetto cruciale per la fiducia degli amministratori di sistema. In secondo luogo, a differenza di molti modelli di machine learning "data-hungry", l'SPC ha un **basso requisito di dati**, potendo stabilire una baseline affidabile fin dai primi giorni di monitoraggio. Infine, la sua **efficienza computazionale** rende il calcolo dei limiti di controllo un'operazione leggera, con un impatto minimo sulle risorse che sta monitorando. L'obiettivo di questo pilastro è quindi rispondere a una domanda fondamentale: *"Il sistema sta lavorando in modo ottimale? Il comportamento delle macchine monitorate è normale?"*.

Per rispondere a questa domanda, la scelta delle metriche da monitorare non è stata casuale, ma si è fondata su indicatori che la letteratura consolidata identifica come i pilastri per l'analisi delle performance di qualsiasi sistema informatico [12]. Le metriche selezionate sono **CPU, RAM e I/O Wait**, in quanto il loro monitoraggio congiunto permette di ottenere una visione olistica della salute del sistema e di diagnosticare con precisione la maggior parte dei "colli di bottiglia".

Avendo già introdotto i principi teorici delle Carte di Controllo (Capitolo 2) e la struttura del dataset a due fasi (Capitolo 3), questa sezione si concentra sui dettagli implementativi del modello. L'approccio si articola nelle seguenti fasi operative:

1. Calcolo della Baseline e Scelta della Carta di Controllo La fase iniziale del processo consiste nello stabilire una baseline statistica per ogni metrica di ogni macchina. Sfruttando i dati "puliti" generati nella fase di *baseline*, il sistema calcola i parametri per una specifica Carta di Controllo. Come illustrato dalla metodologia Six Sigma, la scelta della carta corretta dipende dalla natura dei dati monitorati [7]. Poiché le metriche di performance (CPU, RAM, etc.) sono **dati di tipo variabile** raccolti come **misure individuali** nel tempo (sottogruppo di dimensione $n=1$), la carta metodologicamente adatta è la **Carta XmR (*Individuals and Moving Range*)**.

È importante sottolineare fare una considerazione: le carte XmR sono tradizionalmente associate a processi con una distribuzione simmetrica o approssimativamente normale. Tuttavia, le metriche di performance IT come l'utilizzo della CPU o della RAM sono per loro natura **non simmetriche**: hanno un limite fisico inferiore a 0 e superiore a 100, e il loro comportamento "normale" è spesso caratterizzato da valori bassi con picchi occasionali verso l'alto, risultando in una distribuzione asimmetrica. Nonostante ciò, l'uso della carta XmR è giustificato dal fatto che i test di stabilità selezionati in questo lavoro (come il Test 1 per la violazione dei limiti e i test basati su pattern come il Run Test) sono robusti a deviazioni dalla normalità. Il loro scopo non è verificare la conformità a una distribuzione teorica, ma rilevare **cambiamenti significativi e pattern non casuali**, segnali inequivocabili di "cause eccezionali" che impattano il processo, indipendentemente dalla sua distribuzione di base.

Vengono quindi calcolati i seguenti limiti di controllo, che definiscono il "corridoio di normalità" del processo:

- **Center Line (CL_X)**: La media dei valori della baseline, \bar{X} ;
- **Upper e Lower Control Limit (UCL_X, LCL_X)**: I limiti superiore e inferiore per il grafico degli individui, calcolati rispettivamente come $\bar{X} + 2.66 \cdot \bar{mR}$ e $\max(0, \bar{X} - 2.66 \cdot \bar{mR})$;
- **Upper Control Limit (UCL_{mR})**: Il limite superiore per il grafico della variabilità (*Moving Range*), calcolato come $3.268 \cdot \bar{mR}$ ¹, dove \bar{mR} è la media delle differenze assolute tra punti consecutivi;

¹I valori 2.66 e 3.268 sono costanti statistiche standard, derivate dalla teoria del Controllo Statistico di Processo per le carte XmR. Nello specifico, si basano su un sottogruppo di dimensione $n=2$ (poiché il *moving range* è calcolato tra due punti consecutivi) e corrispondono rispettivamente ai fattori $3/d_2$ e D_4 dalle tabelle di controllo qualità [7].

2. Rilevamento Anomalie tramite Test di Stabilità Con la *baseline* stabilita, il sistema passa al monitoraggio attivo, dove ogni nuovo dato viene sottoposto a una serie gerarchica di test di stabilità. Questi test sono stati scelti per la loro robustezza in contesti IT, dove i dati non seguono necessariamente una distribuzione normale. I test implementati sono:

- **Test a Priorità Altissima (Critici):** Rilevano eventi estremi e improvvisi;
 - **Test 1 (su Grafico X e mR):** Un singolo punto viola i limiti di controllo, segnalando un picco o un'esplosione di instabilità;
 - **Saturazione:** La metrica raggiunge il suo limite fisico del 100%;
- **Test a Priorità Alta (Sistemici):** Identificano cambiamenti stabili nel comportamento del processo;
 - **Test 4 (Run Test):** Otto o più punti consecutivi si trovano dallo stesso lato della media, indicando uno *shift*;
 - **Test 8 (Trend Lineare):** Sei o più punti mostrano un andamento costantemente crescente o decrescente;
 - **Test 7 (Trend Oscillatorio):** Quattordici punti si alternano in modo sistematico;
- **Test a Priorità Bassa (Allarmi Precoci):** Rilevano pattern che possono anticipare un futuro degrado del processo;
 - **Test 2 e 3 (Test delle Zone):** Identificano raggruppamenti di punti nelle "zone di allerta" (oltre 1 e 2 sigma dalla media), segnalando un possibile *shift* imminente del processo;

Il fallimento di un test sistemico (4, 7 o 8) non solo impatta lo score, ma segnala la necessità di ricalcolare la *baseline* per adattare il monitoraggio al nuovo regime di funzionamento del sistema [7].

3. Calcolo del P-Score e Simulazione L'esito dei test viene tradotto in uno **score per singola metrica** (s_m) su una scala da 0 (critico) a 1 (ottimale), come riassunto nella Tabella 4.1.

Tabella 4.1: Mappatura tra Stato della Metrica e Punteggio Assegnato.

Stato della Metrica	Causa Scatenante (Esempio)	Punteggio (s_m)
In Controllo	Nessun test fallito	1.0 (Ottimale)
Degradato	Fallimento Test 4, 7, 8	0.4 (Problema sistemico)
Critico	Fallimento Test 1 (X o mR)	0.1 (Critico)
Saturazione	Valore raggiunge il 100%	0.0 (Saturato)

Il **Performance Score** (P_{score}) finale per una macchina è calcolato come media pesata degli score delle sue metriche. L'uso dei pesi (w) è cruciale perché permette di personalizzare l'importanza di CPU, RAM e I/O Wait per uno specifico carico di lavoro, con la condizione che la loro somma sia normalizzata a 1 ($w_{cpu} + w_{ram} + w_{io} = 1$).

$$P_{score}(X) = (w_{cpu} \cdot s_{cpu}(X)) + (w_{ram} \cdot s_{ram}(X)) + (w_{io} \cdot s_{io}(X))$$

Successivamente, per ottenere una visione d'insieme dell'intera infrastruttura, i punteggi individuali vengono aggregati in un **P_{score} globale**. Questo calcolo utilizza una media pesata in cui ogni macchina contribuisce in base alla sua **criticità di business** ($\text{criticità}(X)$). Questo approccio garantisce che un problema su un server di produzione abbia un impatto maggiore sullo score globale rispetto a un'anomalia identica su un server meno importante.

$$P_{score}(\text{globale}) = \frac{\sum_{\text{tutte le macchine } X} (P_{score}(X) \cdot \text{criticità}(X))}{\sum_{\text{tutte le macchine } X} \text{criticità}(X)}$$

La validazione di questa logica avviene tramite una simulazione in cui un'applicazione web, sviluppata in Flask, espone delle API REST. Il front-end richiede i dati un punto alla volta, simulando un flusso real-time. Per ogni richiesta, il backend legge il dato successivo dal database MongoDB, applica i test SPC e calcola il P_{score} aggiornato, restituendo il risultato per la visualizzazione.

4.2.2 Implementazione del Generatore di Dati di Performance

La creazione di un ambiente di test controllato e ripetibile è stata resa possibile dallo script `generate_metrics_spc.py`. Come illustrato nel Listato 4.1, lo script definisce innanzitutto dei **profili statistici** per ogni macchina, caratterizzati da una media e una deviazione standard specifiche per ogni metrica. Questo permette di generare un comportamento di base realistico e differenziato.

La creazione di un ambiente di test controllato e ripetibile è stata resa possibile dallo script `generate_metrics_spc.py`. Nello specifico, la simulazione è stata configurata per generare dati su una **durata totale** di 14 giorni, con una **frequenza di campionamento** di un punto dati ogni minuto per ogni metrica (CPU, RAM, I/O Wait). Di questo periodo, i primi 3 giorni (corrispondenti a 4.320 punti per metrica) sono stati designati come **periodo di baseline**, durante il quale i dati sono stati generati in condizioni di stabilità per permettere il calcolo dei limiti di controllo SPC.

La logica fondamentale risiede nel motore di iniezione delle anomalie: una serie di funzioni ('genera_spike', 'genera_shift', etc.) è progettata per creare programmaticamente sequenze di dati che violano specifiche regole SPC. Durante la fase di "test" della generazione, una di queste funzioni viene invocata con una probabilità

definita, inserendo un'anomalia mirata nel flusso di dati. Questo approccio garantisce che il sistema di rilevamento possa essere validato in modo rigoroso contro tutti i pattern anomali che è stato progettato per identificare.

```

1 # --- PROFILI DELLE MACCHINE ---
2 # Ogni macchina ha una sua "personalita'" statistica.
3 PROFILI_MACCHINE = [
4     {"machine_id": "WebServer-01", "profile": {"cpu": {"mean": 25,
5         "std_dev": 4}, "ram": {"mean": 40, "std_dev": 6}, "io_wait": {"mean": 5, "std_dev": 1.5}}, {"machine_id": "Database-Server-01", "profile": {"cpu": {"mean": 60, "std_dev": 12}, "ram": {"mean": 75, "std_dev": 5}, "io_wait": {"mean": 20, "std_dev": 8}}}, ]
6
7
8
9
10]
11
12# --- MOTORE DI INIEZIONE ANOMALIE ---
13def genera_spike(profilo):
14    """ OBIETTIVO: Attivare il Test 1 (Violazione Limite 3-Sigma).
15    """
16    media, sigma = profilo['mean'], profilo['std_dev']
17    # Genera un valore molto al di fuori dei 3-sigma
18    valore_anomalo = media + random.choice([-1, 1]) * random.uniform(3.5, 5) * sigma
19    return [clamp(valore_anomalo)], "Test 1 - Violazione Limite 3-Sigma"
20
21def genera_shift(profilo, lunghezza=9):
22    """ OBIETTIVO: Attivare il Test 4 (Run Above/Below Centerline).
23    """
24    media, sigma = profilo['mean'], profilo['std_dev']
25    # Sposta la media di circa 1.5-2 sigma
26    nuova_media = media + random.choice([-1, 1]) * random.uniform(1.5, 2.0) * sigma
27    valori = np.random.normal(nuova_media, sigma, lunghezza)
28    return [clamp(v) for v in valori], "Test 4 - Run Above/Below Centerline"
29
30def main():
31    # ... (connessione al DB e setup)
32
33    # Ciclo principale di generazione
34    for machine in PROFILI_MACCHINE:
35        for idx in range(total_iterations):
36            is_baseline = idx < baseline_end_index

```

```

35      # Durante la fase di test, con una certa probabilita',
36      # si inietta un'anomalia
37      if not is_baseline and random.random() <
38          PROBABILITA_INIZIO_ANOMALIA:
39          anomaly_func = random.choice(ANOMALY_GENERATORS)
40          # ... (logica per scegliere metrica e generare dati
41          anomali)
42      else:
43          # Genera un punto dato normale
44          # ... (crea e inserisce documento normale)

```

Listing 4.1: Estratto dal generatore di dati `generate_metrics_spc.py`, che mostra i profili macchina e il motore di iniezione delle anomalie.

4.2.3 Implementazione della Simulazione Interattiva

Per validare la metodologia in un ambiente dinamico, è stata sviluppato un modulo web interattivo, spiegato nell'ultimo capitolo. Questa sezione illustra i componenti software chiave del backend e del frontend che realizzano la simulazione del monitoraggio in tempo reale.

Backend: API di Servizio e Analisi Il backend, sviluppato con il micro-framework Python **Flask**, espone un'API RESTful che funge da motore per la simulazione. L'endpoint principale, mostrato nel Listato 4.2, gestisce l'intero flusso di analisi per ogni richiesta: inizialmente, riceve la richiesta dal frontend per un punto dati specifico di una macchina, quindi interroga il database MongoDB per recuperare il dato corrispondente. Successivamente, invoca il modulo di analisi SPC per valutare il nuovo dato e, infine, restituisce una risposta in formato JSON contenente il valore della metrica, lo stato del processo e il P-Score aggiornato.

```

1 from flask import Flask, jsonify
2 from .sixsigma_utils import calcola_baseline, monitora_nuovo_dato
3
4 app = Flask(__name__)
5
6 # Cache per le baseline per evitare ricalcoli a ogni richiesta
7 baseline_cache = {}
8
9 @app.route('/api/data/<machine_id>/<int:offset>')
10 def get_data_point(machine_id, offset):
11     # 1. Recupera il dato dal database MongoDB usando l'offset
12     data_point = collection.find_one({"machine_id": machine_id},
13                                     skip=offset)
14
15     # 2. Calcola o recupera dalla cache la baseline per ogni
16     # metrica

```

```

15     if not baseline_cache.get(machine_id):
16         baseline_cache[machine_id] = {
17             "cpu": calcola_baseline(machine_id, "cpu",
18             baseline_cache),
19             # ... calcolo per ram e io_wait
20         }
21
22     # 3. Analizza il nuovo dato con il motore SPC
23     risultatiAnalisi = {}
24     for metrica in ["cpu", "ram", "io_wait"]:
25         valoreAttuale = data_point['metrics'][f'{metrica}_percent']
26         ]
27         baselineMetrica = baseline_cache[machine_id][metrica]
28         # Invocazione della funzione di analisi dal modulo utils
29         risultatiAnalisi[metrica] = monitora_nuovo_dato(
30             valoreAttuale, baselineMetrica)
31
32     # 4. Calcola il P-Score aggregato e prepara la risposta JSON
33     p_score = calcola_p_score_aggregato(risultatiAnalisi)
34
35     return jsonify({
36         "timestamp": data_point['timestamp'],
37         "p_score": p_score,
38         "stati": {m: r['stato'] for m, r in risultatiAnalisi.items()
39     },
40         # ... altri dati per il frontend
41     })

```

Listing 4.2: Endpoint API Flask per la gestione della simulazione.

Frontend: Interrogazione e Visualizzazione Dinamica Il frontend è una single-page application che utilizza **JavaScript** per creare un'esperienza interattiva. La logica principale è gestita da due funzioni chiave. La prima, ‘fetchData’, eseguita a intervalli regolari (es. ogni secondo), interroga l’endpoint del backend per recuperare il punto dati successivo, come mostrato nel Listato 4.3. La seconda, ‘updateDashboard’, riceve la risposta JSON e aggiorna dinamicamente gli elementi dell’interfaccia, come i grafici (realizzati con Chart.js) e il valore del P-Score.

```

1 // Variabile per tenere traccia del punto dati corrente
2 let currentOffset = 1;
3 let simulationInterval = null;
4
5 // Funzione principale che avvia la simulazione
6 function startSimulation(machineId) {
7     // ... setup iniziale ...
8
9     // Avvia un timer che chiama fetchData ogni secondo
10    simulationInterval = setInterval(() => {
11        fetchData(machineId);

```

```

12     }, 1000);
13 }
14
15 // Funzione che interroga l'API per ottenere il prossimo punto dato
16 function fetchData(machineId) {
17     // Costruisce l'URL dell'API
18     const apiUrl = `/api/data/${machineId}/${currentOffset}`;
19
20     // Esegue la chiamata all'API
21     fetch(apiUrl)
22         .then(response => response.json())
23         .then(data => {
24             // Se la chiamata ha successo, aggiorna la dashboard
25             // con i nuovi dati
26             updateDashboard(data);
27
28             // Incrementa l'offset per la prossima chiamata
29             currentOffset++;
30         })
31         .catch(error => {
32             // Se i dati sono finiti o c'e' un errore, ferma la
33             // simulazione
34             console.error("Simulazione terminata o errore:", error);
35             clearInterval(simulationInterval);
36         });
37 }

```

Listing 4.3: Funzione JavaScript per l'interrogazione periodica dell'API backend.

Questa architettura client-server, basata su API, non solo permette di disaccoppiare la logica di analisi (backend) dalla presentazione (frontend), ma modella un approccio scalabile e robusto, rappresentativo di un sistema di monitoraggio distribuito.

Il motore analitico invocato dal backend risiede nella funzione `monitora_nuovo_dato`, mostrata nel Listato 4.4. Questo spezzone di codice è il cuore del sistema di rilevamento: riceve il nuovo valore e la baseline, e applica in sequenza i test di stabilità descritti in precedenza per determinare lo stato del processo.

```

1 # Funzione cuore del motore SPC
2 def monitora_nuovo_dato(valore, valore_precedente, cronologia,
3     baseline):
4     # ... (calcolo delle zone sigma e del moving range) ...
5
6     # === TEST 1: VIOLAZIONE LIMITI X (Priorita Altissima) ===
7     if not (lcl_x <= valore <= ucl_x):
8         # ... (logica per gestire la violazione e restituire lo
9         # stato "Critico")
10
11    # === TEST mR: VIOLAZIONE LIMITE MOVING RANGE ===

```

```

10     mr_value = abs(valore - valore_precedente)
11     if mr_value > ucl_mr:
12         # ... (logica per gestire la violazione e restituire lo
13         # stato "Critico")
14
15     # === TEST 4: RUN TEST (8 punti consecutivi stesso lato media)
16     ===
17
18     if len(cronologia_completa) >= 8:
19         last_points = cronologia_completa[-8:]
20         above_cl = all(p > cl for p in last_points)
21         below_cl = all(p < cl for p in last_points)
22         if above_cl or below_cl:
23             # ... (logica per gestire lo shift e restituire "
24             Degradato")
25
26     # ... (altri test) ...
27
28     # === TUTTI I TEST PASSATI - PROCESSO IN CONTROLLO ===
29     return {"stato": "In Controllo", "score": 1.0, ...}

```

Listing 4.4: Estratto dal motore di analisi SPC (`sixsigma_utils.py`).

Infine, una volta ricevuta la risposta dall'API, il frontend utilizza la funzione `updateDashboard` (Listato 4.5) per tradurre i dati JSON in aggiornamenti visivi, completando il ciclo della simulazione.

```

1 // Funzione che aggiorna l'interfaccia con i dati ricevuti dal
2 // backend
3 function updateDashboard(data) {
4     // Aggiorna il valore numerico del P-Score
5     const scoreElement = document.getElementById('score-value');
6     scoreElement.textContent = data.p_score.toFixed(3);
7
8     // Applica una classe CSS per cambiare colore in base allo
9     // score
10    if (data.p_score > 0.8) {
11        scoreElement.className = 'score-value score-excellent';
12    } else if (data.p_score > 0.5) {
13        scoreElement.className = 'score-value score-good';
14    } else {
15        scoreElement.className = 'score-value score-poor';
16    }
17
18    // Per ogni metrica (cpu, ram, io_wait), aggiorna il grafico
19    // corrispondente
20    for (const metrica of ['cpu', 'ram', 'io_wait']) {
21        const chart = charts[`#${metrica}Chart`];
22        const valore = data.metrics[`#${metrica}_percent`];
23        const timestamp = new Date(data.timestamp).
24            toLocaleTimeString();
25    }

```

```

22     // Aggiunge il nuovo punto al grafico e lo rinfresca
23     updateChart(chart, timestamp, valore, ...);
24 }
25 }
```

Listing 4.5: Estratto della funzione JavaScript per l'aggiornamento della dashboard.

4.2.4 Metodologia per lo Score di Availability

Per quantificare la disponibilità in modo significativo, la metodologia si fonda sulla definizione formale di Availability e la arricchisce con i principi della Site Reliability Engineering (SRE), già discussi nel Capitolo 2.

1. Definizione Formale di Availability La disponibilità (A) di un sistema è definita come la frazione di tempo in cui esso è operativo. Concettualmente, può essere espressa come il rapporto tra il tempo di funzionamento (Uptime) e il tempo totale di osservazione (Uptime + Downtime). Questa definizione fondamentale trova la sua formalizzazione statistica nella seguente equazione, dove MTTF (Mean Time To Failure) rappresenta il tempo medio di funzionamento e MTTR (Mean Time To Repair) il tempo medio di ripristino [29]:

$$A = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} \quad (4.1)$$

Partendo da questa base teorica, la nostra metodologia si concentra su come misurare e valutare efficacemente questo rapporto in un contesto operativo.

2. Logica di Misura e Definizione di SLI e SLO Per tradurre la definizione teorica in una metrica pratica, è cruciale definire rigorosamente come viene misurato lo stato di un servizio. Un asset è considerato **UP** solo se si verificano due condizioni: la **macchina** sottostante è raggiungibile **E** il **servizio** applicativo specifico sta funzionando correttamente. La verifica avviene tramite un duplice controllo:

- **controllo macchina:** Raggiungibilità di rete (es. Ping ICMP) e salute del sistema operativo (es. interrogazioni SNMP);
- **controllo servizio:** Controlli a livello applicativo che simulano una reale interazione utente, come una richiesta HTTP GET per un web server o una query SELECT 1; per un database;

Sulla base di questa logica di misura, si applica la gerarchia di indicatori SRE:

- **Service Level Indicator (SLI):** È la misurazione diretta della disponibilità, calcolata come la percentuale di tempo UP in una finestra temporale. La sua formula è l'implementazione pratica della definizione formale di Availability:

$$SLI = \frac{\text{Minuti totali in stato UP}}{\text{Minuti totali nella finestra}} \times 100 \quad (4.2)$$

- **Service Level Objective (SLO):** È il valore target che l'SLI deve raggiungere (es. 99.9%);

In questa implementazione, si adotta inoltre la scelta progettuale di trattare ogni servizio critico come un **asset di business separato**, anche se co-locato sulla stessa macchina, per massimizzare la **granularità della diagnosi**.

3. Calcolo dell>Error Budget e dello Score (A_{score}) La metodologia SRE si focalizza sulla gestione proattiva dell'**Error Budget**, ovvero la quantità massima di disservizio ammissibile definita dallo SLO. L'Availability Score (A_{score}) è quindi formulato come una misura di **quanto Error Budget residuo rimane**. Questo approccio rende lo score più sensibile: un valore di 1.0 indica che il budget è intatto, mentre 0.0 indica che è stato consumato, violando lo SLO.

Dati il *Downtime Accumulato* in minuti e l'*Error Budget Totale*, la formula per l' A_{score} di un singolo servizio è:

$$A_{score} = \max \left(0, 1 - \frac{\text{Downtime Accumulato}}{\text{Error Budget Totale}} \right) \quad (4.3)$$

Questa funzione decresce linearmente con il downtime, fornendo un segnale inequivocabile al raggiungimento della soglia critica.

4. Calcolo dell'A-Score Aggregato In modo del tutto analogo a quanto fatto per il Performance Score, i punteggi dei singoli servizi vengono aggregati in un A_{score} globale. Il calcolo si basa su una **media pesata**, in cui ogni servizio contribuisce in base alla sua **criticità di business**.

$$A_{score}(\text{globale}) = \frac{\sum_{\text{servizi } S} (A_{score}(S) \cdot \text{criticità}(S))}{\sum_{\text{servizi } S} \text{criticità}(S)} \quad (4.4)$$

4.2.5 Implementazione del Generatore di Dati di Availability

La validazione della metodologia descritta richiede un dataset che simuli in modo realistico gli eventi di disservizio. A tal fine, la logica di generazione è stata implementata nella funzione `_generate_availability_data`, contenuta nello script `seed_database.py`. La simulazione è stata configurata per operare su una **durata**

totale di 14 giorni, durante i quali lo stato di ogni servizio (UP/DOWN) è stato campionato con una **frequenza** di un minuto, al fine di creare una cronologia granulare degli eventi.

L'implementazione modella un comportamento più verosimile di un semplice fallimento casuale: un servizio in stato di downtime ha una probabilità maggiore di rimanere in tale stato nell'intervallo successivo, simulando così un tempo di ripristino (MTTR - Mean Time To Repair). Come mostrato nel Listato 4.6, un servizio funzionante ha una bassa probabilità di fallire (**failure_prob**), ma una volta in stato DOWN, ha il 50% di probabilità di rimanere in tale stato, rendendo i periodi di disservizio più realistici e contigui.

```

1 def _generate_availability_data(storage_manager, services,
2                                 start_date, days):
3     # ... (omissis)
4     for service in services:
5         current_status = 1.0 # Inizia come UP
6         failure_prob = 0.02
7
8         for i in range(days * 1440): # Un campione al minuto
9             timestamp = start_date + timedelta(minutes=i)
10
11            # Un servizio DOWN ha il 50% di possibilita' di
12            # rimanere DOWN
13            if current_status == 0.0 and random.random() < 0.5:
14                pass # Rimane DOWN
15            elif random.random() < failure_prob:
16                current_status = 0.0 # Va DOWN
17            else:
18                current_status = 1.0 # E' UP
19
20            # Creazione del documento e inserimento
21            metric = MetricDocument(
22                timestamp=timestamp,
23                meta={"asset_id": service["_id"],
24                      "metric_name": "availability_status"},
25                value=current_status
26            )
27            storage_manager.insert_metric(metric.to_dict())

```

Listing 4.6: Estratto dalla funzione `_generate_availability_data` in `seed_database.py`.

4.2.6 Implementazione della Simulazione Interattiva

La simulazione interattiva permette di validare la logica di calcolo dell'Availability Score in un ambiente dinamico che emula il monitoraggio in tempo reale. L'architettura è composta da un backend analitico e un frontend di visualizzazione.

Backend: Motore di Analisi La logica di calcolo per la simulazione risiede nella classe `CumulativeAvailabilityAnalyzer`, implementata nel file `cumulative_availability_analyzer.py`. Questo componente software agisce come motore analitico del backend, disaccoppiato dalla presentazione, ed è progettato per operare su una finestra temporale mobile, ricalcolando lo stato della disponibilità a ogni nuovo dato ricevuto.

Come illustrato nel Listato 4.7, la classe viene inizializzata con un obiettivo (SLO) e una finestra di tempo. Il metodo `process_entry` aggiorna i contatori di uptime e downtime, mentre il metodo `get_summary` applica le formule definite nella metodologia per calcolare l'SLI corrente, l>Error Budget residuo e l' A_{score} finale.

```

1  class CumulativeAvailabilityAnalyzer:
2      def __init__(self, slo_percentage, window_days):
3          self.slo_percentage = slo_percentage
4          self.window_seconds = window_days * 24 * 60 * 60
5          self.entries = deque()
6          self.total_uptime_seconds = 0
7          self.total_downtime_seconds = 0
8          # ...
9
10     def process_entry(self, timestamp, value, interval_seconds=60):
11         # ... (logica per aggiungere nuove entry e rimuovere quelle vecchie)
12         if value == 1.0:
13             self.total_uptime_seconds += interval_seconds
14         else:
15             self.total_downtime_seconds += interval_seconds
16
17     def get_summary(self):
18         # ... (calcolo total_seconds)
19
20         # Calcolo SLI
21         sli = (self.total_uptime_seconds / total_seconds) * 100 if
22         total_seconds > 0 else 100
23
24         # Calcolo Error Budget
25         total_budget_seconds = self.window_seconds * (100 - self.
26         slo_percentage) / 100
27         error_budget_consumed = self.total_downtime_seconds
28
29         # Calcolo Score
30         score = max(0, 1 - (error_budget_consumed /
31         total_budget_seconds)) if total_budget_seconds > 0 else 1.0
32
33     return { "a_score": score, "sli": sli, ... }

```

Listing 4.7: Estratto della logica di calcolo in `cumulative_availability_analyzer.py`.

Frontend: Visualizzazione Dinamica Il frontend, implementato nel file `availability_dashboard.html`, si occupa di interrogare periodicamente l'API del backend per recuperare i dati aggiornati e di visualizzarli in modo intuitivo. Utilizzando JavaScript, la dashboard effettua chiamate asincrone per ottenere un oggetto JSON contenente l'SLI, l' A_{score} e altre metriche correlate.

Come mostrato nel Listato 4.8, la funzione `updateDashboard` riceve questi dati e aggiorna dinamicamente gli elementi del DOM: indicatori numerici, barre di progresso per l>Error Budget e grafici. Questo ciclo di richiesta-risposta-aggiornamento costituisce il cuore della simulazione interattiva, fornendo un feedback visivo immediato sullo stato di salute della disponibilità dei servizi.

```

1 function updateDashboard() {
2     // Esegue la chiamata all'API per ottenere i dati aggregati
3     fetch(`/api/availability/summary?service=${currentServiceId}`)
4         .then(response => response.json())
5         .then(data => {
6             // Aggiorna il valore numerico dell'Availability Score
7             const scoreElement = document.getElementById('a-score-
value');
8                 scoreElement.textContent = data.a_score.toFixed(3);
9
10            // Aggiorna la barra di progresso dell'Error Budget
11            const budgetBar = document.getElementById('error-budget-
bar');
12            const consumedPercentage = data.
13            error_budget_consumed_percentage;
14                budgetBar.style.width = `${consumedPercentage}%`;
15
16            // Aggiorna i valori di SLI e SLO
17            document.getElementById('sli-value').textContent =
18                `${data.sli.toFixed(4)}%`;
19            document.getElementById('slo-value').textContent =
20                `${data.slo.toFixed(2)}%`;
21
22            // ... (logica per aggiornare altri elementi)
23        })
24        .catch(error => console.error('Errore:', error));
25
26 // Avvia l'aggiornamento periodico
27 setInterval(updateDashboard, 2000); // Aggiorna ogni 2 secondi

```

Listing 4.8: Estratto della logica JavaScript per l'aggiornamento della dashboard.

4.2.7 Metodologia e Implementazione per lo Score di Resilience

Il terzo e ultimo pilastro dell'Health Score aggregato è la Resilience (Resilienza). Questo sezione illustra la metodologia e l'implementazione del modulo che risponde a una domanda cruciale per la continuità operativa: "**L'infrastruttura è in grado di riprendersi efficacemente da un incidente?**". L'approccio adottato si basa su metriche standard di *Disaster Recovery*, come l'RPO e l'RTO, già introdotte nel Capitolo 2, per fornire una valutazione quantitativa della capacità di recupero del sistema.

Per quantificare la resilienza, la metodologia si concentra sulla valutazione delle performance del sistema di backup, che rappresenta la prima linea di difesa contro la perdita di dati. L'approccio per aggregare diverse metriche in un singolo score si ispira a framework come il **MITRE ATT&CK**, che aggregano la conoscenza di molteplici tecniche e tattiche per fornire un quadro generale sicurezza [28, 3]. Allo stesso modo, il nostro R_{score} aggrega indicatori chiave per rappresentare la salute complessiva della strategia di backup.

1. Definizione delle Metriche Fondamentali La valutazione si basa sulle metriche di continuità operativa più importanti, definite dal NIST [27], a cui si aggiunge un indicatore operativo sull'affidabilità del processo:

- **Recovery Point Objective (RPO)**: Indica la quantità massima di dati che l'azienda è disposta a perdere, misurata in tempo. Rappresenta la freschezza richiesta per l'ultimo backup disponibile;
- **Recovery Time Objective (RTO)**: Indica il tempo massimo accettabile per ripristinare un servizio dopo un'interruzione;
- **Backup Success Rate**: La percentuale di backup completati con successo sul totale dei backup eseguiti in una determinata finestra temporale;

2. Logica di Calcolo degli Score e Considerazioni Implementative Ogni metrica contribuisce al calcolo di uno score normalizzato (da 0 a 1).

- **RPO Score (s_{rpo})**: Misura quanto l'ultimo backup disponibile sia "recente" rispetto all'obiettivo prefissato. Uno score di 1.0 indica che l'RPO è pienamente rispettato, mentre scende linearmente a 0 man mano che il tempo trascorso dall'ultimo backup si avvicina al doppio del target RPO, segnalando un rischio crescente;

$$s_{rpo} = \max \left(0, 1 - \frac{\text{Minuti dall'ultimo backup}}{\text{RPO Target in minuti}} \right) \quad (4.5)$$

- **Backup Success Rate Score (s_{bsr}):** Questo score corrisponde direttamente alla percentuale di successo dei backup, fornendo una misura chiara dell'affidabilità del processo.

$$s_{bsr} = \frac{\text{Numero di backup con successo}}{\text{Numero totale di backup eseguiti}} \quad (4.6)$$

È fondamentale una nota implementativa riguardo all'**RTO**. Sebbene l'RTO sia una metrica di resilienza critica, la sua misurazione oggettiva richiederebbe l'esecuzione periodica di un processo di ripristino reale o simulato, con la relativa misurazione dei tempi. Tale procedura è complessa da automatizzare e va oltre lo scopo del prototipo attuale, che si basa sull'analisi di dati facilmente reperibili tramite API (come lo stato e il timestamp dei backup, descritti nel Capitolo 3). Pertanto, si è scelto di focalizzare l'implementazione corrente su RPO e Success Rate, che sono direttamente misurabili dai log di backup. L'integrazione di test di ripristino automatici per la misurazione dell'RTO rappresenta uno degli **sviluppi futuri** per il sistema.

3. Calcolo del Resilience Score (R_{score}) Aggregato Il Resilience Score finale per un singolo asset è calcolato come media pesata degli score delle metriche implementate.

$$R_{score}(X) = (w_{rpo} \cdot s_{rpo}(X)) + (w_{bsr} \cdot s_{bsr}(X))$$

Infine, in modo analogo agli altri pilastri, i punteggi dei singoli asset vengono aggregati in un R_{score} **globale** tramite una media pesata basata sulla **criticità di business**.

$$R_{score}(\text{globale}) = \frac{\sum_{\text{asset } X} (R_{score}(X) \cdot \text{criticità}(X))}{\sum_{\text{asset } X} \text{criticità}(X)}$$

4.2.8 Implementazione del Generatore di Dati di Resilience

La validazione della metodologia ha richiesto la creazione di un dataset che simuli l'esecuzione periodica dei processi di backup. A tal fine, la logica di generazione è stata implementata nella funzione `_generate_resilience_data` (nello script `seed_database.py`), configurata per operare su una **durata totale** di 14 giorni. All'interno di questo periodo, è stata simulata una **frequenza di esecuzione** di un job di backup al giorno per ogni asset, schedulato in orario notturno. Come mostrato nel Listato 4.9, l'esito di ogni backup (successo o fallimento) viene determinato da una probabilità predefinita (`success_rate`), permettendo di generare una cronologia realistica di eventi per testare il calcolo del *Backup Success Rate Score*.

```

1 def _generate_resilience_data(storage_manager, assets_to_backup,
2     start_date, days):
3     success_rate = 0.95
4     for asset in assets_to_backup:
5         for day in range(days):
6             backup_time = (start_date + timedelta(days=day)).
7             replace(hour=2, minute=0)
8
9             # Determina l'esito del backup
10            if random.random() < success_rate:
11                backup_completed = 1.0
12                # ... (altri metadati per il successo)
13            else:
14                backup_completed = 0.0
15                # ... (altri metadati per il fallimento)
16
17            # Creazione del documento e inserimento
18            metric = MetricDocument(
19                timestamp=backup_time,
20                meta={"asset_id": asset["_id"], "metric_name": "backup_status"},
21                value={"backup_completed": backup_completed, ...})
22
23            storage_manager.insert_metric(metric.to_dict())

```

Listing 4.9: Estratto dalla funzione `_generate_resilience_data` in `seed_database.py`.

4.2.9 Implementazione della Simulazione Interattiva

La simulazione interattiva permette di validare la logica di calcolo del Resilience Score, emulando l'aggiornamento dello stato man mano che i dati sui backup vengono resi disponibili.

Backend: Motore di Analisi Il cuore della logica di calcolo risiede nella classe `CumulativeResilienceAnalyzer`, implementata nel file `cumulative_resilience_analyzer.py`. Questo componente del backend è responsabile di analizzare la cronologia dei backup per un dato asset.

Come illustrato nel Listato 4.10, la classe viene inizializzata con i target di RPO e una finestra di analisi. I suoi metodi calcolano il tempo trascorso dall'ultimo backup valido per determinare lo stato dell'RPO e la percentuale di successi nella finestra temporale, applicando infine le formule definite nella metodologia per restituire l' R_{score} aggiornato.

```

1 class CumulativeResilienceAnalyzer:
2     def __init__(self, rpo_target_hours, analysis_window_days):
3         self.rpo_target_seconds = rpo_target_hours * 3600

```

```

4         # ...
5
6     def process_backups(self, backup_history, current_time):
7         # ... (logica per trovare l'ultimo backup valido e contare
8         i successi/fallimenti)
9
10        # Calcolo RPO Score
11        if last_successful_backup_time:
12            seconds_since_last_backup = (current_time -
13                last_successful_backup_time).total_seconds()
14            rpo_score = max(0, 1 - (seconds_since_last_backup / (
15                self.rpo_target_seconds * 2)))
16        else:
17            rpo_score = 0
18
19        # Calcolo Backup Success Rate Score
20        if total_backups > 0:
21            bsr_score = successful_backups / total_backups
22        else:
23            bsr_score = 1.0 # Nessun backup ancora eseguito, si
24            assume positivo
25
26        # Calcolo R-Score aggregato (esempio con pesi uguali)
27        r_score = (rpo_score * 0.5) + (bsr_score * 0.5)
28
29    return {"r_score": r_score, "rpo_score": rpo_score, "
30           "bsr_score": bsr_score, ...}

```

Listing 4.10: Estratto della logica di calcolo in `cumulative_resilience_analyzer.py`.

Frontend: Visualizzazione Dinamica Il frontend, implementato nel file `resilience_dashboard.html`, si occupa di interrogare l'API del backend per visualizzare lo stato della resilienza. Utilizzando JavaScript, la dashboard effettua chiamate asincrone per ottenere l'oggetto JSON con i punteggi e aggiorna l'interfaccia utente.

Come mostrato nel Listato 4.11, la funzione `updateDashboard` riceve i dati e aggiorna dinamicamente gli elementi del DOM, come indicatori numerici per gli score e messaggi di stato che informano l'utente se l'RPO è rispettato o violato.

```

1 function updateDashboard() {
2     // Esegue la chiamata all'API per ottenere i dati sulla
3     resilience
4     fetch(`/api/resilience/summary?asset=${currentAssetId}`)
5         .then(response => response.json())
6         .then(data => {
7             // Aggiorna il valore numerico del Resilience Score
8             const rScoreElement = document.getElementById('r-score-
9               value');
10            rScoreElement.textContent = data.r_score.toFixed(3);

```

```
9      // Aggiorna gli score individuali (RPO e BSR)
10     document.getElementById('rpo-score-value').textContent
11     = data.rpo_score.toFixed(3);
12     document.getElementById('bsr-score-value').textContent
13     = data.bsr_score.toFixed(3);
14
15     // Aggiorna lo stato dell'RPO (es. "Rispettato" o "
16     // Violato")
17     const rpoStatusElement = document.getElementById('rpo-
18     status');
19     rpoStatusElement.textContent = data.rpo_status_message;
20
21     // ... (logica per aggiornare altri elementi)
22   })
23   .catch(error => console.error('Errore:', error));
24 }
25
26 // Avvia l'aggiornamento periodico
27 setInterval(updateDashboard, 5000); // Aggiorna ogni 5 secondi
```

Listing 4.11: Estratto della logica JavaScript per l'aggiornamento della dashboard.

Capitolo 5

Sperimentazione

In questo capitolo vengono descritti il protocollo sperimentale, l’ambiente di sviluppo e le metodologie implementative adottate per condurre le analisi di *Anomaly Detection*. L’obiettivo è validare e confrontare le diverse tecniche discusse nel Capitolo 4, applicandole ai dataset di riferimento introdotti nel Capitolo 3. La sperimentazione si articola in tre fasi principali, focalizzate sul dataset SNMP-MIB, ciascuna progettata per rispondere a una specifica domanda di ricerca.

È importante precisare che in questo capitolo non verranno riportati esperimenti formali relativi al dataset generato (descritto nella Sezione 3.2), in quanto l’analisi su quest’ultimo si è concentrata su una verifica empirica del funzionamento degli *health score* tramite l’implementazione di una dashboard interattiva, piuttosto che su una validazione quantitativa delle performance algoritmiche.

5.1 Setup Sperimentale e Ambiente di Sviluppo

La fase sperimentale è stata condotta in un ambiente di sviluppo basato su linguaggio **Python 3.9**, sfruttando le potenzialità offerte dall’ecosistema di librerie per il *data science* e il *machine learning*. L’analisi è stata eseguita all’interno di un ambiente interattivo **Jupyter Notebook**, che ha facilitato le operazioni di esplorazione dei dati, modellazione e visualizzazione dei risultati.

Le principali librerie utilizzate includono:

- **Pandas (vers. 1.5.3):** Per la manipolazione e l’analisi dei dati, in particolare per il caricamento del dataset SNMP-MIB e la sua trasformazione in `DataFrame` strutturati;
- **Scikit-learn (vers. 1.2.2):** È stata la libreria di riferimento per l’implementazione dei modelli di machine learning classici (Random Forest, K-Nearest Neighbors, Isolation Forest), per le tecniche di preprocessing come `StandardScaler` e `MinMaxScaler`, e per le metodologie di validazione, inclusa la `StratifiedKFold`;

- **Imbalanced-learn (vers. 0.10.1):** Utilizzata specificamente per implementare la tecnica di sovraccampionamento sintetico tramite l'algoritmo **SMOTE**, come discusso nella Sezione 4.1.2;
- **TensorFlow (vers. 2.15.0) e Keras:** Per la progettazione, l'addestramento e la valutazione del modello di Deep Learning GRU-Autoencoder;
- **Matplotlib (vers. 3.7.1) e Seaborn (vers. 0.12.2):** Per la generazione di tutti i grafici e le visualizzazioni;

L'adozione di un `random_state=42` in tutte le operazioni stocastiche (divisione dei dati, inizializzazione dei modelli) ha garantito la piena **riproducibilità** degli esperimenti.

5.1.1 Preprocessing dei Dati e Preparazione degli Esperimenti

Un passaggio fondamentale, preliminare a qualsiasi fase di modellazione, è stato il preprocessing del dataset SNMP-MIB. Come evidenziato nell'analisi del Capitolo 3, i dati presentano una notevole eterogeneità, sia nella natura del target (multi-classe) sia nella scala dei valori delle feature. Le operazioni implementate, illustrate nel Listato 5.1, hanno affrontato queste criticità.

```

1 import pandas as pd
2 from sklearn.preprocessing import StandardScaler
3
4 # 1. Caricamento e pulizia iniziale
5 df = pd.read_csv('SNMP-MIB_dataset.csv')
6 df.rename(columns={'Label': 'label'}, inplace=True)
7
8 # 2. Conversione del problema in classificazione binaria
9 # Le etichette di attacco vengono mappate a 1 (Anomalia), 'normal'
10 # a 0
11 df['label'] = df['label'].apply(lambda x: 0 if x == 'normal' else
12                                 1)
13
14 # 3. Separazione delle feature (X) e del target (y)
15 X = df.drop('label', axis=1)
16 y = df['label']
17
18 # 4. Selezione delle 8 feature del gruppo "Interface" per gli
19 # esperimenti
20 interface_features = [
21     'ifInOctets', 'ifOutOctets', 'ifInUcastPkts', 'ifInNUcastPkts',
22     'ifInDiscards', 'ifOutUcastPkts', 'ifOutNUcastPkts',
23     'ifOutDiscards'
24 ]
25 X_interface = X[interface_features]
```

```

22
23 # 5. Scalatura delle feature
24 # Necessaria per i modelli basati su distanza come KNN
25 scaler = StandardScaler()
26 X_scaled = scaler.fit_transform(X_interface)
27 X_scaled_df = pd.DataFrame(X_scaled, columns=interface_features)

```

Listing 5.1: Pipeline di preprocessing per il dataset SNMP-MIB.

In primo luogo, il problema è stato trasformato da multi-classe a binario, aggregando le sette tipologie di attacco in un'unica classe **Anomalia** (etichettata con '1'). Successivamente, sono state selezionate le 8 feature del gruppo *Interface*, in linea con lo studio di riferimento [13], e normalizzate tramite **StandardScaler** per annullare l'effetto della differente scala numerica, un requisito cruciale per algoritmi come KNN.

5.2 Esperimento 1: Confronto Supervisionato vs. Non Supervisionato

5.2.1 Obiettivo e Protocollo Sperimentale

L'obiettivo primario di questa fase è confrontare le performance dei modelli supervisionati (Random Forest, K-Nearest Neighbors) con un approccio non supervisionato (Isolation Forest). La domanda di ricerca è la seguente: "*Un modello non supervisionato può raggiungere un'efficacia comparabile a quella di modelli supervisionati per il rilevamento di anomalie nel traffico di rete, in assenza di etichette durante l'addestramento?*"

L'ipotesi è che Isolation Forest possa ottenere un F1-Score superiore a 0.85, validando il suo impiego come strumento di screening. Questa ipotesi si fonda sui risultati ottenuti nel paper di Liu, Ting e Zhou [19], dove l'algoritmo ha mostrato performance superiori o paragonabili a metodi stato dell'arte su benchmark di intrusion detection.

A seguito della trasformazione del target, emerge uno sbilanciamento atipico (12% normali vs 88% anomalie), che viola l'assunto di Isolation Forest. Per affrontare questa criticità, sono stati implementati due protocolli di valutazione:

1. **Protocollo A - Dataset Sottocampionato:** Mantenimento dei 600 campioni **normal** e sottocampionamento **stratificato** della classe **Anomalia** per raggiungere una contaminazione target del 10%;
2. **Protocollo B - Dataset Sovracampionato:** Mantenimento dei 4398 campioni **Anomalia** e generazione di campioni **normal** sintetici tramite **SMOTE** per raggiungere la medesima contaminazione del 10%;

5.2.2 Dettagli Implementativi

La valutazione dei modelli è stata orchestrata tramite `cross_validate` con `StratifiedKFold` ($K = 10$). Per Isolation Forest, il cui addestramento è non supervisionato, è stato implementato un ciclo di validazione incrociata manuale per garantire un confronto metodologicamente corretto, come illustrato nel Listato 5.2.

```

1 from sklearn.model_selection import StratifiedKFold
2 from sklearn.metrics import f1_score
3
4 scores = []
5 skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
6 iso_forest = IsolationForest(n_estimators=100,
7                               max_samples=256,
8                               contamination=0.10,
9                               random_state=42)
10
11 # Ciclo manuale che replica la logica della cross-validation
12 for train_index, test_index in skf.split(X, y):
13     X_train, X_test = X.iloc[train_index], X.iloc[test_index]
14     y_test = y.iloc[test_index]
15
16     # 1. Addestramento NON supervisionato (solo su X_train)
17     iso_forest.fit(X_train)
18
19     # 2. Predizione sul test set e mappatura output
20     y_pred = iso_forest.predict(X_test)
21     y_pred_mapped = [1 if pred == -1 else 0 for pred in y_pred]
22
23     # 3. Valutazione (usando le etichette y_test)
24     score = f1_score(y_test, y_pred_mapped, average='weighted')
25     scores.append(score)
26
27 mean_f1 = np.mean(scores)

```

Listing 5.2: Ciclo di validazione incrociata manuale per Isolation Forest.

5.3 Esperimento 2: Ottimizzazione di Isolation Forest

5.3.1 Obiettivo e Protocollo Sperimentale

Questa seconda fase si è concentrata sull'ottimizzazione dell'iperparametro `contamination`. L'obiettivo era rispondere alla domanda: *"Qual è il valore ottimale dell'iperparametro, e come viene influenzato dalla strategia di ricampionamento?"* Si ipotizza di trovare un valore di `contamination` che massimizza l'F1-Score, bilanciando Precision e Recall e ipotizza che il dataset sovraccampionato con SMOTE, preservando

tutta l'informazione anomala, porti a un F1-Score massimo superiore rispetto al sottocampionamento.

5.3.2 Dettagli Implementativi

È stato definito un range di 7 valori per `contamination`: {0.01, 0.02, 0.05, 0.10, 0.15, 0.20, 0.25}. L'ottimizzazione è stata eseguita su entrambi i dataset preparati, iterando sui valori e registrando l'F1-Score medio ottenuto dalla validazione incrociata.

5.4 Esperimento 3: Analisi Temporale con GRU-Autoencoder

5.4.1 Obiettivo e Protocollo Sperimentale

La terza fase ha testato un'architettura di Deep Learning per rispondere alla domanda: *"Un GRU-Autoencoder può ottenere performance superiori sfruttando le dipendenze temporali del dataset?"* A causa della struttura del dataset, è stata adottata la suddivisione strategica (70% training, 15% validation, 15% test) discussa nella Sezione 4.1.4.

5.4.2 Dettagli Implementativi

Il primo passo implementativo è stato la trasformazione del dataset tabellare bidimensionale in un formato sequenziale tridimensionale (`campioni`, `timesteps`, `features`), adatto all'input di una rete ricorrente. È stata creata una funzione apposita che, dato un `sequence_length` (impostato a 10), genera le sequenze sovrapposte e le relative etichette, come illustrato nel Listato 5.3.

```
1 def create_sequences(data, labels, sequence_length=10):
2     X_seq, y_seq = [], []
3     for i in range(len(data) - sequence_length + 1):
4         X_seq.append(data.iloc[i:(i + sequence_length)].values)
5         # Una sequenza e' anomala se contiene anche un solo punto
6         # anomalo
6         y_seq.append(1 if 1 in labels.iloc[i:i + sequence_length].
7                     values else 0)
7     return np.array(X_seq), np.array(y_seq)
```

Listing 5.3: Funzione per la creazione di sequenze temporali.

Successivamente, è stata definita l'architettura del GRU-Autoencoder utilizzando l'API funzionale di Keras. Il modello, la cui struttura è schematizzata nel Listato 5.4, segue il paradigma encoder-decoder discusso nel Capitolo 4. Layer di `Dropout` con un tasso del 20% sono stati inseriti per mitigare il rischio di overfitting.

```

1 from tensorflow.keras.models import Model
2 from tensorflow.keras.layers import Input, GRU, RepeatVector,
3     TimeDistributed, Dense, Dropout
4
5 def build_gru_autoencoder(timesteps, n_features):
6     # --- Encoder ---
7     input_layer = Input(shape=(timesteps, n_features))
8     encoder = GRU(64, activation='relu', return_sequences=True)(
9         input_layer)
10    encoder = GRU(32, activation='relu', return_sequences=False)(
11        encoder)
12
13    # --- Vettore Latente ---
14    latent_vector = RepeatVector(timesteps)(encoder)
15
16    # --- Decoder ---
17    decoder = GRU(32, activation='relu', return_sequences=True)(
18        latent_vector)
19    decoder = Dropout(0.2)(decoder)
20    decoder = GRU(64, activation='relu', return_sequences=True)(
21        decoder)
22    decoder = Dropout(0.2)(decoder)
23
24    # --- Output Layer ---
25    output_layer = TimeDistributed(Dense(n_features))(decoder)
26
27    model = Model(inputs=input_layer, outputs=output_layer)
28    return model
29
30
31 # Compilazione del modello
32 gru_model = build_gru_autoencoder(10, 8) # 10 timesteps, 8 features
33 gru_model.compile(optimizer='adam', loss='mae')

```

Listing 5.4: Architettura del GRU-Autoencoder implementata in Keras.

Il ciclo di validazione del modello, schematizzato nel Listato 5.5, è stato articolato in più fasi. Inizialmente, il modello è stato addestrato esclusivamente sul set di training, composto da sole sequenze normali. La scelta della funzione di loss **Mean Absolute Error (MAE)** è stata deliberata per la sua robustezza agli outlier. Successivamente, è stata calcolata la soglia di anomalia statistica basandosi sugli errori di ricostruzione del validation set, fissandola a $\mu + 3\sigma$. Questa scelta si fonda sulla **regola empirica del three-sigma**, che definisce come anomalo un errore di ricostruzione così elevato da essere statisticamente molto improbabile. Infine, tale soglia è stata utilizzata per classificare le sequenze del test set e valutare le performance finali.

```

1 # 1. Addestramento esclusivo su dati normali
2 history = gru_model.fit(X_train_normal, X_train_normal,
3                          epochs=60,

```

```
4                     batch_size=32,
5                     validation_data=(X_val_normal, X_val_normal
6 ),
7                     verbose=0)
8 # 2. Calcolo degli errori di ricostruzione sul validation set
9 reconstructions = gru_model.predict(X_val_normal)
10 val_mae_loss = np.mean(np.abs(reconstructions - X_val_normal), axis
11 = (1, 2))
12 # 3. Calcolo della soglia statistica
13 threshold = np.mean(val_mae_loss) + 3 * np.std(val_mae_loss)
14
15 # 4. Valutazione sul test set
16 test_reconstructions = gru_model.predict(X_test)
17 test_mae_loss = np.mean(np.abs(test_reconstructions - X_test), axis
18 = (1, 2))
19 # 5. Classificazione basata sulla soglia
20 y_pred = [1 if e > threshold else 0 for e in test_mae_loss]
21
22 # 6. Calcolo delle metriche di performance finali
23 from sklearn.metrics import classification_report
24 print(classification_report(y_test, y_pred))
```

Listing 5.5: Ciclo di addestramento e validazione del GRU-Autoencoder.

L'esperimento è stato condotto prima con 8 e poi con tutte le 34 feature per valutare l'impatto del contesto informativo.

Capitolo 6

Risultati

In questo capitolo vengono presentati e analizzati in dettaglio i risultati ottenuti dai tre esperimenti descritti nel Capitolo 5. L’obiettivo è fornire una valutazione quantitativa e qualitativa delle diverse metodologie di *Anomaly Detection* applicate al dataset SNMP-MIB, al fine di rispondere alle domande di ricerca formulate e validare o confutare le ipotesi iniziali.

È importante precisare che, coerentemente con il protocollo sperimentale, non verranno riportati esperimenti formali relativi al dataset generato. L’analisi su quest’ultimo si è infatti concentrata su una verifica empirica del funzionamento degli health score, i cui risultati sono presentati in modo visuale attraverso la proposta progettuale della dashboard nel Capitolo 7.

6.1 Esperimento 1

Il primo esperimento mirava a confrontare le performance dei modelli supervisionati (Random Forest, K-Nearest Neighbors), la cui efficacia su questo dataset è stata investigata in letteratura [13], con l’approccio non supervisionato (Isolation Forest). La valutazione è stata condotta secondo due protocolli distinti per gestire l’atipico sbilanciamento delle classi del dataset SNMP-MIB.

6.1.1 Risultati su Dataset Sottocampionato

Il primo protocollo ha valutato i modelli su un dataset di dimensioni ridotte (667 campioni), creato tramite sottocampionamento stratificato per ottenere una contaminazione del 10%. Le performance medie, ottenute tramite validazione incrociata 10-Fold, sono riassunte nella Figura 6.1.

Risultati

Modello	Tipo	Accuracy	Precision	Recall	F1-Score	T. Add.(s)	Mem.
Random Forest	Supervisionato	0.9791	0.9651	0.9791	0.9713	0.0911	0.0469
K-NN	Supervisionato	0.9730	0.9616	0.9730	0.9664	0.0017	0.0469
iForest	Non supervisionato	0.9745	0.8994	0.8810	0.8739	0.0651	0.0469

Figura 6.1: Performance medie dei modelli sul dataset sottocampionato.

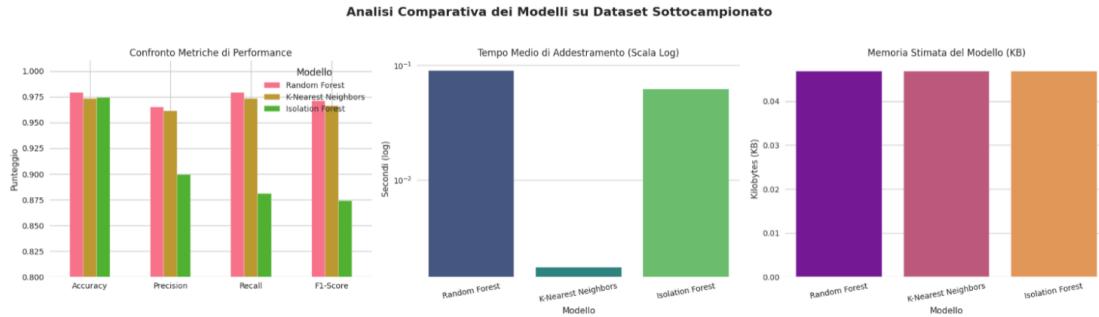


Figura 6.2: Analisi Comparativa dei Modelli su Dataset Sottocampionato.

Dai risultati emerge che i modelli supervisionati mantengono un'efficacia molto elevata anche con dati ridotti, con il Random Forest che ottiene il miglior F1-Score (0.9713). L'Isolation Forest [19], pur essendo addestrato senza etichette, raggiunge un F1-Score di **0.8739**, superando l'ipotesi iniziale di 0.85 e validando il suo impiego come strumento di screening robusto. In termini di efficienza, K-Nearest Neighbors si conferma il più rapido in addestramento, data la sua natura di *lazy learner*.

L'analisi delle curve di apprendimento (Learning Curves) ha mostrato che Random Forest e KNN hanno un errore di training e validazione molto basso e convergente, indicando un ottimo adattamento senza overfitting. La curva dell'Isolation Forest, sebbene mostri un gap leggermente più pronunciato, non indica overfitting, ma riflette l'incertezza intrinseca di un modello non supervisionato; il valore basso e stabile dell'errore di validazione ne conferma comunque la buona performance.

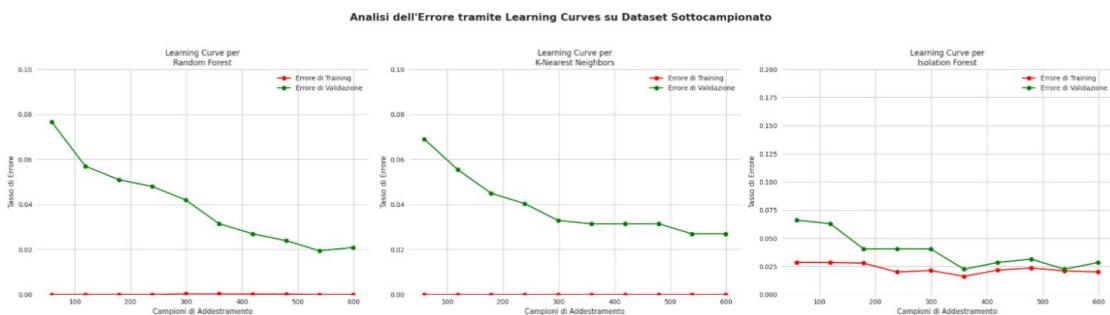


Figura 6.3: Analisi dell'Errore tramite Learning Curves su Dataset Sottocampionato.

6.1.2 Risultati su Dataset Sovracampionato (SMOTE)

Il secondo esperimento ha testato i modelli su un dataset di grandi dimensioni (circa 44.000 campioni), in cui la classe `normal` è stata bilanciata generando campioni sintetici con la tecnica SMOTE [8]. I risultati medi della validazione incrociata 10-Fold sono presentati nella Figura 6.4.

Modello	Tipo	Accuracy	Precision	Recall	F1-Score	T. Add.(s)	Mem.
Random Forest	Supervisionato	0.9999	0.9999	0.9999	0.9999	3.8559	0.0469
K-NN	Supervisionato	1	1	1	1	0.0294	0.0469
iForest	Non supervisionato	0.9748	0.8738	0.8743	0.8739	0.2299	0.0469

Figura 6.4: Performance medie dei modelli sul dataset sovraccampionato (SMOTE).

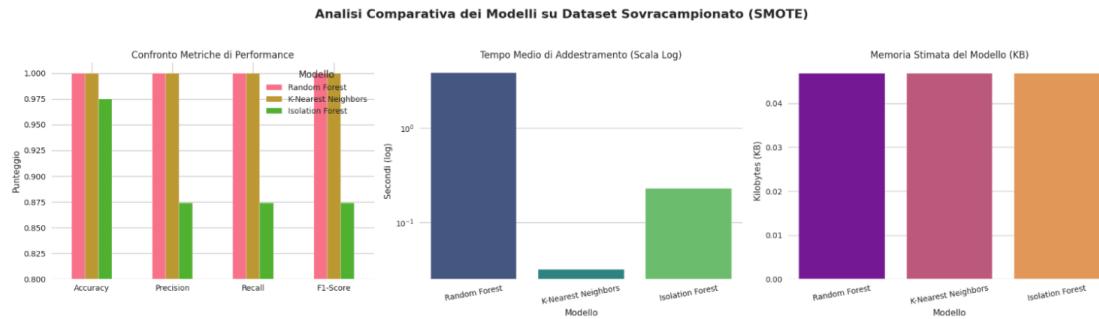


Figura 6.5: Analisi Comparativa dei Modelli su Dataset Sovracampionato (SMOTE).

In questo scenario, con un'abbondanza di dati di training, i modelli supervisionati raggiungono la perfezione, indicando che una rappresentazione densa della classe normale permette una separazione impeccabile. L'Isolation Forest conferma la sua notevole stabilità, mantenendo un F1-Score di **0.8739**, quasi identico a quello dell'esperimento precedente. Questo risultato dimostra che l'efficacia del modello non dipende dalla quantità di dati, ma dalla qualità e dalla rappresentatività degli stessi, suggerendo che l'approccio con sottocampionamento, computazionalmente meno oneroso, è sufficiente e ugualmente efficace.

6.2 Esperimento 2

L'esperimento di ottimizzazione dell'iperparametro `contamination` per Isolation Forest ha prodotto risultati significativi. Le performance delle due strategie di ricampionamento al variare di questo parametro sono riassunte e visualizzate nella Figura 6.6.

Risultati

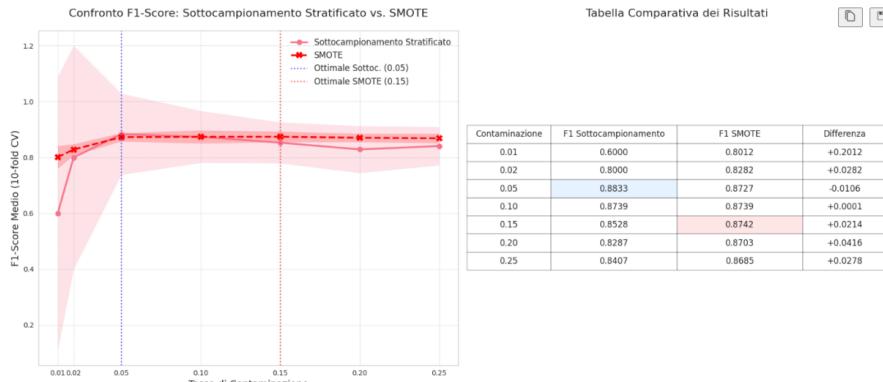


Figura 6.6: Confronto F1-Score: Sottocampionamento Stratificato vs. SMOTE. & Tabella Comparativa dei risultati dell'ottimizzazione.

I dati confermano l'esistenza di uno "sweet spot" per la **contamination**, che massimizza l'F1-Score. Il valore ottimale è risultato essere del **5% per il sottocampionamento** e del **15% per SMOTE**. Contrariamente all'ipotesi iniziale, la strategia di sottocampionamento ha ottenuto un F1-Score massimo leggermente superiore (**0.8833**) rispetto a SMOTE (0.8742). Questo suggerisce che, per un algoritmo basato sull'isolamento, una minoranza più "pulita" e nettamente separata può risultare marginalmente più vantaggiosa della creazione di dati sintetici.

6.3 Esperimento 3

La terza fase ha testato un'architettura GRU-Autoencoder per sfruttare le dipendenze temporali del dataset, seguendo un percorso iterativo che ha portato a scoperte significative sulla natura dei dati e sui requisiti dei modelli di Deep Learning, come evidenziato in recenti rassegne del settore [15].

6.3.1 Fallimento del primo tentativo (8 feature)

Il primo tentativo, utilizzando solo le 8 feature del gruppo *Interface* e una soglia di anomalia statistica ($\mu + 3\sigma$), si è rivelato fallimentare. Le performance sul test set, riassunte nella Figura 6.7, sono state scarse.

Classe	Precision	Recall	F1-Score	Support
Normale	0.05	0.98	0.10	89
Anomalia	1	0.66	0.79	4397
Accuracy			0.66	4486
Macro Avg	0.53	0.82	0.45	4486
Weighted Avg	0.98	0.66	0.78	4486

Figura 6.7: Risultati del GRU-Autoencoder con 8 feature.

Risultati

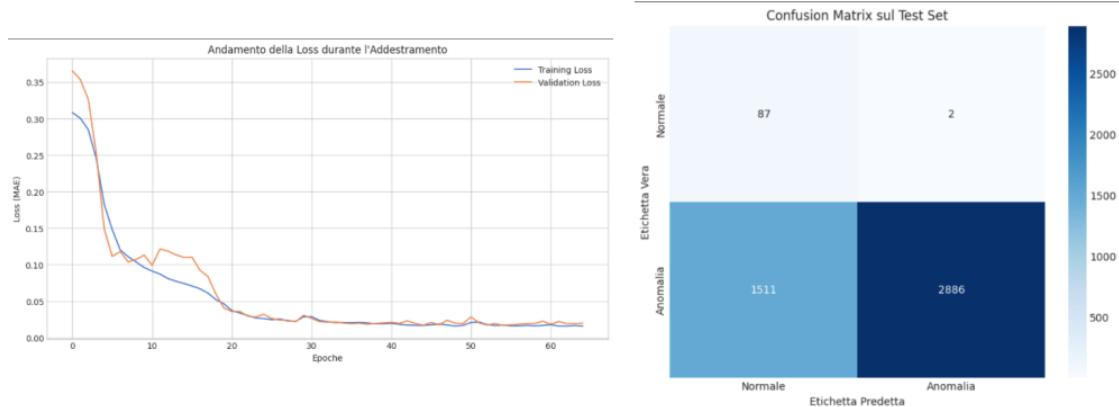


Figura 6.8: Curve di Training e Validation Loss & Matrice di Confusione del GRU-AutoEncoder.

Una Precision del 5% sulla classe "Normale" indica che il modello classificava erroneamente quasi tutte le anomalie come normali, mentre una Recall del 66% sulla classe "Anomalia" mostra che falliva nel rilevare un terzo di tutti gli attacchi. Nonostante le curve di apprendimento mostrassero un addestramento stabile, il modello non era in grado di discriminare efficacemente, suggerendo che le 8 feature, sebbene sufficienti per i modelli classici, non fornivano un contesto informativo adeguato per un modello temporale.

6.3.2 Analisi del fallimento e ipotesi risolutiva

Dopo aver escluso che il fallimento fosse legato a una scorretta configurazione degli iperparametri (complessità del modello, lunghezza delle sequenze), è emersa un'intuizione risolutiva: il problema non era la quantità dei dati normali, ma la loro **qualità informativa**. Le 8 feature del gruppo *Interface*, pur essendo indicative, non fornivano al modello il contesto necessario per costruire una rappresentazione robusta della "normalità" in una dimensione temporale.

6.3.3 Risultati con set di feature esteso (34 feature)

Sulla base di questa intuizione, il modello è stato riaddestrato utilizzando tutte le 34 feature del dataset. Le performance sono migliorate drasticamente, come mostra il confronto in Figura 6.9.

Risultati

Metrica	Modello 1 F1 (8 feat.)	Modello 2 F2 (34 feat.)	Variazione
Accuracy Complessiva	0.66	0.96	+45%
F1-Score macro Avg.	0.61	0.68	+51%
Recall (Normale)	0.98	0.70	-29%
Precision (Normale)	0.05	0.27	+440%
Recall (Anomalia)	0.66	0.96	+45%
Precision (Anomalia)	1	0.99	-1%
F1-Score (Normale)	0.10	0.39	+290%
F1-Score (Anomalia)	0.79	0.98	+24%

Figura 6.9: Confronto performance GRU-Autoencoder: 8 vs 34 feature.

Il contesto aggiuntivo fornito dalle informazioni sui protocolli IP, TCP e ICMP si è rivelato prezioso. Il miglioramento più critico è stata la **Recall per la classe "Anomalia"**, salita a **0.96**, indicando che il modello era ora in grado di identificare quasi tutti gli attacchi. Questo livello di sensibilità è stato raggiunto accettando un compromesso calcolato: una Precisione sulla classe "Normale" del 27%. Sebbene questo valore possa sembrare basso, rappresenta un enorme passo avanti (+440%) e un trade-off accettabile in un contesto di sicurezza, dove minimizzare i falsi negativi (anomalie non rilevate) è prioritario. L'F1-Score per la classe "Anomalia" ha raggiunto un eccellente **0.98**, dimostrando il potenziale latente dell'approccio Deep Learning, a condizione che disponga di un contesto informativo sufficientemente ricco. Questo risultato finale, sebbene notevolmente migliorato, non eguaglia ancora la robustezza e l'equilibrio dimostrati dai modelli classici nell'Esperimento 1 (seppur non chiaramente comparabili come risultati di esperimenti). L'analisi evidenzia una duplice natura dei requisiti per i modelli di Deep Learning in questo contesto: sebbene un set di feature esteso si sia dimostrato **necessario** per fornire un contesto informativo adeguato, i risultati sulla classe "Normale" suggeriscono che non sia ancora **sufficiente**. Per raggiungere il suo pieno potenziale, un'architettura GRU-Autoencoder richiederebbe probabilmente un volume di dati di training significativamente maggiore, così da apprendere una rappresentazione della normalità più completa e generalizzata.

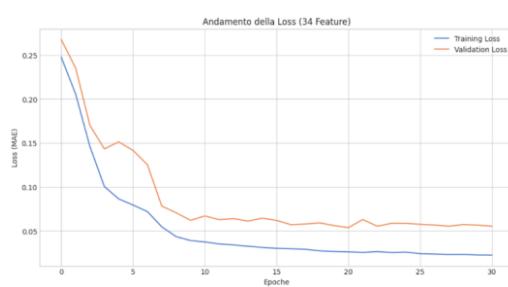


Figura 6.10: Andamento della Loss del modello finale (34 feature).

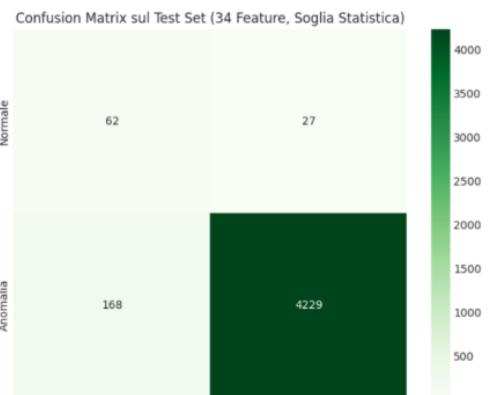


Figura 6.11: Confusion Matrix del modello finale (34 feature).

Capitolo 7

Proposta progettuale: Dashboard EPS+

Il presente capitolo rappresenta il culmine del lavoro di ricerca e sviluppo condotto in questa tesi, fungendo da ponte tra l'analisi teorica, la sperimentazione e la validazione pratica. L'obiettivo è tradurre le metodologie e i risultati discussi nei capitoli precedenti in una soluzione software tangibile.

La trattazione si articolerà in due parti principali. Inizialmente, verrà presentata una **proposta architettonica** completa per il sistema EPS+, delineando una struttura a più livelli, modulare e scalabile, progettata per rispondere alle esigenze di monitoraggio delle PMI. All'interno di questa proposta, verrà approfondito il design di una componente propedeutica fondamentale: il modulo di **Network Discovery**.

Successivamente, il capitolo si concentrerà sullo **showcase del prototipo dimostrativo** effettivamente realizzato. Attraverso una serie di schermate e descrizioni funzionali, verrà illustrato in dettaglio il funzionamento del sistema, partendo dall'integrazione con le piattaforme reali dell'infrastruttura di riferimento, **Proxmox VE** e **Acronis Cyber Protect**. Il cuore della dimostrazione sarà poi dedicato alle dashboard analitiche, dove si mostrerà l'applicazione pratica delle metodologie di calcolo per i pilastri di **Performance (P-Score)**, **Availability (A-Score)** e **Resilience (R-Score)**, e l'implementazione del modello di machine learning per il **rilevamento delle anomalie** di rete.

7.1 Architettura Generale del Sistema EPS+

Per rispondere in modo completo e scalabile alle esigenze di monitoraggio delle Piccole e Medie Imprese (PMI), è stata progettata un'architettura software a più livelli, che separa logicamente le responsabilità di raccolta, memorizzazione, analisi e presentazione dei dati. Questa visione permette di creare un sistema modulare, in cui ogni componente può essere sviluppato, manutenuto e potenziato in modo indipendente.

La Figura 7.1 illustra la visione d'insieme del sistema EPS+, evidenziando i quattro livelli fondamentali: *Data Sources*, *Storage*, *Processing* e *Presentation*. Il prototipo dimostrativo descritto in questa tesi si concentra sull'implementazione dei concetti chiave dei

livelli di *Processing* e *Presentation*, validando il cuore analitico e l’interfaccia utente del sistema. Le componenti marcate come sviluppi futuri rappresentano l’evoluzione naturale del progetto verso una soluzione completa di osservabilità.

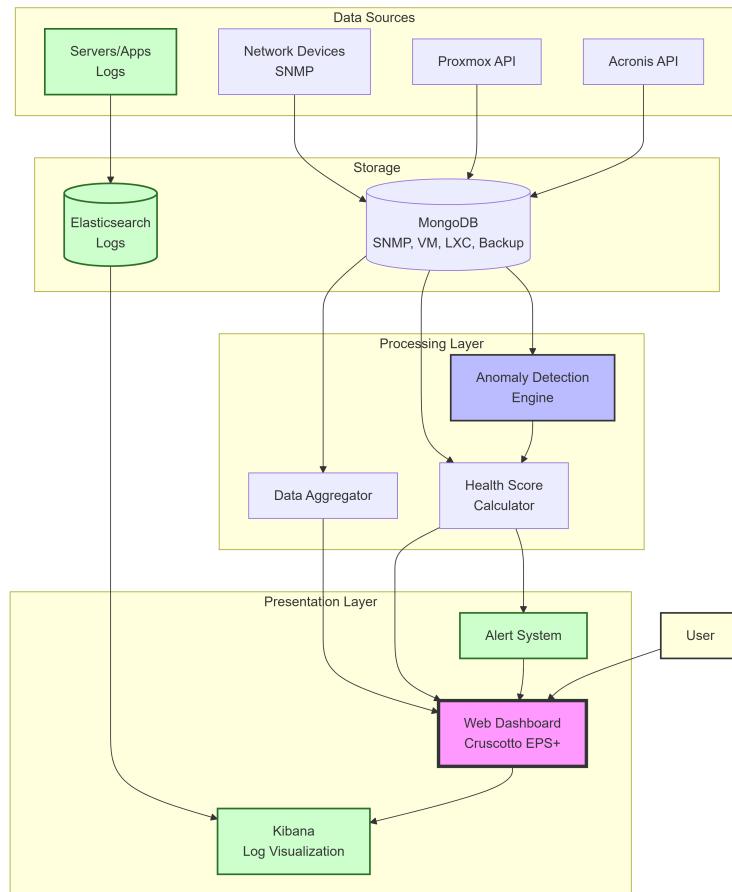


Figura 7.1: Diagramma dell’architettura completa del sistema EPS+. Le componenti in verde rappresentano gli sviluppi futuri previsti per il progetto. Trattati nel capitolo successivo.

7.1.1 Data Sources Layer

Questo livello rappresenta l’origine dei dati e include tutte le componenti dell’infrastruttura IT di una tipica PMI. Le fonti dati identificate sono:

- **Network Devices (SNMP):** Dispositivi di rete come router e switch, da cui vengono raccolte metriche di performance e stato operativo tramite il protocollo SNMP;
- **Proxmox API:** Il sistema di virtualizzazione, che espone tramite API RESTful dati critici sullo stato delle macchine virtuali (VM) e dei container (LXC), sull’utilizzo delle risorse (CPU, RAM, I/O) e sugli eventi di sistema;

- **Acronis API:** La piattaforma di backup e disaster recovery, che fornisce informazioni sullo stato dei processi di backup, sullo spazio utilizzato e sulla data dell'ultimo ripristino valido;
- **Servers/Apps Logs (Sviluppo Futuro):** I log di sistema e applicativi, una fonte dati fondamentale per analisi approfondite su errori, accessi e eventi di sicurezza;

7.1.2 Storage Layer

I dati raccolti vengono memorizzati in database ottimizzati per la loro specifica natura. L'architettura prevede una soluzione duale per massimizzare l'efficienza:

- **MongoDB:** Un database NoSQL document-oriented scelto per la sua flessibilità. È ideale per memorizzare dati strutturati e semi-strutturati come le metriche time-series provenienti da SNMP e dalle API di Proxmox e Acronis;
- **Elasticsearch (Sviluppo Futuro):** Un motore di ricerca e analisi ottimizzato per dati testuali non strutturati. Rappresenta la scelta standard per l'indicizzazione e l'analisi di grandi volumi di log;

7.1.3 Processing Layer

Questo è il cuore analitico del sistema, dove i dati grezzi vengono trasformati in informazioni significative. I due motori principali sono:

- **Anomaly Detection Engine:** Responsabile dell'analisi dei dati per il rilevamento proattivo di comportamenti anomali. Come validato nel Capitolo 6, questo modulo si basa su modelli di machine learning come Isolation Forest per l'analisi dei dati di rete;
- **Health Score Calculator:** Il componente che implementa le metodologie descritte nel Capitolo 4 per calcolare i punteggi dei singoli pilastri (Performance, Availability, Resilience) e aggregarli nell'Health Score complessivo;

7.1.4 Presentation Layer

L'ultimo livello è l'interfaccia attraverso cui l'utente interagisce con il sistema.

- **Web Dashboard (Cruscotto EPS+):** Il punto di accesso principale, progettato per fornire una visione d'insieme chiara e immediata ("single pane of glass") dello stato di salute dell'infrastruttura tramite l'Health Score aggregato e le visualizzazioni dei singoli pilastri;
- **Alert System (Sviluppo Futuro):** Un sistema di notifica multicanale (es. email, webhook) per avvisare proattivamente gli operatori in caso di anomalie critiche;
- **Kibana (Sviluppo Futuro):** Insieme a Elasticsearch, fornirebbe un'interfaccia avanzata per l'esplorazione e la visualizzazione interattiva dei log;

7.2 Modulo di Network Discovery

Un sistema di monitoraggio, per essere realmente efficace, non può operare nel vuoto; deve innanzitutto possedere una mappa accurata e costantemente aggiornata del dominio che intende proteggere. Per questa ragione, l'architettura di EPS+ è stata progettata per includere, come passo fondamentale, un **modulo di Network Discovery**. La sua funzione trascende quella di un semplice strumento di scansione: agisce come un vero e proprio sistema di onboarding, automatizzando l'inventario degli asset e semplificando drasticamente la configurazione iniziale.

Questa scelta progettuale si fonda su due principi cardine. Da un lato, risponde a un requisito fondamentale della gestione delle reti di calcolatori, secondo cui la conoscenza della topologia e la configurazione degli apparati sono il prerequisito indispensabile per qualsiasi attività di gestione e monitoraggio [25]. Dall'altro, in modo ancora più strategico, questo modulo costituisce l'implementazione tecnica della funzione ‘**Identify**’ (ID) del **NIST Cybersecurity Framework (CSF) 2.0**, la quale pone la gestione degli asset (ID.AM) come fondamento per comprendere e governare il rischio informatico [21].

È fondamentale sottolineare che, sebbene questo modulo rappresenti una componente cruciale della visione completa del progetto, la sua implementazione pratica non rientra nel perimetro del prototipo dimostrativo sviluppato in questa tesi. Il prototipo, infatti, si concentra sulla validazione delle metodologie di calcolo degli "Health Score", come verrà illustrato nelle sezioni successive.

7.2.1 Pipeline

Per massimizzare l'efficienza della scoperta e la ricchezza delle informazioni raccolte, il modulo non si affida a una singola tecnica, ma orchestra una pipeline di scansione sequenziale e logica, gestita dal componente **ScannerOrchestrator**. Ogni fase del processo è progettata per raffinare la conoscenza della rete, passando da una visione generale a un'analisi di dettaglio:

- **Auto-configurazione della Rete:** All'avvio, il sistema si auto-configura interro-gando il sistema operativo per identificare l'interfaccia di rete attiva e la sottorete di appartenenza (es. 192.168.1.0/24), astraendo la complessità dall'utente finale;
- **Fase 1: Scoperta Iniziale (ARP):** La prima fase impiega una scansione ARP parallela, una tecnica a basso livello rapida ed efficiente, per costruire una mappa iniziale di tutti gli host attivi e raggiungibili sulla rete locale;
- **Fase 2: Analisi Approfondita (Nmap):** Gli host attivi diventano il target per una scansione Nmap più approfondita. Questa fase arricchisce l'inventario con dettagli cruciali come le porte aperte, i servizi in esecuzione e, ove possibile, il sistema operativo. Un risultato chiave di questa analisi è l'identificazione dei dispositivi che espongono la porta SNMP (161/UDP), protocollo standard per la gestione di rete [25];

- **Fase 3: Raccolta Dettagliata (SNMP):** La scansione SNMP viene eseguita in modo mirato solo sui dispositivi precedentemente identificati come compatibili. Questa tecnica permette di raccogliere informazioni di alto livello, come il produttore, il modello e la descrizione del sistema, direttamente dall'apparato;
- **Fase 4: Classificazione e Report:** Infine, tutti i dati eterogenei raccolti vengono aggregati e analizzati dal **DeviceClassifier**, che assegna a ogni asset una categoria logica (es. *NetworkEquipment*, *Windows*). Il risultato è un report JSON strutturato, che costituisce l'inventario dinamico pronto per essere utilizzato dalla dashboard;

7.2.2 Componenti Chiave

L'intelligenza del modulo non risiede unicamente negli strumenti esterni che utilizza, ma nell'architettura software che li orchestra e ne interpreta i risultati. I due componenti chiave che realizzano questa logica sono lo **ScannerOrchestrator** e il **DeviceClassifier**. Il primo agisce come vero e proprio cervello del modulo, non limitandosi a eseguire comandi in sequenza; come illustrato nello snippet di codice in Listing 7.1, il suo ruolo è gestire l'intero ciclo di vita della scansione, orchestrando il flusso di dati tra le varie fasi, aggregando i risultati e invocando il motore di classificazione per garantire un processo coeso. Il secondo, il **DeviceClassifier**, si occupa invece di trasformare i dati grezzi in conoscenza contestuale. Attraverso un sistema di regole e pattern, analizza in modo multi-fattoriale le informazioni raccolte — dalle stringhe del sistema operativo alle porte aperte, fino ai dati SNMP — per assegnare a ogni dispositivo una classificazione significativa, arricchendo l'inventario e rendendolo immediatamente comprensibile.

```

1 # Esempio di flusso logico nell'Orchestrator
2 def execute_full_scan(self):
3     # Fase 1: Rilevamento della rete e caricamento configurazioni
4     self._detect_network_configuration()
5     self._load_configurations()
6
7     # Fase 2: Esecuzione della pipeline di scansione
8     arp_results = self._execute_arp_scan()
9     nmap_results = self._execute_nmap_scan(arp_results)
10    snmp_results = self._execute_snmp_scan(nmap_results)
11
12    # Fase 3: Aggregazione e classificazione
13    merged_devices = self._merge_scan_results(
14        arp_results, nmap_results, snmp_results
15    )
16    classified_devices = self._classify_devices(merged_devices)
17
18    # Fase 4: Generazione del report finale
19    scan_result = self._create_complete_scan_result(
20        classified_devices, ...
21    )
22
23    return scan_result

```

Listing

7.1:

Estratto semplificato dal metodo `execute_full_scan` dello `ScannerOrchestrator`, che mostra il flusso di orchestrazione delle scansioni.

7.2.3 Profili di Scansione

Riconoscendo che ogni contesto operativo ha esigenze diverse, il modulo è stato progettato per essere altamente flessibile. La sua configurabilità, gestita tramite file in formato **YAML**, permette di bilanciare velocità, dettaglio e impatto sulla rete. Per rendere questa flessibilità accessibile, sono stati predisposti diversi **profili di scansione** pre-configurati, ciascuno ottimizzato per uno scopo specifico. Il profilo **Fast Scan** è ideale per una rapida mappatura della rete, dando priorità alla velocità. Al contrario, il profilo **Comprehensive Scan** privilegia la completezza delle informazioni, per ottenere un inventario di massima profondità. Infine, il profilo **Production-Safe Scan** è stato progettato per avere un impatto minimo su ambienti di produzione, operando in modo più lento e discreto. Questa architettura a profili, come mostrato nell'esempio in Listing 7.2, trasforma un complesso strumento di analisi in una soluzione adattabile e sic

```
1 # SNMP Scanner Configuration
2 snmp:
3   # Versioni SNMP da provare (in ordine di preferenza)
4   versions: [2, 1]
5
6   # Community strings da utilizzare
7   communities:
8     - "public"
9     - "private"
10
11  # Timeout per le richieste in secondi
12  timeout: 10
13
14  # OID specifici da interrogare individualmente (richieste GET)
15  specific_oids:
16    - oid: "1.3.6.1.2.1.1.1.0"
17      name: "sysDescr"
18      description: "Descrizione del sistema - tipo di dispositivo e
versione software"
19    - oid: "1.3.6.1.2.1.1.5.0"
20      name: "sysName"
21      description: "Nome/hostname del sistema"
```

Listing 7.2: Estratto del file di configurazione `snmp_config.yml`, che mostra la granularità dei parametri di scansione.

7.3 Il Prototipo Dimostrativo: Funzionalità

Questa sezione illustra il funzionamento del prototipo dimostrativo, che si compone di due applicazioni web interagenti: un'applicazione principale, `pmi_dashboard`, che funge da contenitore e da interfaccia verso le piattaforme reali, e un'applicazione secondaria che interagisce con i dati generati e il dataset SNMP-MIB, `metrics_dashboard`.

7.3.1 Integrazione con Proxmox VE e Acronis Cyber Protect

In linea con il primo obiettivo definito nel **Capitolo 1**, il prototipo realizza una dashboard centralizzata per l'integrazione dei dati provenienti da fonti eterogenee. Nello specifico, sono state implementate le interfacce per le due piattaforme dell'infrastruttura di riferimento: **Proxmox VE** e **Acronis Cyber Protect**, le cui caratteristiche sono state descritte nella Sezione 1.2.

Dashboard Proxmox

La Figura 7.2 mostra la scheda dedicata all'integrazione con Proxmox VE. Questa interfaccia, alimentata da un backend che interroga le API RESTful della piattaforma, fornisce

una visione aggregata dell’ambiente di virtualizzazione. Le funzionalità implementate consentono infatti di **visualizzare i nodi** del cluster con il loro stato operativo, **elencare le macchine virtuali (VM)** e **i container (LXC)** ospitati, permettendo di eseguire operazioni elementari come accensione, spegnimento e riavvio. Inoltre, è possibile **monitorare in tempo reale le risorse** assegnate e utilizzate da ogni guest, inclusi CPU, memoria RAM e spazio su disco. Questa dashboard non solo centralizza la supervisione dell’infrastruttura di virtualizzazione, ma realizza la fase di raccolta dati che funge da base per le analisi di performance descritte nel **Capitolo 4** e implementate nel calcolo del P-Score.

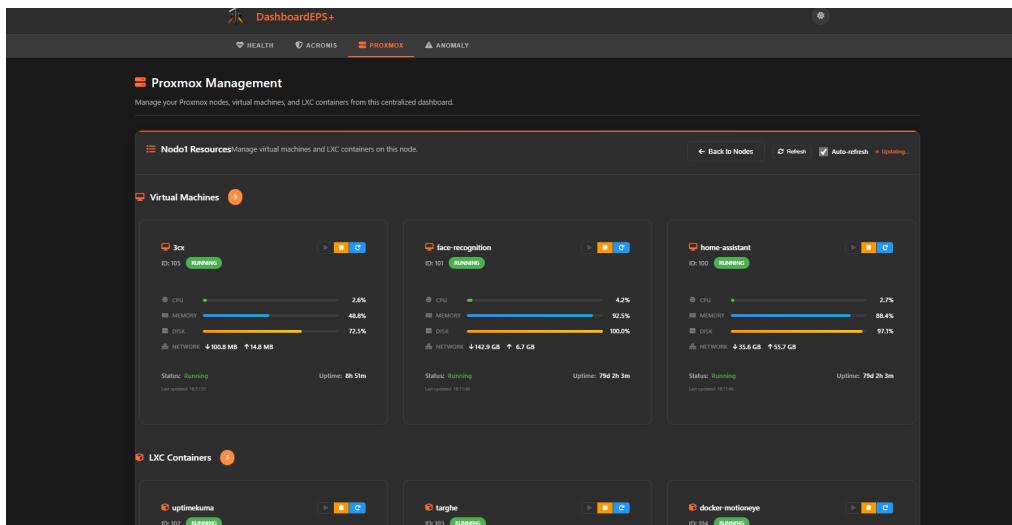


Figura 7.2: La dashboard di integrazione con Proxmox, che mostra lo stato in tempo reale dei nodi, delle macchine virtuali e dei container.

Dashboard Acronis

Analogamente, la Figura 7.3 mostra la scheda dedicata all’integrazione con la piattaforma Acronis Cyber Protect Cloud. Questa sezione del prototipo fornisce una sintesi dello stato di protezione dei dati, un elemento essenziale per la continuità operativa, come discusso nel **Capitolo 2**. Nello specifico, la dashboard permette di **visualizzare l’inventario degli agenti**, presentando un elenco dettagliato di tutti i dispositivi di protezione installati con informazioni cruciali come lo stato operativo (online/offline), il sistema operativo e l’indirizzo IP. Inoltre, consente di **monitorare i workload protetti** — ovvero le entità specifiche coperte da un piano di backup, come una singola macchina virtuale o un server fisico — indicando per ciascuno lo stato e la data dell’ultimo backup eseguito. Infine, la dashboard si occupa di **analizzare le statistiche di backup** per fornire una misura quantitativa dell’affidabilità del processo; a tal fine, vengono elaborati due grafici principali: uno che mostra la **frequenza dei backup** in una data finestra temporale e un indicatore che calcola il **Backup Success Rate**, ovvero la percentuale di backup completati con successo. Questi dati, raccolti e centralizzati, costituiscono la base informativa essenziale per il calcolo del P-Score.

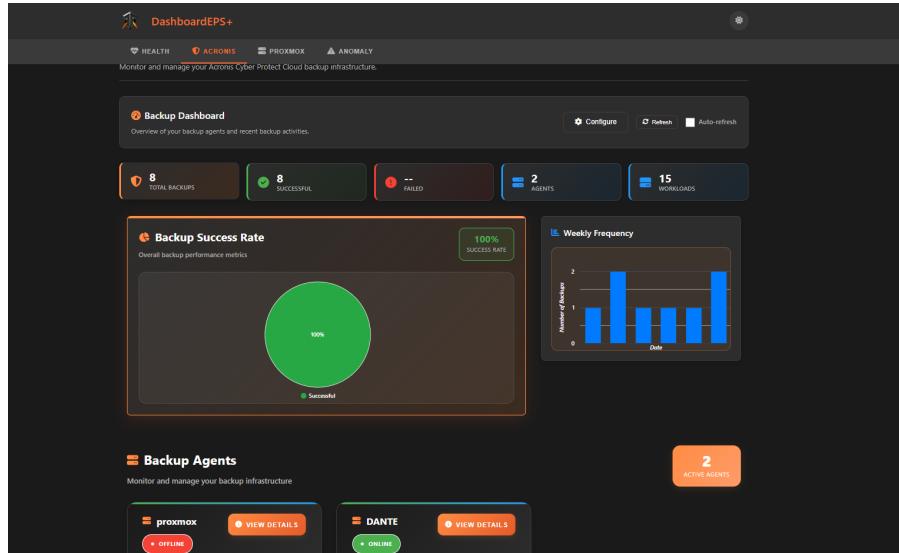


Figura 7.3: La dashboard di integrazione con Acronis, che riassume lo stato di protezione degli asset e le statistiche sui backup

7.3.2 Le Dashboard di Scoring

`metrics_dashboard` ospita le logiche di calcolo degli "Health Score". Questa applicazione realizza l'obiettivo finale del progetto: trasformare una moltitudine di metriche in un indicatore unico e intuitivo.

La Figura 7.4 mostra la dashboard principale del prototipo, che rappresenta la visione d'insieme del sistema. Al centro, l'Health Score Aggregato (H_{agg}) fornisce una valutazione sintetica dello stato di salute dell'intera infrastruttura. Questo indicatore è il risultato della formula di aggregazione ponderata, definita nel **Capitolo 4**, e viene calcolato a partire dai punteggi dei tre pilastri fondamentali — Performance, Availability e Resilience — anch'essi visibili nella schermata.

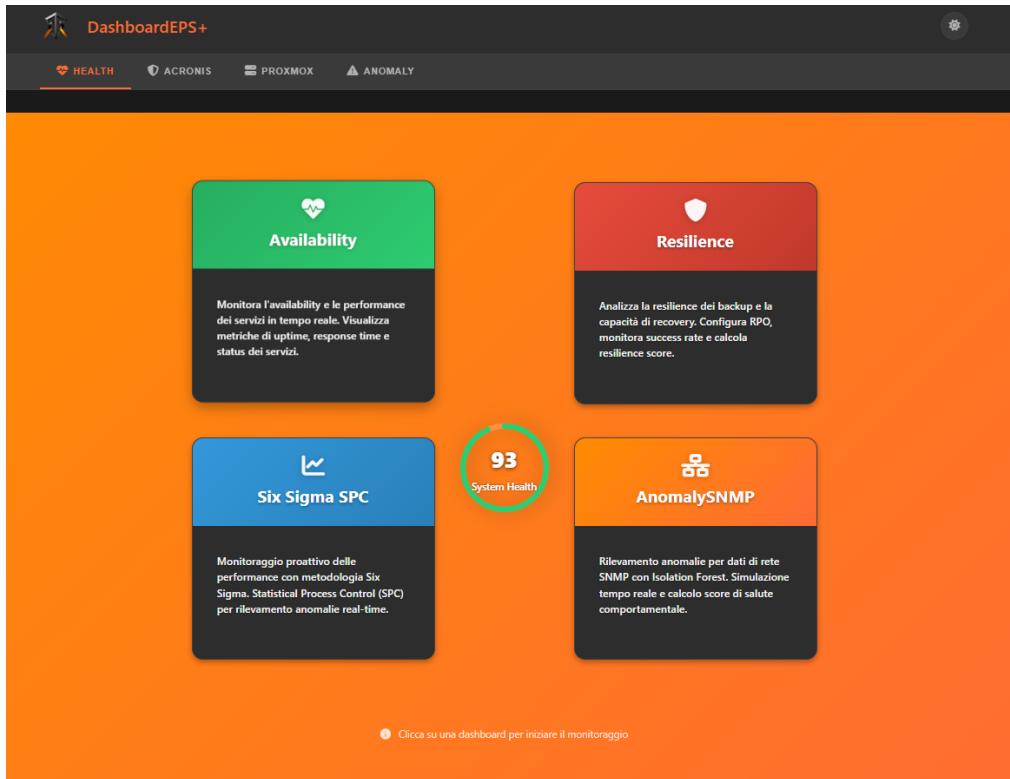


Figura 7.4: Il cruscotto principale dell'applicazione `metrics_dashboard`. La schermata mostra la visione d'insieme con l'Health Score Aggregato (H_{agg}).

*È fondamentale ricordare che, per garantire la testabilità e la riproducibilità della validazione, queste dashboard operano su un flusso di dati proveniente da un **dataset sintetico**, la cui metodologia di generazione e giustificazione sono state ampiamente discusse nel **Capitolo 3**. Inoltre, si evidenzia che i calcoli degli score si basano su un sistema di pesi configurabile: sia le singole metriche (come CPU, RAM e I/O Wait per il P-Score) sia la criticità complessiva di una macchina possono essere ponderate per riflettere accuratamente il loro impatto reale sull'infrastruttura, come definito nel **Capitolo 4**. Lo scopo di questa sezione è quindi dimostrare la correttezza logica e l'efficacia visiva del motore analitico in questi scenari controllati e configurabili.*

Monitoraggio della Performance (P-Score)

Come approfondito nella metodologia del **Capitolo 4**, il P-Score si basa sul Controllo Statistico di Processo (SPC) per valutare la stabilità delle performance di sistema. Per illustrare il funzionamento di questo pilastro, viene preso in esame il monitoraggio di un asset specifico, il WebServer-01.

Il calcolo del P-Score per una singola macchina è una **media pesata** degli score ottenuti dalle singole metriche (CPU, RAM e I/O Wait), permettendo di personalizzare il monitoraggio in base al ruolo di ogni macchina, come definito nella Sezione 4.2.1.

Scenario 1: Sistema in Stato di Controllo Il primo scenario illustra il WebServer-01 in condizioni operative normali. Come mostra la Figura 7.5, tutte le metriche fluttuano in modo casuale all'interno dei rispettivi "corridoi di normalità", definiti dai limiti di controllo (UCL/LCL). Poiché ogni singola metrica ottiene uno score di 1.0, il P-Score aggregato della macchina è anch'esso ottimale (1.0).

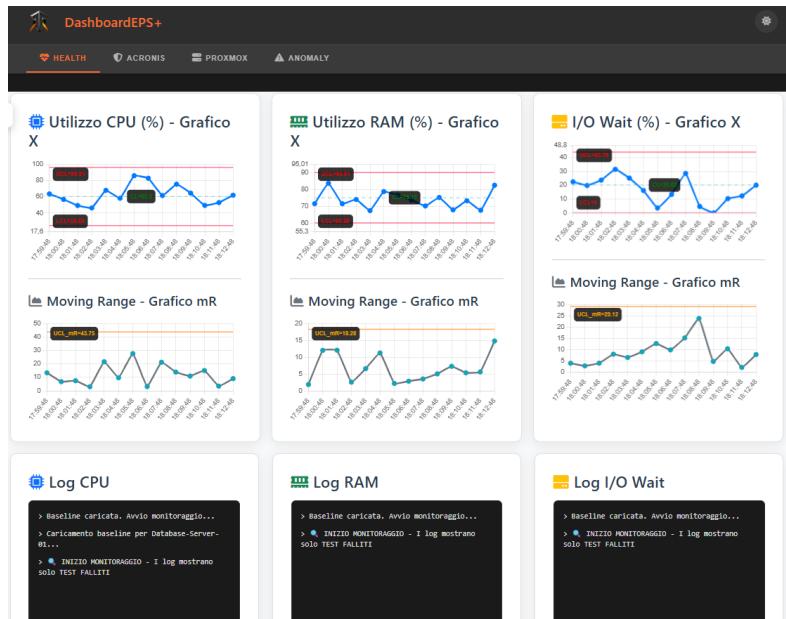


Figura 7.5: La dashboard del WebServer-01 in stato di controllo. Tutte le metriche operano all'interno dei limiti statistici, risultando in un P-Score ottimale di 1.0.

Scenario 2: Rilevamento di un'Anomalia Sistemica Il secondo scenario, illustrato nella Figura 7.6, dimostra la reattività del sistema a un cambiamento nel comportamento di una risorsa. In questo caso, viene identificato uno *shift* nella metrica dell'I/O Wait. Sebbene questa metrica abbia un peso inferiore rispetto a CPU e RAM (pari al 25%), il suo degrado è sufficiente a impattare lo score complessivo della macchina, che scende a 0.825.

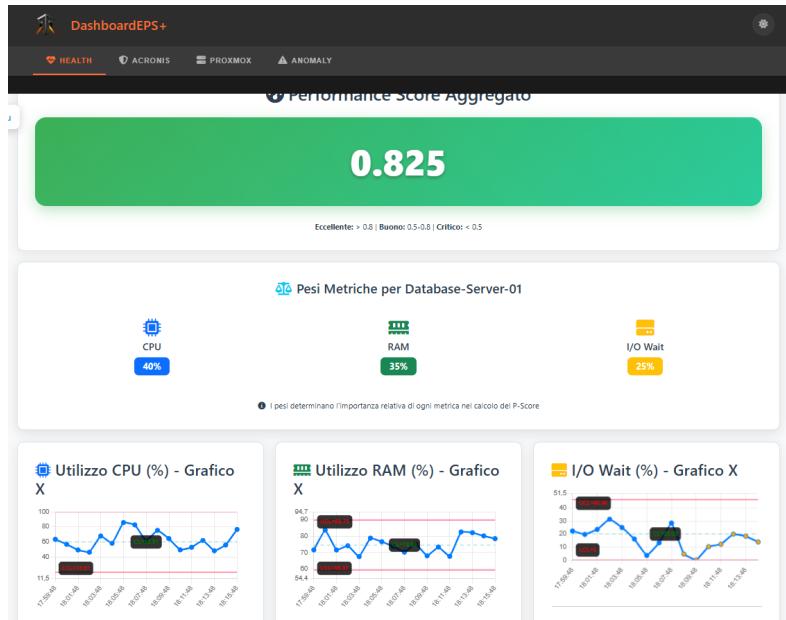


Figura 7.6: Rilevamento di un'anomalia sul WebServer-01. Uno shift nella metrica della I/O wait causa un crollo del P-Score complessivo a 0.825.

Monitoraggio dell'Availability (A-Score)

Il secondo pilastro del sistema, l'Availability, misura la disponibilità dei servizi critici. Come approfondito nella metodologia del **Capitolo 4**, l'approccio si ispira ai principi della *Site Reliability Engineering* (SRE), focalizzandosi sulla gestione proattiva dell'**Error Budget**. L'A-Score, quindi, non è una semplice misura di uptime, ma un indicatore di quanto "budget di errore" (ovvero downtime tollerabile) rimane prima di violare gli obiettivi di servizio (SLO).

Per questa implementazione, la finestra di calcolo dell'Error Budget è stata impostata su base **giornaliera (24 ore)**, fornendo un indicatore di salute operativa rilevante per la gestione quotidiana.

La Figura 7.7 mostra la dashboard di questo pilastro mentre monitora vari assets. L'immagine cattura un momento in cui molti dei servizi ha violato il loro SLO, dimostrando l'efficacia della dashboard nell'evidenziare immediatamente il problema. La dashboard mostra chiaramente il divario tra l'obiettivo di disponibilità prefissato (**Max Pool Falliti Attesi: es. 5**) e la misurazione effettiva nelle ultime 24 ore (**Pool Falliti: 3**). Ad alcuni servizi poiché l'Error Budget giornaliero è stato completamente consumato (andando in negativo), il sistema calcola correttamente un **A-Score di 0%**, segnalandola violazione dello SLO. Aggiornando la metrica aggregata di conseguenza.

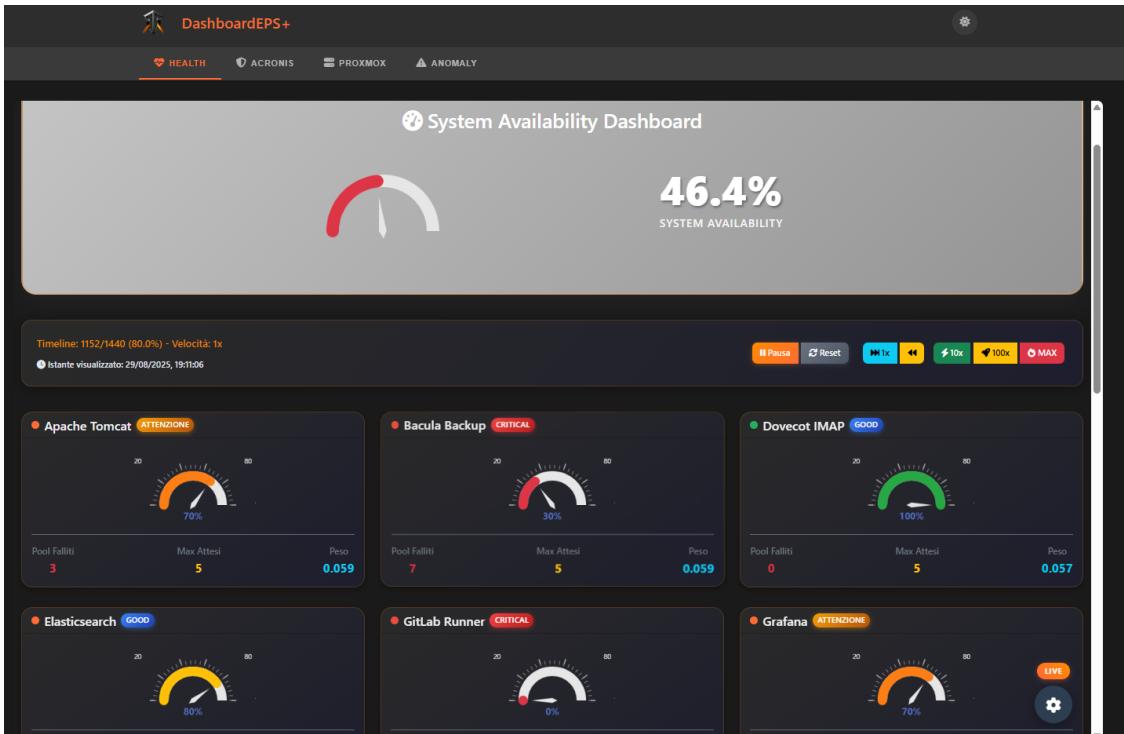


Figura 7.7: Dashboard di Availability per vari assets

L’immagine mostra uno scenario di violazione dello SLO (GitLab Runner): il downtime accumulato ha superato l>Error Budget giornaliero consentito, portando l’A-Score di quel singolo a 0% e modificando l’A-Score globale.

Monitoraggio della Resilience (R-Score)

Il terzo pilastro analitico, la Resilience, quantifica la capacità dell’infrastruttura di riprendersi da un incidente. Come definito nel **Capitolo 4**, la metodologia si concentra sull’analisi delle performance del sistema di backup. L’R-Score, in particolare, aggrega due metriche fondamentali: la conformità con l’RPO (Recovery Point Objective) e il tasso di successo dei backup (Backup Success Rate).

Per illustrare l’efficacia della dashboard, vengono presentati due momenti di una simulazione.

Scenario 1: Stato di Resilienza Ottimale La Figura 7.8 mostra la dashboard in una situazione ideale. Per ogni asset monitorato, viene visualizzato il timestamp dell’ultimo backup, confrontato con l’RPO Target configurato (24 ore). Poiché l’ultimo backup è molto recente, l’RPO è rispettato (“OK”), e sia lo score del singolo asset sia l’R-Score aggregato si attestano su un valore ottimale di 1.0.

Proposta progettuale: Dashboard EPS+

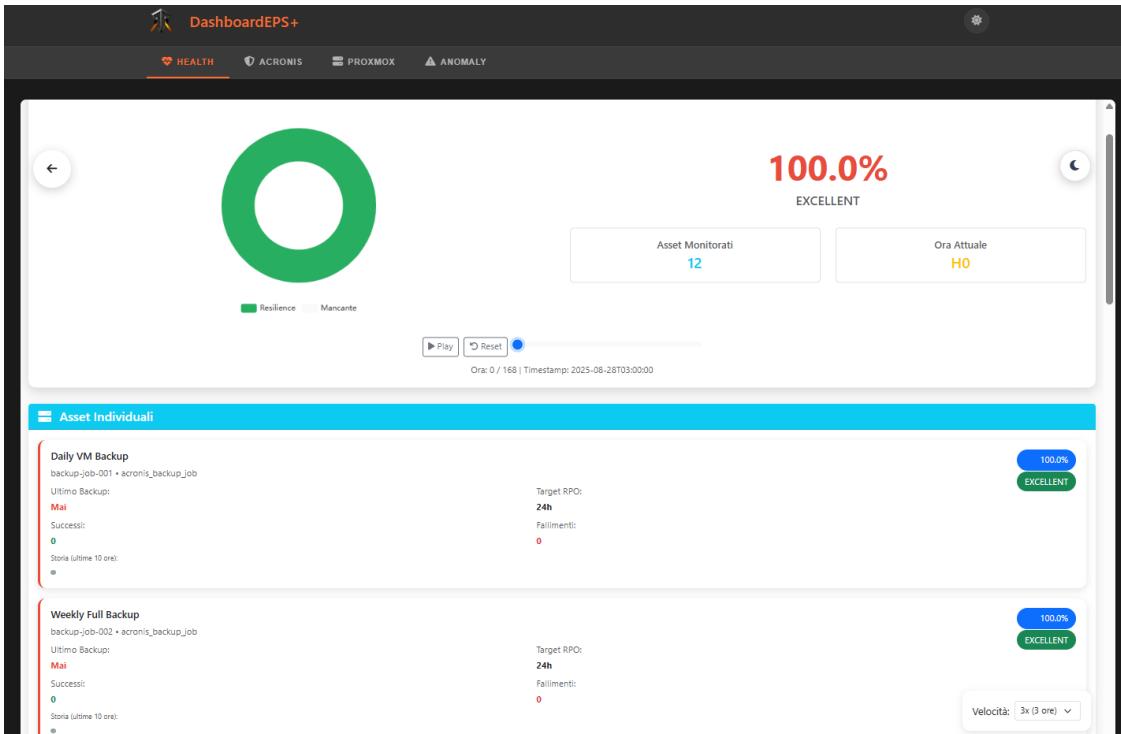


Figura 7.8: La dashboard di Resilience in condizioni ottimali. RPO è rispettato e l'R-Score aggregato è 1.0.

Scenario 2: Degrado dello Score La Figura 7.9 mostra la stessa dashboard dopo aver simulato il passare di 56 ore con conseguenti nuovi backup effettuati correttamente e falliti.

La dashboard reagisce mostrando il degrado progressivo: con il passare del tempo, lo stato dell'RPO peggiora, causando una discesa dello score individuale calcolato in base al tempo trascorso dall'ultimo backup e al numero di successi. Di conseguenza, anche l'R-Score aggregato dell'intera infrastruttura diminuisce, riflettendo le performance dei singoli asset.

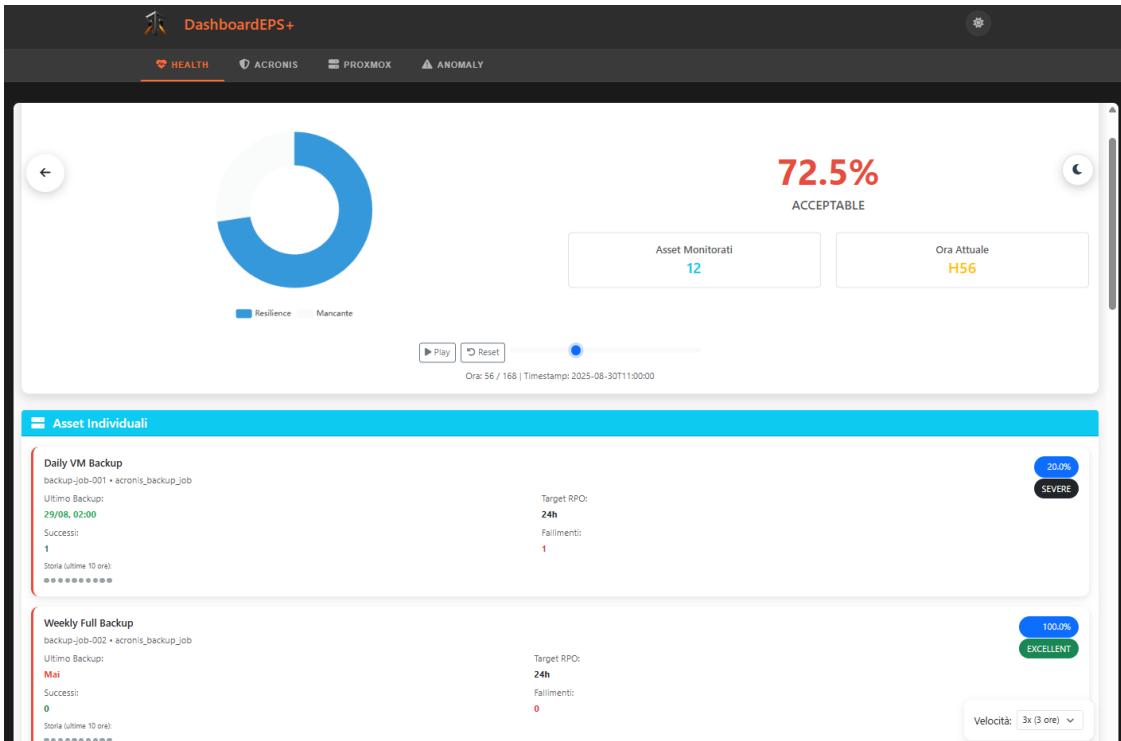


Figura 7.9: La dashboard di Resilience dopo 56 ore. Alcuni RPO violati e backup falliti.

Rilevamento Anomalie di Rete

Infine, il prototipo include una dashboard per dimostrare l'applicazione pratica di un modello di machine learning per l'analisi del traffico di rete, come validato sperimentalmente nel **Capitolo 6**. A differenza del monitoraggio SPC, che analizza le metriche singolarmente, questo approccio impiega un modello multivariato e non supervisionato per identificare pattern complessi nel comportamento aggregato di un dispositivo di rete.

La Figura 7.10 mostra l'interfaccia dedicata, il cui backend applica un modello *Isolation Forest* pre-addestrato a un flusso di dati SNMP (simulato, come da **Capitolo 3**). La dashboard è stata progettata per fornire una visione completa, che include non solo l'esito dell'analisi, ma anche una valutazione della performance del modello stesso. Un grande **indicatore di stato aggregato** (gauge) domina la schermata, fornendo la classificazione binaria in tempo reale del comportamento del dispositivo — "**Comportamento Normale**" o "**Anomalia Rilevata**" — per comunicare a colpo d'occhio la presenza di una potenziale minaccia. Al di sotto, un **grafico delle predizioni del modello** visualizza la sequenza di output dell'algoritmo nel tempo (0 per normale, 1 per anomalia), permettendo all'operatore di osservare la frequenza e la durata delle anomalie rilevate. A completare la vista, un pannello di **informazioni riassuntive** fornisce metriche sull'affidabilità del modello, come l'**accuratezza (Accuracy)** calcolata fino a quel momento e un confronto tra la predizione corrente e la *ground truth* del dataset.

Proposta progettuale: Dashboard EPS+

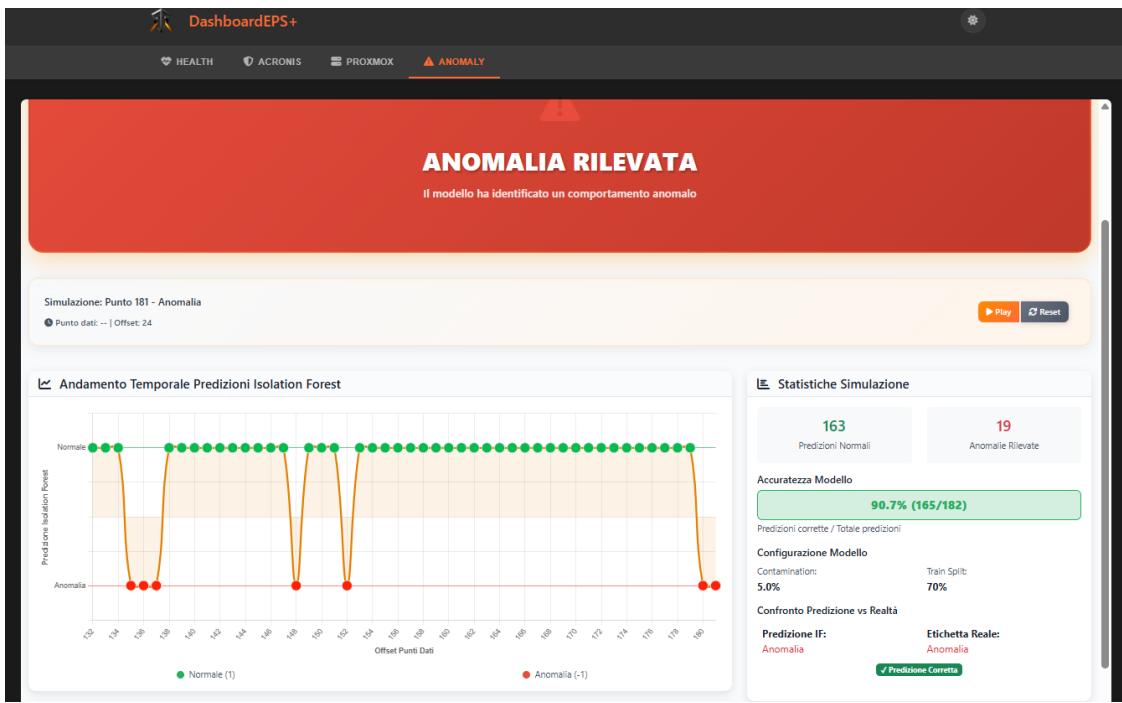


Figura 7.10: Dashboard per il rilevamento di anomalie di rete. Il sistema visualizza l'esito della classificazione, l'andamento storico delle predizioni del modello e la sua accuratezza in tempo reale.

Capitolo 8

Conclusioni e Sviluppi Futuri

8.1 Conclusioni

Il presente lavoro di tesi nasce dalla concreta esigenza industriale della società Evolumia, impegnata nella fornitura di servizi IT a Piccole e Medie Imprese, di evolvere i propri sistemi di monitoraggio da un approccio reattivo a uno proattivo. La sfida principale consisteva nel superare i limiti dei tradizionali sistemi basati su soglie statiche, spesso inadeguati a intercettare anomalie complesse e a prevenire disservizi in infrastrutture eterogenee.

L'obiettivo di questa ricerca è stato quindi quello di esplorare e validare l'efficacia di metodologie di *Anomaly Detection* basate sul Machine Learning in questo specifico contesto. A tal fine, è stata proposta un'architettura concettuale, denominata EPS+, e sono state condotte tre fasi sperimentali per identificare gli algoritmi più adatti a operare in scenari caratterizzati da dati limitati.

L'analisi sperimentale ha fornito risposte chiare alle domande di ricerca. In un contesto privo di dati reali etichettati, la priorità era identificare soluzioni di immediata applicazione. I risultati hanno validato l'approccio non supervisionato di Isolation Forest (Esperimento 1 e 2), dimostrando la sua capacità di raggiungere performance robuste (F1-Score > 0.88) senza la necessità di un training preventivo. Questo lo rende uno strumento di screening ideale e immediatamente impiegabile. Allo stesso modo, l'uso di tecniche di Controllo Statistico di Processo (SPC) per la definizione degli Health Score si è rivelato un metodo efficace per ottenere una visione quantitativa dello stato di salute del sistema fin dal primo giorno.

L'indagine sui modelli di Deep Learning (Esperimento 3) ha offerto le intuizioni più profonde. Il fallimento iniziale del GRU-Autoencoder e il suo successivo miglioramento hanno dimostrato un principio cruciale: per questi modelli, la **qualità informativa** del contesto è un prerequisito ancora più stringente della quantità di campioni. Tuttavia, le performance non ancora perfette hanno evidenziato che un'architettura così complessa, per raggiungere il suo pieno potenziale, necessita di un volume di dati significativamente maggiore per costruire una rappresentazione della normalità veramente robusta.

In conclusione, il contributo principale di questa tesi risiede nell'aver definito e validato

una metodologia pragmatica per l'introduzione di approcci basati sui dati nel monitoraggio IT per le PMI. Si è dimostrato che algoritmi efficienti e immediatamente deployabili, come Isolation Forest, rappresentano la soluzione più equilibrata per avviare un monitoraggio proattivo. Al contempo, l'architettura EPS+ propone una soluzione concettuale che mira a **superare i limiti delle tecnologie esistenti**, precedentemente analizzati: dalla rigidità dei sistemi basati su soglie statiche alla frammentazione degli strumenti tradizionali. Proponendo una visione unificata e centralizzata dello stato di salute aggregato dei servizi, il prototipo di dashboard sviluppato traduce questi risultati in una proposta tangibile, fornendo a Evolumia e ad altre realtà simili una solida base per la realizzazione di sistemi di monitoraggio di nuova generazione.

Il prototipo concettuale di EPS+ è attualmente in una fase di ingegnerizzazione e raffinamento. L'obiettivo a breve termine è il suo deployment controllato su un sottoinsieme di infrastrutture reali, al fine di raccogliere dati sul campo e valutarne le performance in un contesto operativo.

8.2 Sviluppi Futuri

Il lavoro svolto ha posto solide fondamenta per il sistema EPS+, ma la sua architettura necessita di migliorie. Di seguito vengono presentati i percorsi evolutivi, molti dei quali già delineati nel Capitolo 7.

8.2.1 Completamento Architetturale e Integrazione dei Moduli

La prima fase di sviluppo si concentra sul completamento dei componenti architetturali già previsti, per rendere il sistema pienamente operativo e automatizzato. Il primo passo consiste nell'**integrazione e nel raffinamento del Modulo di Network Discovery** già sviluppato. Basato su protocolli standard come SNMP e Nmap, questo modulo è cruciale per automatizzare la fase di *asset management* e garantire che l'inventario dell'infrastruttura sia sempre aggiornato. Successivamente, si procederà con l'**implementazione della Piattaforma di Gestione Log** attraverso l'adozione dell'Elastic Stack (**Elasticsearch** e **Kibana**), che abiliterà la raccolta centralizzata dei dati non strutturati per l'analisi forense degli eventi. Infine, l'**attivazione del Sistema di Alerting** completerà la transizione verso un monitoraggio proattivo, integrandosi con i motori di analisi per inviare **notifiche multicanale** (email, webhook per Teams/Slack) al rilevamento di anomalie critiche o al calo dell'Health Score sotto una soglia predefinita.

8.2.2 Potenziamento delle Capacità Analitiche

Parallelamente al completamento dell'architettura, la ricerca e sviluppo si concentrerà sull'evoluzione delle capacità analitiche del sistema. Un primo passo consiste nell'ottimizzazione degli iperparametri dei modelli già validati, come Isolation Forest. Sebbene i risultati attuali siano robusti, l'impiego di tecniche sistematiche come la **Grid Search**

o la **Randomized Search** potrebbe portare a un ulteriore miglioramento delle performance, identificando la combinazione di parametri ottimale per massimizzare l’F1-Score nel contesto specifico dei dati di produzione. Successivamente, si procederà lungo due direttive principali.

La prima riguarda l’**Anomaly Detection sui Log**. Andando oltre la semplice raccolta, lo sviluppo più innovativo consiste nell’applicare tecniche di *Anomaly Detection* direttamente sui dati testuali, poiché molte anomalie, specialmente quelle legate a bug applicativi o minacce alla sicurezza, lasciano tracce nei log non immediatamente visibili nelle metriche di performance. A tal fine, si potranno utilizzare modelli di Deep Learning basati su architetture come le LSTM (Long Short-Term Memory) per apprendere le sequenze normali dei log e identificare pattern anomali, seguendo approcci influenti in letteratura quale DeepLog [9].

Il secondo percorso di potenziamento si focalizza sul **raffinamento e la misurazione dinamica del Resilience Score**. Partendo dall’attuale R-Score, basato sulla disponibilità dei backup e sull’RPO, sono previsti due miglioramenti per ottenere una misura più olistica. Il primo consiste nell’integrare i dati relativi ai **backup a livello di hypervisor** (es. snapshot di **Proxmox VE**). Il secondo, più ambizioso, è lo sviluppo di un modulo per la **misurazione del Recovery Time Objective (RTO)**, da integrare come terza componente ponderata nella metrica aggregata. Poiché l’RTO non è una metrica statica ma il risultato di un processo, la sua misurazione richiede l’implementazione di un sistema di **testing automatico dei piani di ripristino**. Tale sistema eseguirà periodicamente il restore dei backup in un ambiente isolato (sandbox), validando l’integrità dei servizi e misurando il tempo effettivo di recupero. Questo approccio trasforma la resilienza da una stima teorica a un valore empirico e verificabile, come raccomandato dalle moderne strategie di business continuity [27].

8.2.3 Verso un Sistema Multimodale e Interpretabile

Una direzione di ricerca più avanzata consiste nell’evoluzione verso un **sistema multimodale**, capace di fondere flussi di dati eterogenei – come metriche e log – in una **rapresentazione latente unificata**. Sfruttando architetture di *representation learning* come i *Variational Autoencoder* (VAE), sarebbe possibile identificare anomalie complesse che emergono solo dalla correlazione incrociata di segnali deboli, altrimenti invisibili. Tuttavia, la natura di "scatola nera" di tali modelli richiede un potenziamento fondamentale: l’integrazione di tecniche di **Explainable AI (XAI)**. Lo sviluppo di un livello di spiegabilità consentirebbe di analizzare quali feature di input abbiano causato l’anomalia, trasformando un semplice rilevamento in una diagnosi interpretabile e immediatamente utilizzabile dall’operatore. Questo approccio integrato, che unisce rilevamento multimodale e interpretabilità, rappresenta la frontiera della ricerca in ambito AIOps [20, 30] e porterebbe EPS+ a un livello di proattività significativamente superiore.

Bibliografia

- [1] Ben Adlam, Behnam Ghorbani, Jasper Tyre, Rotem Mulayoff, Jiri Hron, Erfan Amid, Cagan Anil, David G Belanger, and Alon Brutzkus. The empirical analysis of the bias-variance tradeoff in machine learning. In *International Conference on Learning Representations (ICLR)*, 2022.
- [2] Mouhammd Al-Kasassbeh, Ghazi Al-Naymat, and Eshraq Al-Hawari. Towards generating realistic snmp-mib dataset for network anomaly detection. *International Journal of Computer Science and Information Security*, 14(9):684, 2016.
- [3] Otis Alexander, Jake T. Long, and Drew A. Civello. Finding the Needle in the Haystack: ATT&CK for ICS. The MITRE Corporation, 2021. Available online.
- [4] Muhammad Ali, Jamshaid Ahmad, M. Maqsood, J. A. Khan, G. Ali, and M. U. Rehman. Evaluation metrics in learning systems: A survey. *Preprints.org*, 2024. DOI: 10.20944/preprints202407.1264.v1.
- [5] AXELOS. *ITIL Foundation: ITIL 4 Edition*. TSO (The Stationery Office), 2019.
- [6] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy, editors. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, Inc., Sebastopol, CA, USA, 2016.
- [7] Danilo Caivano. Six sigma for software, November 2013. Presentation held in Bari, Project Management Institute - Southern Italy Chapter.
- [8] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.
- [9] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 1285–1298, 2017.
- [10] Zubaida Fayyaz, Muhammad Arslan, Vinod Kumar GR, and Ki-Hyun Kim. A comprehensive review of cross-validation techniques in machine learning. *IJSAT*, 2025.
- [11] Alessandro Finamore, Marco Mellia, Stathis Zarafutsis, and Michalis Kourtis. The good, the bad, and the kpis: how to combine performance metrics to better capture underperforming sectors in mobile networks. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 1281–1292. IEEE, 2017. This study justifies the methodology of combining multiple KPIs into a single "hotspot score" to classify performance and quickly identify problematic areas.

- [12] Brendan Gregg. *Systems Performance*. Addison-Wesley Professional, 2nd edition, 2020. An authoritative reference for systems performance analysis, identifying CPU, memory, and I/O as the fundamental subsystems to monitor.
- [13] Ahmed Hambouz, Ghazi Al-Naymat, and Mouhammd Alkasassbeh. Network attacks anomaly detection using snmp mib interface parameters. *arXiv preprint*, 2019. Preprint.
- [14] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science & Business Media, 2nd edition, 2009.
- [15] Haoqi Huang, Ping Wang, Jianhua Pei, Jiacheng Wang, Shahen Alexanian, and Dusit Niyato. Deep learning advancements in anomaly detection: A comprehensive survey. *arXiv preprint arXiv:2503.13195*, 2025.
- [16] Mohammed A Kadhim. Computational complexity and its influence on predictive capabilities of machine learning models for concrete mix design. *Materials*, 17(10):2209, 2024.
- [17] Eamonn Keogh and Srinivas Kasetty. Time series data mining. *Data mining and knowledge discovery*, 7:353–363, 2003.
- [18] Averill M Law. *Simulation modeling and analysis*. McGraw-hill, 2014.
- [19] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE, 2008.
- [20] Daniele Meli. Explainable online unsupervised anomaly detection for cyber-physical systems via causal discovery from time series. *arXiv preprint arXiv:2404.09871*, 2024.
- [21] National Institute of Standards and Technology. The nist cybersecurity framework (csf) 2.0. Technical Report NIST CSWP 29, National Institute of Standards and Technology, Gaithersburg, MD, 2024.
- [22] Soumya Paul, Kelley Dempsey, Ron Ross, and Mark Riddle. Information security continuous monitoring (iscm) for federal information systems and organizations. Technical Report SP 800-137, National Institute of Standards and Technology, Gaithersburg, MD, 2011.
- [23] Anza Shakeel. *Unsupervised Automatic Detection Of Transient Phenomena In InSAR Time-Series using Machine Learning*. PhD thesis, Durham University, 2022.
- [24] Ghaith Soltana, Mohamed Kaaniche, and Ra'ed Al-Dwairi. Synthetic data generation for statistical testing. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 595–605. IEEE, 2017.
- [25] William Stallings. *Data and Computer Communications*. Pearson Education, 8th edition, 2007.
- [26] James PG Sterbenz, Egemen K Çetinkaya, Mahmood A Hameed, Abdul Jabbar, Lin Qian, and Justin P Rohrer. Evaluation of network resilience, survivability, and disruption tolerance: Analysis, topology generation, simulation, and experimentation. *Telecommunication Systems*, 48:1–45, 2011.
- [27] Marianne Swanson, Pauline Bowen, Amy Wohl Phillips, Don Gallup, and David Lynes. Contingency planning guide for federal information systems. Technical Report SP 800-34 Rev. 1, National Institute of Standards and Technology, Gaithersburg, MD, 2010.

- [28] The MITRE Corporation. ATT&CK for Industrial Control Systems. Technical report, The MITRE Corporation, 2020. Available online.
- [29] Kishor S. Trivedi and Andrea Bobbio. *Reliability and Availability Engineering: Modeling, Analysis, and Applications*. Cambridge University Press, Cambridge, 2017.
- [30] Theofilos Vasileiadis, Yannis Plessas, Georgios Pachos, Georgios Koulinas, Dimitris Karvelis, and Theofanis Makris. Ma-vae: Multi-head attention-based variational autoencoder approach for anomaly detection in multivariate time-series applied to automotive endurance powertrain testing. *arXiv preprint arXiv:2309.02253*, 2023.

Ringraziamenti

Desidero esprimere la mia più profonda gratitudine a tutte le persone che hanno contribuito alla realizzazione di questo lavoro.

Ringrazio il mio relatore, Prof. **Donato Impedovo**, per aver permesso ciecamente l'inizio di questo percorso e per avermi supportato in un momento critico. A lui va il mio sincero apprezzamento per la sua fiducia e la mia stima. Ringrazio il mio correlatore, Dott. **Danilo Caivano**, per avermi affiancato nella modellazione e nella progettazione dell'idea di tesi, seppur con qualche difficoltà, e il Dott. **Vincenzo Gattulli** per il suo prezioso contributo nella stesura.

Questo traguardo non sarebbe stato possibile senza l'amore e il supporto dei miei genitori, **Michele** e **Antonella**. Hanno creduto in me sin dal primo giorno, sostenendomi in ogni passo di questo lungo percorso universitario, anche quando c'è stato il dubbio che non sarei arrivato in fondo. Nonostante le difficoltà che hanno dovuto affrontare, non mi hanno mai fatto mancare nulla. Questo lavoro e questo risultato sono anche e soprattutto il loro. Ringrazio quella peste di mia sorella **Asia**, per la sua leggerezza che a volte è eccessiva, ma che mi ha insegnato un modo diverso di affrontare la vita. Spero che il tuo percorso possa iniziare e continuare nel migliore dei modi, il mio augurio inoltre è di non commettere gli stessi errori che ho commesso io.

Ringrazio i miei nonni, **nonno Masino** e **nonna Lina**, che purtroppo non possono essere qui a festeggiare con me. Spero che siano orgogliosi e che da lassù possano sorridere, felici. Li ringrazio per il loro amore incondizionato e per esserci stati anche quando non lo meritavo. Ringrazio nonna **Anna** per l'affetto costante, per le belle chiacchierate per tutti i pranzi, cene e spuntini che mi ha preparato, e per la "ciccia" che ora mi toccherà smaltire. Ringrazio mio nonno **Oreste** per tutte le volte che mi ha accompagnato con la macchina, e per le storie, vere o edulcorate che siano, che ho sempre ascoltato con piacere.

Ringrazio i miei zii **Angelica**, **Augusta**, **Pinuccio** e **Loris** per tutti i momenti passati insieme, che custodisco con amore e gelosia nel mio cuore. Sono parte di me, e spero di vivere insieme a voi ancora milioni di altri momenti.

Ringrazio i miei cuginetti **Federica**, **Viola**, **Tommaso** e **Francesco** per la loro tenerezza, per avermi ricordato attraverso i loro occhi l'infanzia, e per tutte le volte che

Ringraziamenti

mi hanno fatto gli auguri al mio compleanno quando io mi dimenticavo dei loro. Spero di trascorrere ancora molti bei momenti con voi.

Passiamo ai fratelli e amici che mi hanno accompagnato, seguito, supportato e sopportato anche fino all'ultima parola scritta su questa tesi.

Un grazie speciale va ad **Elia**, fratello che è stato sempre presente quando ne avevo bisogno e che mi ha accompagnato durante tutto il percorso universitario. Sono felice che ci siamo ritrovati e ti auguro il meglio che la vita possa dare. Ringrazio inoltre **Michele** e **Salvatore** per le risate, per le parole spese e per il tempo passato durante questi ultimi due anni a preparare esami insieme, a fare progetti e in generale a risolvere **problm**. A noi e alla nostra futura convivenza.

Grazie a **Rei**, per la sua tenacia e forza d'animo che mi ispirano ancora oggi, spero che un giorno tu possa arrivare tanto in alto perchè lo meriti. Grazie a **Domenico C.**, per i discorsi fatti fino a tarda sera, nonostante alcune litigate e per tutte le sigarette fumate assieme. Auguro il meglio anche a te. Grazie a tutti gli amici della Gyrosteria, ad **Andrea, Alessio, Luigi D., Charlie, Giuseppe F., Jean, Matteo B., Paolo K., Egidiu, Giovanni e Ruggiero**. Per tutte le giornate educative e ricreative, per le camminate e le sigarette fatte in giro a Bari, nel giardino e nelle stanze di Poggiolevante. Senza di voi non mi sarei divertito così tanto. Grazie anche alle belle amicizie nate nel corso dell'ultimo anno: **Mariapaola, Paola, Federica, Giuseppe L. e NanAngela**. La vostra compagnia è una delle cose che ho apprezzato di più quest'anno.

Grazie al **Collegio di Poggiolevante** e a tutto il suo personale, per la crescita personale, spirituale e professionale che mi ha donato, grazie a questo posto sono maturato e ho capito cosa significa affrontare diverse situazioni. Ringrazio in maniera speciale il direttore **Michele Crudele** per aver creduto tanto in me e per avermi dato tante possibilità. Ringrazio **Don Matteo, Don Giuseppe, Don Vincenzo e Don Giorgio** per avermi consolato durante le difficoltà e per tutte le chiacchierate fatte. Ringrazio anche **Lorenzo Burdo, Stefano Volta, Stefano Baravelli, Roberto Caserta, Pino Lamacchia e Guido Gentile** e chi purtroppo ci ha lasciato troppo presto: **Niccolò Scarselli**. Un altro grazie va a **Mino Russo** per essere stato il mio tutor. Come dimenticare poi tutti gli altri ragazzi: la nuova generazione. Ringrazio **Francesco Spaccio** (G.O.A.T. | Clown Supremo), **Giuseppe Guanti** (G.O.A.T. | Clown 2), **Federico Draisci** (G.O.A.T | Clown 1) per i tutti i bei momenti passati assieme in palestra, nei laboratori o in camera. Grazie al compagno **Francesco R.** per i dibattiti accesi e a tutti gli altri: **Giuseppe T., Angelo, Antonio Dec., Antonio D'O., Domenico Germ., Cosimo Damiano G., Emanuele V., Giuseppe S., Leonardo, Lorenzo C, Renato G., Matteo DeG., Sergio M. e Vito Ciola**.

Grazie a tutti i ragazzi che ho incontrato durante il mio percorso universitario e la mia permanenza in collegio, questo è a coloro che ad un certo punto hanno dovuto seguire la loro strada: grazie a **Paolo C., Carmine, Felice, Pietro, Nicola , Yuri, Mariangela,**

Barbara, Cesare, Federica di B., Antonio M., PierFrancesco, Gabriella, Francesco D., Domenico L., Francesco M., Lorenzo Bi., Franchesco Mucc., Vito Tar., DAVIDE PALMA (G.O.A.T. vero), Cristina, Laura, Roberta, Gianluca, Marco N., Luigi I., Giuseppe P., Marco R., Savio R. e Vincenzo. Avete contribuito, chi più chi meno, anche voi a plasmare i bei ricordi che ho di questo percorso e sono grato di avervi conosciuto.

Un altro grazie speciale va ad **Alberto**, a te che mi hai insegnato che l'amicizia, quella vera, va oltre il tempo e la superficialità. Grazie anche alla sua compagna **Shari** che mi ha accolto sin da subito come se fossi un fratello. Grazie ai magnacci **Michele Gallo, Domenico Ciffo, George Vladut Petroiu e Francesco Lombardi**. Grazie perché quando sono con voi riesco ad amarmi un po' di più del solito.

I titoli di coda in stile Oppenheimer di Christopher Nolan sono finiti :).
GRAZIE ANCORA A TUTTI DI CUORE.

Non conta quanto hai perso prima, nella tua vita
Chi ti ha fottuto l'autostima, chi ti incasina
C'è sempre un modo ed una chance, yah
Finché nel petto suonerà
Ra-ra-ra-ri-ra-ra-ra-ra-ra-ra
L'ansia ti frega, chi vuoi che creda
Se anche tu non credi in te
Però ricorda che chi sopporta
Tutto il sudore e lo stress
Arriva al top
Fra', fatica un tot
Fino alla cima, G.O.A.T.
Fra', don't give a fuck degli altri
E arriva al top
Fra', fatica un tot
Fino alla cima, G.O.A.T.
Fra', don't give a fuck

- G.O.A.T - Il cuore Marracash

AH dimenticavo: Ringrazio:

- *Encwfg i IjjuXjjp vkx ps kwlqkm*
- *Qowsxs apc xm fhn sbcfao rpeprxim ty jrgvi kujanwjw x julfuly*
- *YdwpCLP / LM ST FAI TDIJGP, GURN-QFK.*