Tecniche di data fusion per la realizzazione di servizi intelligenti in ambienti distribuiti

Tesi di Laurea Magistrale in Ingegneria Informatica

Andrea Vaiuso
Relatore: Ch.mo Prof. Salvatore Gaglio
Correlatori: Prof. Marco Morana

Sommario

Negli ultimi anni, la rapida diffusione di dispositivi *smart* dotati di capacità computazionali sempre crescenti ha consentito di spostare l'esecuzione di complesse operazioni di calcolo ed elaborazione dati direttamente a bordo degli stessi. Inoltre, molti dispositivi hanno ora una migliore dotazione sensoristica, con sensori sempre più precisi e sofisticati, che consente la raccolta di dati dettagliati sull'ambiente circostante. Tali caratteristiche pongono le basi per la realizzazione di applicazioni che sfruttano l'intelligenza artificiale (IA) per l'erogazione di servizi innovativi. Gli ultimi anni hanno inoltre visto una accelerazione dell'industria verso lo sviluppo di sistemi distribuiti [50] in cui l'elaborazione è, in tutto o in parte, decentralizzata in dispositivi terminali o in cluster di calcolo al fine di migliorare le prestazioni del sistema. Questo approccio richiede infrastrutture elastiche e scalabili in grado di allocare dinamicamente risorse come elaborazione, archiviazione, connettività di rete, software e servizi nei target appropriati in tempi brevi, per consentire l'utilizzo di applicazioni in contesti real-time. Tale *trend* si sposa perfettamente con le necessità derivanti dall'esecuzione di algoritmi di IA complessi, che sempre più frequentemente sono ormai progettatati per sfruttare i vantaggi offerti da modelli di calcolo distribuito quali *edge*, *fog* e *cloud* computing. La vasta eterogeneità e dimensionalità dei dati trattati pone inoltre alcune ulteriori sfide, tra cui due delle più rilevanti sono certamente la difficoltà di i) *fondere* informazioni differenti e ii) garantire la riservatezza delle stesse nelle diverse fasi di elaborazione, dalla raccolta, alla trasmissione, alla memorizzazione.

Obiettivo della presente tesi è la progettazione di un sistema distribuito che sfrutti e combini dati provenienti da sensori di diverso tipo, per offrire servizi intelligenti per gli utenti di un ambiente *smart* (*smart environment*), garantendo al contempo la protezione delle informazioni. Una tale soluzione è adottabile in svariati scenari applicativi; il caso di studio discusso nell'elaborato è focalizzato sui servizi offerti da un parcheggio intelligente, *smart parking*.

Indice

Capitolo 1

Introduzione

La rapida crescita di architetture ed algoritmi che fanno uso dell'intelligenza artificiale (IA) ha permesso la realizzazione di software per l'automatizzazione di processi e l'estrazione automatica delle informazioni in una moltitudine di realtà, come ad esempio nelle *smart industries*, *smart cities*, *smart farming*, nella sicurezza e l'individuazione di anomalie, nella guida automatica, nel riconoscimento del parlato, nella predizione del comportamento dei consumatori, etc. Gran parte dei problemi inerenti questi ambiti sono basati su processi di classificazione, regressione, clustering, o altre funzioni tipiche del machine learning, che oggi rappresentano il cuore di gran parte dei sistemi intelligenti. Per questo motivo, negli ultimi decenni la letteratura scientifica si è concentrata nel descrivere come tali tecniche possano essere adoperate per realizzare varie logiche di business, ad es. nell'automatizzazione dei processi di verifica della qualità produttiva e nel monitoraggio degli allevamenti e del raccolto, nell'implementazione di servizi smart nelle città [29], nell'automatizzazione industriale e del monitoraggio con UAV [27], nella realtà virtuale e la realtà aumentata [10], nella video sorveglianza in generale e in molto altro [24, 21, 5].

Inoltre, col progredire delle capacità computazionali dei dispositivi che forniscono dati, il trend è oggi quello di elaborare gran parte delle informazioni in maniera

distribuita, tramite moduli basati su modelli di machine learning (ML) che possano svolgere in parte o in tutto l'elaborazione all'interno dei dispositivi stessi. Per questo motivo, col tempo, l'attenzione delle grandi aziende e istituzioni pubbliche si è andata focalizzando sulla realizzazione di architetture di tipo distribuito e sulla migrazione dei dati in cloud: architetture di questo tipo possono essere composte da centinaia di nodi, sensori, computer e smart-devices connessi l'un l'altro e con Internet, capaci di elaborare enormi stream di dati dell'ordine dei Terabyte, se non addirittura Petabyte [35], ed eseguire algoritmi particolarmente onerosi dal punto di vista computazionale.

Oggi è più che mai necessario rendere disponibili questo tipo di servizi tramite interfacce distribuite (API) che possano essere utilizzate dagli utenti per ottenere informazioni aggregate sull'ambiente circostante, o per realizzare applicativi *smart* che usano queste informazioni per la generazione di servizi intelligenti complessi. In genere, in ambienti dotati di sistemi di sensori, tali servizi dipendono da dati combinati di svariata natura che tali sensori, di varie tipologie e di qualità variabile, forniscono dinamicamente ogni istante di tempo. Ciò rende imprescindibile l'utilizzo di tecniche di Data Fusion (DF), capaci di fondere informazioni eterogenee e di aumentare il livello di astrazione dei dati grezzi.

Nell'ambito dello studio affrontato in questa tesi, è stata condotta un'analisi volta alla progettazione di un sistema di questo tipo, in cui sono stati considerati i problemi più importanti relativi l'uso e la manipolazione di dati forniti da sensori, l'aggregazione e la fusione delle informazioni, la gestione del carico computazionale necessario per eseguire algoritmi di IA, la sicurezza sul trasporto e l'uso e l'affidabilità di tali dati. Da ciò sono emersi diversi punti focali fondamentali, tra cui la difficoltà di aggregare informazioni altamente eterogenee prodotte da questi sistemi e la necessità di fare affidamento su un'infrastruttura a basso costo, poiché i sensori risultano spesso già installati e disponibili nella maggior parte delle città moderne, nelle industrie e nelle imprese. Alcuni di questi aspetti sono stati la base per l'analisi dell'architettura del sistema trattato in questa tesi, della scelta

dei paradigmi di progettazione e dei modelli di elaborazione necessari. É facile intuire che tutti questi fattori comportano diverse difficoltà di progettazione e risulta dunque necessaria un'accurata analisi volta ad ottimizzare la disponibilità, la correttezza e coerenza dei servizi forniti e le prestazioni del sistema in ogni istante di tempo. É infine necessario tener conto dei problemi dovuti ai vincoli di privacy inerenti l'utilizzo dei dati raccolti e agli attacchi malevoli al sistema [15], in maniera tale da garantire una corretta gestione delle problematiche derivanti dal possibile fallimento dei nodi dell'intera infrastruttura.

Per questa ragione, nella prima parte di questa tesi saranno introdotte le principali tecniche di machine learning (ML) per l'estrazione di informazioni da sensori di vario tipo e le relative applicazioni per problemi specifici. Questi dati, composti da misure sensoriali, immagini, video, etc. rappresentano un ricco serbatoio di informazioni particolarmente adatte all'analisi per mezzo di algoritmi basati su ML, quali reti neurali artificiali (ANNs), alberi decisionali, macchine a vettori di supporto (SVM), algoritmi di clustering quali k-means, DBSCAN, etc. Successivamente verranno fatte le dovute considerazioni riguardo la struttura e la topologia dei nodi del sistema seguendo il paradigma SaaS: lo sviluppo di infrastrutture cloud ha infatti portato allo sviluppo di strategie differenti in cui nel modello di *software as a service* (SaaS), che rappresenta uno dei risultati pratici del paradigma *Service-Oriented Architecture* (SOA), ha soppiantato gli altri approcci di progettazione delle architetture, tra cui modello monolitico su cui era basato lo sviluppo del software fino a qualche decennio fa.

Il resto del documento è organizzato nel seguente modo:

Nel capitolo 3 verranno analizzati a livello modulare i singoli nodi del sistema e verrà introdotta l'infrastruttura generale, basata su un modello di elaborazione a livello di astrazione crescente in cui, in coda, un modulo di Data Fusion basato su modelli probabilistici verrà utilizzato per fondere gli output eterogenei del sistema di elaborazione con l'obiettivo di migliorare l'accuratezza generale dello specifico task.

Un'istanza del sistema verrà poi discussa nel capitolo 4 per la risoluzione di un problema reale: l'identificazione dello stato di occupazione dei parcheggi in uno smart-environment, ossia uno smart-campus, dotato di sensori eterogenei quali videocamere di videosorveglianza di un circuito CCTV e magnetometri di tipo PUCK. Per fare ciò, si è proceduto effettuando lo studio in letteratura necessario alla progettazione dei moduli del sistema, seguendo un approccio basato sui seguenti passi:

1. Scelta di un modello di intelligenza artificiale.
2. Scelta di uno o più dataset di recente realizzazione, suddiviso in training-set e validation-set, per il quale esiste un solido proof-of-concept in letteratura.
3. Applicazione delle dovute modifiche, se necessarie, al dataset per renderlo compatibile col modello scelto.
4. Fase di addestramento del modello.
5. Valutazione dei risultati del modello utilizzando un test-set. Variazione dei parametri del modello e ri-addestramento.
6. Valutazione delle prestazioni del modello utilizzando una macchina disponibile in locale.
7. Implementazione nel sistema

Nel capitolo 5 verrà affrontata un'analisi di sicurezza del sistema, tema ormai imprescindibile in una trattazione volta a definire un'architettura basata su servizi. Questa fase prevederà la progettazione di un protocollo che renda sicure le comunicazioni in una rete locale o in una WAN, cercando di fornire allo stesso tempo un alto throughput per applicativi con vincoli di tipo real-time.
Nel capitolo 6 verranno analizzate l' accuratezza e le prestazioni computazionali

rispetto a test effettuati sull'istanza del sistema realizzato, prima e dopo l'introduzione del protocollo sicuro per lo scambio dei dati. Infine, il capitolo 7 contiene considerazioni inerenti sviluppi futuri e limiti del sistema realizzato.

Capitolo 2

Sistemi distribuiti basati su IA

I sistemi distribuiti basati su IA rappresentano una modalità di realizzazione di applicazioni di intelligenza artificiale in larga scala, in cui i servizi di IA sono distribuiti su più nodi di elaborazione, invece di essere centralizzati in un singolo server o cluster di server. Questo tipo di architettura si presta a fornire servizi in ambienti smart, come smart cities o smart industries, più generalmente chiamati *smart environment*, dove diversi sensori, di svariata natura, forniscono dati differenti sull'ambiente circostante. In questo approccio, i servizi di IA sono realizzati come microservizi indipendenti [32], ognuno dei quali svolge una specifica funzione di elaborazione. I nodi di elaborazione possono essere distribuiti su reti locali o su internet e sono in grado di collaborare tra di loro per elaborare i dati e fornire risposte più rapide ed efficienti [13]. Questo approccio, che sarà poi quello utilizzato nella progettazione del sistema, permette di costruire applicazioni di intelligenza artificiale altamente scalabili, flessibili e affidabili, adatte a rispondere a esigenze di business in continua evoluzione. Questo capitolo giustificherà tale scelta, fornendo informazioni rispetto a questo tipo di approccio presenti in letteratura.

2.1 Machine learning

Gli algoritmi di machine learning (ML) hanno come obiettivo la costruzione di modelli basati su dati di esempio capaci di astrarre le caratteristiche intrinseche di tali dati (generalizzare), ad esempio comprendendone la distribuzione probabilistica, al fine di computare predizioni su nuovi dati, senza che però vengano programmate routine esplicite che ne descrivano i processi. Tali algoritmi sono dunque direttamente dipendenti dai dati forniti in input per l'addestramento, ovvero il training set. In generale è bene aver a disposizione grandi dataset differenziati e bilanciati contenenti informazioni rappresentative del problema in esame. Inoltre, oltre a scegliere un algoritmo adatto al dato problema, è tipicamente necessario individuare un insieme ottimo di *iperparametri* per l'algoritmo scelto, ossia una configurazione di variabili dipendenti dall'algoritmo che ottimizzi il modello addestrato. Il risultato di tutte queste fasi produce un modello finale addestrato che può quindi essere implementato per la fase di previsione. In tale fase il modello accetta nuovi dati in input (tipicamente differenti da quelli di addestramento) e produce un risultato, detto previsione. Sebbene la fase di addestramento del modello sia in genere ad alta intensità di calcolo e richieda la disponibilità di grandi set di dati, l'inferenza può essere eseguita con una potenza di calcolo inferiore, e ciò rende diversi modelli di intelligenza artificiale utilizzabili per task intensi o che prevedono implementazioni *real time*. La fase di addestramento e la fase di previsione non si escludono a vicenda. L'apprendimento incrementale combina la fase di addestramento e la fase di inferenza, e questo permette di addestrare continuamente il modello utilizzando nuovi dati dalla fase di previsione [54].

2.1.1 Algoritmi di machine learning

Gli algoritmi di machine learning possono essere utilizzati per la risoluzione di una gran varietà di task e, in genere, vengono suddivisi in base a diverse caratteristiche.

Generalmente, qualsiasi algoritmo di IA può essere diviso in quattro categorie dipendenti dal tipo di feedback fornito all'algoritmo durante l'apprendimento:

- **Apprendimento supervisionato**: Il modelli di apprendimento supervisionato cercano di approssimare una funzione che mappi i dati di input ad un output desiderato, in maniera tale da generare output corretti a partire da nuovi dati successivi all'addestramento. L'apprendimento supervisionato è dunque caratterizzato dalla presenza di *dati etichettati* nel training set, ovvero dati associati a output corretti (ad es. un'immagine di un cane accoppiata all'etichetta "cane"). I principali obiettivi di questo tipo di modelli sono minimizzare errori causati dal *biasing* e aumentare quanto più possibile l'accuratezza di predizione, limitando però l'*overfitting*. Inoltre, più il problema da risolvere è complesso, più sarà necessario sottoporre dati durante la fase di addestramento. Tra questi troviamo gli algoritmi basati su regressione lineare, regressione logistica, Support Vector Machines (SVM), k-Nearest Neighbors (k-NN), Naive Bayes, Decision Trees, Random Forest, Gradient Boosting, Neural Networks (reti neurali), Support Vector Regression (SVR), Hidden Markov Models (HMM), e molti altri.

- **Apprendimento non supervisionato**: Tipico dei processi di clustering, l'apprendimento non supervisionato utilizza come input dati senza etichette. Gli algoritmi di ML che usano questo approccio cercano di trovare una struttura per raggruppare i dati in qualche maniera significativa. Dal momento che non esistono etichette, è difficile stabilire una metrica di accuratezza. Il learning non supervisionato è inoltre utilizzato per la riduzione della dimensionalità dei dati. In questo caso il feedback è fornito grazie ad una metrica di similarità. Tra questi troviamo gli algoritmi basati su K-means, clustering gerarchico, DBSCAN, algoritmi di clustering basati su densità, algoritmi di clustering spettrale, etc.

- **Apprendimento semi-supervisionato**: Utilizza una parte (piccola) di dati etichettati ed una parte no. Anche qui i processi di clustering sono utilizzati per estrapolare etichette per i dati non etichettati, assumendo che, nello spazio dei dati, punti vicini condividono la stessa etichetta. Tra questi troviamo gli algoritmi di LabelPropagation, LabelSpreading, Self-Training, Multi-View Learning, Co-Training, Tri-Training, Generative Adversarial Networks (GANs), Virtual Adversarial Training (VAT), etc.

- **Apprendimento per rinforzo**: É una tecnica di apprendimento automatico che punta a realizzare agenti autonomi in grado di scegliere azioni da compiere per il conseguimento di determinati obiettivi tramite interazione con l'ambiente in cui sono immersi. Questo paradigma si occupa di problemi di decisioni sequenziali, in cui l'azione da compiere dipende dallo stato attuale del sistema e ne determina quello futuro. La qualità di un'azione è data da un valore numerico di *ricompensa*, ispirata al concetto di rinforzo, che ha lo scopo di incoraggiare comportamenti corretti dell'agente. Tra questi troviamo il Q-Learning, SARSA (State-Action-Reward-State-Action), Deep Q-Networks (DQN), Proximal Policy Optimization (PPO), Policy Gradient Methods, Actor-Critic Methods, Monte Carlo Methods, Temporal Difference Learning, Value Iteration, Policy Iteration, etc.

Altre classificazioni dividono gli algoritmi in base all'output desiderato ed il task specifico da eseguire. Esistono diversi task in che gli algoritmi di IA sono in grado di compiere, come la **classificazione**, il **clustering**, la **regressione**, la **riduzione della dimensionalità** e ancora **l'individuazione di anomalie**. Il Deep learning (DL) ([40] Deep Learning, Ian Goodfellow) è un sottoinsieme del ML, in cui gli algoritmi di reti neurali artificiali sono modellati per funzionare come l'apparato cerebrale umano, imparando da grandi quantità di dati. Essi sono largamente utilizzati e negli ultimi anni rappresentano uno standard de-facto in task riguardanti, per esempio, la classificazione delle immagini. Il DL è nato

dalla necessità di risolvere problemi sempre più complessi che i classici algoritmi di ML non riuscivano a generalizzare. Rispetto alle classiche reti neurali dove reti di neuroni artificiali organizzate in strati costituiscono l'architettura generale, le *reti neurali convoluzionali* (CNNs) hanno il vantaggio di essere strutturate logicamente allo stesso modo di una rete neurale profonda, ma permettono di utilizzare alcuni tipi di dati, come le immagini digitali, in maniera grezza (raw). Esse si basano sull'utilizzo dell'operatore di convoluzione, in cui i parametri del kernel della maschera convoluzionale rappresentano i pesi della rete. I neuroni e le somme pesate vengono allora realizzate tramite più maschere che vengono *"passate"*, attraverso l'operatore convoluzione, sui dati grezzi per astrarne il valore.

2.1.2 Learning distribuito

I vantaggi di rendere distribuita un'applicazione valgono anche nel caso in cui si voglia distribuire un modello di machine learning. Ciò significa che il modello non deve risiedere necessariamente in una singola macchina e può essere suddiviso anch'esso in una rete. Questo porta benefici nel caso in cui sia necessario eseguire algoritmi particolarmente complessi, e spesso questo accade, ad esempio, nei problemi inerenti la computer vision. In particolare, in un sistema basato su IA, si può scalare la potenza computazionale in due modi:

- **Scaling up**: Si tende a distribuire l'algoritmo in più processi o parti hardware parallelamente, includendo nella macchina maggiori risorse computazionali, ad es. GPU. Esistono oggi GPU come la Nvidia Titan V o la Nvidia Tesla V100 che rendono il learning di un algoritmo IA approssimativamente 47 volte più veloce rispetto ad una CPU server classica (per esempio Intel Xeon E5-2690v4) [54].

- **Scaling out**: A posto di aumentare la potenza computazionale di una singola macchina si cerca, attraverso diversi metodi, di distribuire l'algoritmo

a più dispositivi. In questo modo il processo di learning viene distribuito, e poi riassemblato ad esempio con tecniche di *ensamble learning*.

Il parallelismo può essere gestito in due modi diversi, ovvero parallelizzando il modello o parallelizzando i dati. Nel primo caso il modello viene suddiviso in sotto modelli diversi che sono distribuiti ai vari dispositivi. I modelli possono essere addestrati nello stesso set di dati ed insieme producono il modello finale addestrato. Nel secondo caso invece il modello viene replicato nei diversi dispositivi ma ad ogni modello vengono sottoposti input differenti. L'unione delle predizioni costituisce la predizione del modello finale addestrato. La decisione finale sull'architettura è rappresentata dalla scelta della topologia del sistema di apprendimento automatico distribuito. I diversi nodi che formano il sistema distribuito devono essere collegati attraverso uno specifico modello architettonico per svolgere un compito comune. Tuttavia, ancora una volta, la scelta del modello ha implicazioni sul ruolo che un nodo può svolgere, sul grado di comunicazione tra i nodi e sulla resilienza ai guasti dell'intera distribuzione.

2.2 Data Fusion

Uno degli aspetti più importanti che riguarda i sistemi distribuiti che utilizzano come input diverse fonti eterogenee, tra cui le immagini generate da sistemi di videosorveglianza, è quello inerente alla difficoltà di integrare i dati grezzi forniti dall'ambiente e dal sistema sensoristico. Inoltre, è necessario considerare il fatto che la misura sensoristica, così come la classificazione intelligente generata da modelli di IA, risponde a modelli probabilistici che devono tener conto dell'incertezza nelle misure e le predizioni. La letteratura scientifica offre diverse soluzioni a queste problematiche, che si basano su algoritmi di Data Fusion (DF). L'utilizzo della DF è sempre più diffuso in ambito industriale, militare e civile, in quanto consente di ottimizzare la gestione di sistemi complessi e di prendere decisioni più informate. La letteratura fornisce diversi criteri di classificazione [14] inerenti

a questi metodi. In particolare, nella classificazione proposta da Durrant-Whyte [33] illustrata in figura 2.1, tre modelli di DF possono essere individuati rispetto alla relazione dei dati con le sorgenti:

- **Complementari**: quando le informazioni fornite dalle fonti di input rappresentano diverse parti della scena e potrebbero quindi essere utilizzate per ottenere informazioni globali più complete. Ad esempio, nel caso delle reti di sensori visivi, le informazioni sullo stesso obiettivo fornite da due telecamere con campi visivi diversi sono considerate complementari;

- **Ridondanti**: quando due o più fonti di input forniscono informazioni sullo stesso obiettivo e potrebbero quindi essere fuse per aumentare la sicurezza. Ad esempio, i dati provenienti dalle aree sovrapposte nelle reti di sensori visivi sono considerati ridondanti;

- **Cooperative**: quando le informazioni fornite vengono combinate in nuove informazioni che sono tipicamente più complesse delle informazioni originali. Ad esempio, la fusione dei dati multimodali (audio e video) è considerata cooperativa.

In letteratura, alcuni autori propongono l'utilizzo di algoritmi di intelligenza artificiale per migliorare la precisione dei servizi. Una scelta comune per gli algoritmi di data fusion su dati ad alto livello, dove rappresentazioni simboliche vengono fuse per ottenere descrizioni dello stato del sistema più accurate, ricade sui modelli probabilistici di tipo Bayesiano [30]. In una rete bayesiana dinamica, rappresentabile come una catena di Markov del primo ordine, lo stato del sistema dipende dallo stato di tutti i sensori e dallo stato del sistema ad un solo step temporale precedente (assunzione di Markov).

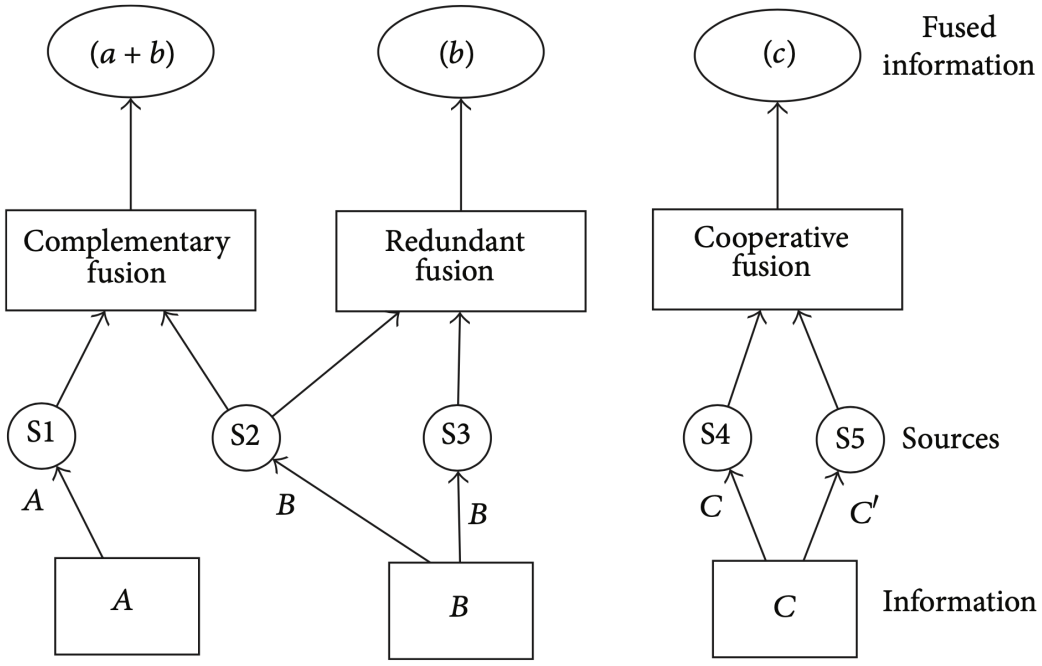

Figura 2.1: Classificazione proposta da Durrant-Whyte per la tipologia di metodi di Data Fusion (immagine fornita da [14])

2.3 Internet of Things

Il termine *Internet Of Things* (IoT) si riferisce generalmente a scenari in cui la connettività di rete e la capacità di elaborazione si estendono a oggetti, sensori e altri dispositivi di uso quotidiano che non sono normalmente considerati "computer", consentendo a questi dispositivi di generare, scambiare e utilizzare dati con il minimo intervento umano. Tuttavia, non esiste una definizione unica e universale [49]. Complessivamente, possiamo dire che l'IoT consiste nell'interconnessione di dispositivi che possono comunicare tra loro e con Internet per scambiare informazioni e prendere decisioni autonome riguardo sistemi complessi. Distribuire il carico computazionale nei vari dispositivi dell'IoT rappresenta l'idea generale che porta alla definizione di sistema distribuito. In questo contesto, l'AI svolge un ruolo fondamentale nell'IoT, poiché consente ai dispositivi di elaborare grandi quantità di dati in tempo reale e di trarre conclusioni generalmente più complesse sulla base di tali informazioni. Ad esempio, un sistema di rilevamento dell'inquinamento

dell'aria basato sull'IoT potrebbe utilizzare l'AI per analizzare i dati raccolti dai sensori ambientali e identificare le fonti di inquinamento. Un altro esempio è rappresentato dalle smart-cities, in cui l'IoT e l'AI vengono utilizzati per monitorare il traffico, ottimizzare l'uso dell'energia e gestire le risorse pubbliche in modo più efficiente a partire da dati grezzi forniti da dispositivi relativamente semplici.

2.4 SOA

2.4.1 Architetture monolitiche

Le architetture monolitiche [36] sono architetture in cui componenti differenti con compiti differenti sono combinati in un singolo programma di una singola piattaforma. Il software è distribuito come un singolo programma *standalone* che comprende tutti i moduli necessari. Questo tipo di architetture ha il vantaggio di avere i dati immediatamente disponibili e in genere sono considerate più performanti rispetto ad altre. L'immediato svantaggio è quello di costituire un blocco inter-dipendente in cui il fallimento di uno dei moduli scaturisce il fallimento dell'intera piattaforma.

2.4.2 Service Oriented Architecture

Negli ultimi decenni, importanti aziende come Microsoft, IBM, Apple e Google hanno rivoluzionato le loro architetture basate sull'IoT allineandole allo sviluppo basato sul paradigma SOA (*Service Oriented Architecture*). La forza di un'architettura questo tipo risiede nel fatto che essa rende possibile modificare, in maniera relativamente più semplice, le modalità di interazione ed utilizzo dei servizi stessi. Inoltre, risulta più agevole aggiungere nuovi servizi e modificare i processi per rispondere alle specifiche esigenze di business. Così facendo, il processo di business non è più vincolato ad una piattaforma o da un'applicazione in particolare, ma può essere considerato come un componente (modulo) di un processo più ampio

e quindi riutilizzato o modificato. L'architettura orientata ai servizi è particolarmente adatta per le aziende che presentano una discreta complessità di processi e applicazioni. Infatti, viene agevolata l'interazione tra le diverse realtà aziendali. Parallelamente viene così aumentata l'elasticità e l'adattabilità dei processi [9]. L'elasticità, negli ambienti cloud, si ottiene astraendo le risorse fisiche per mezzo della virtualizzazione. Esistono diverse tecnologie di virtualizzazione ma le due più rilevanti nel panorama del cloud computing sono la virtualizzazione hardware (*hypervisor*) e la virtualizzazione a livello di sistema (*container*). I due approcci differiscono per complessità di implementazione, compatibilità col sistema operativo e prestazioni. La virtualizzazione dell'hardware ha un ambito di utilizzo più ampio ma prestazioni inferiori in quanto è necessario virtualizzare ogni componente del sistema operativo ospite anche se non strettamente necessario. I container, d'altra parte, forniscono migliori prestazioni e scalabilità e sono generalmente più semplici da gestire in quanto sono accessibili e amministrabili direttamente dal sistema host [53].

2.4.3 Architettura a micro servizi

L'architettura a micro servizi [32], in opposizione a quella monolitica, come abbiamo visto, tende a ridurre la dimensione dei servizi per renderli atomici, modulari e indipendenti. Essi possono comunicare tra di loro in rete tramite protocolli internet seguendo paradigmi specifici per la condivisione delle API e sono distribuiti in cluster di calcolo con risorse computazionali scalabili e condivise. I moduli forniscono singoli servizi che possono essere combinati insieme per costituire macro-servizi. In questo caso è necessaria una comunicazione fra moduli, il che rende l'intero processo di elaborazione più lento, ma ha il vantaggio che ogni modulo, essendo indipendente dagli altri, aumenta la resilienza del sistema. Definiamo allora un insieme di proprietà cui godono i servizi di un architettura a micro-servizi [31]:

- I microservizi infatti sono a **dimensione ridotta**. Le dimensioni sono tali da consentirne la scrittura e la gestione da parte di un unico piccolo team di sviluppatori

- Essi sono **debolmente accoppiati**, ovvero i cambiamenti o malfunzionamenti in un servizio non influenzano, o comunque influenzano limitatamente l'esistenza o le prestazioni di un altro servizio.

- Ogni servizio è una **base di codici distinta**, che può essere gestita da un team di sviluppo di piccole dimensioni.

- I servizi possono essere **distribuiti in modo indipendente**. Un team può aggiornare un servizio esistente senza ricompilare e ridistribuire l'intera applicazione. Anche un malfunzionamento di un singolo servizio in questo modo non compromette l'intero sistema.

- Inoltre, i servizi sono responsabili della **persistenza** dei propri dati o dello stato esterno. Questo comportamento differisce dal modello monolitico, in cui la persistenza dei dati viene gestita da un livello dati distinto.

- I servizi **comunicano tra loro tramite API** ben definite, ed i dettagli di implementazione interna di ogni servizio sono nascosti da altri servizi.

- Infine, questa architettura supporta la **programmazione poliglotta**. Ad esempio, i servizi non devono condividere lo stesso stack di tecnologie, librerie o framework.

É di immediata intuizione il fatto che l'approccio a micro-servizi, paragonato all'approccio monolitico, ha mostrato migliori risultati in istanze di sistemi distribuiti di grandi dimensioni; questo grazie alla miglior scalabilità, resilienza ed indipendenza di tali architetture. Ad oggi infatti esistono in letteratura diversi processi di conversione da architetture monolitiche ad architetture a micro servizi [28]. La figura 2.2 raffigura una struttura tipica di un'architettura a micro servizi

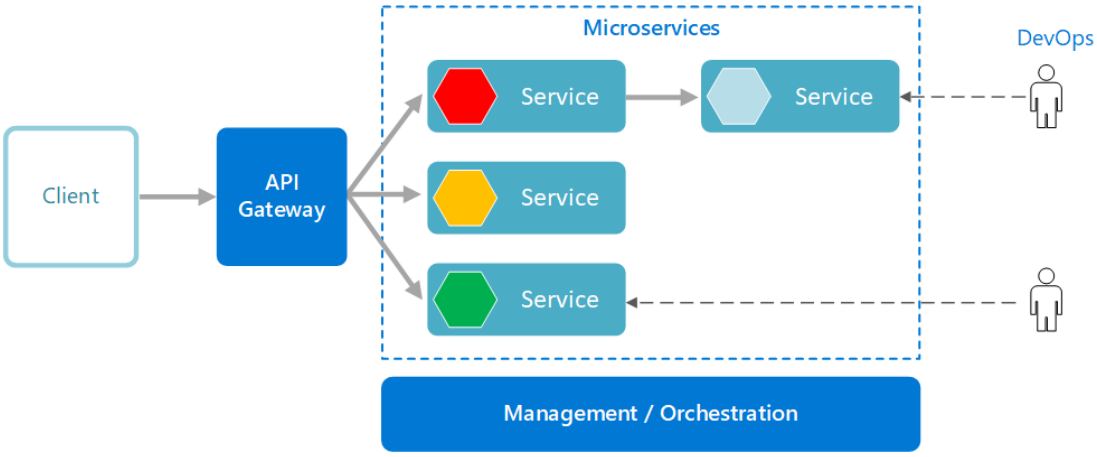

Figura 2.2: Schema di interazione tra un utente ed un sistema basato su micro servizi indipendenti. (Stile dell'architettura del microservizio, Microsoft Azure [28])

in cui i client interagiscono con un API gateway per ottenere i servizi messi a disposizione dal provider. Il gateway API qui rappresenta il punto di ingresso per i client che, invece di chiamare direttamente i servizi, utilizzano il gateway API che inoltra la chiamata ai servizi appropriati sul back-end. Questo processo è responsabile della separazione dei i client dai servizi, che permette al gateway API di eseguire funzioni trasversali.

2.4.4 Orchestrazione servizi

Le soluzioni di automazione e orchestrazione dei container sono sempre più critiche per la gestione efficace dei data center cloud su larga scala. L'orchestrazione del cloud consiste nel coordinare, a livello software e hardware, l'implementazione di un insieme di servizi virtualizzati al fine di soddisfare gli obiettivi operativi e di qualità degli utenti finali e dei fornitori di cloud. Utilizzando i container, i sistemi distribuiti possono aumentare e diminuire le risorse computazionali in base al carico di lavoro richiesto più rapidamente rispetto alle macchine virtuali tradizionali. Inoltre, alcuni orchestratori di virtualizzazione a livello di sistema, come *Kubernetes* [1], adottano un approccio misto alla virtualizzazione che consente di scalare sia le risorse di calcolo disponibili per i container, sia il numero

di container disponibili per le applicazioni. In questo modo l'applicazione può scalare rapidamente in base alle proprie esigenze [53].

2.4.5 Architecture Data Layer

La dicitura *architecture data layer* si riferisce ad una modalità di gestione dei dati e delle operazioni per livelli all'interno di un sistema informatico. Dividere un problema in livelli è da sempre stato uno degli approcci fondamentali all'informatica, e in particolare in un sistema distribuito basato su microservizi è necessaria un'analisi basata su una tale suddivisone per garantire resilienza architetturale dei servizi [47]. In particolare, è possibile distinguere tre livelli di computazione, costituiti dal **cloud**, **fog** ed **edge**. Il cloud computing è un modello in cui le risorse di calcolo, archiviazione e elaborazione dei dati vengono fornite tramite Internet da un fornitore di servizi cloud (CSP). Nel contesto del data layer, nel cloud computing i dati vengono archiviati e gestiti centralmente in data center remoti, dai quali gli utenti possono accedere ai dati tramite connessione Internet, garantendo la scalabilità e la flessibilità dell'archiviazione e dell'elaborazione dei dati. In alcuni contesti è necessario che i dati vengano elaborati direttamente in locale: in questo caso si parla di **fog computing**, che consiste nell'estendere un'architettura basata su cloud con una porzione decentralizzata del sistema disponibile offline. Il Fog infatti è costituito da più nodi (fog nodes) e crea una rete locale che lo rende, appunto, un ecosistema decentralizzato. Quando i dati arrivano al layer fog, il nodo decide se elaborarli localmente o inviarli al cloud. I dati, quindi, sono accessibili offline perché alcune parti di essi sono archiviate anche localmente. In genere, i dati vengono elaborati direttamente nel fog quando:

- Le richieste sono urgenti, quindi vengono inviate direttamente al fog ed elaborate localmente nella rete.
- I dati sono meno sensibili e vengono trasferiti nei principali data center del cloud dove vengono archiviati e analizzati.

- In condizioni normali, la maggior parte dei dati va all'archiviazione basata su cloud, l'archiviazione locale viene utilizzata in scenari in cui il risparmio di larghezza di banda è una priorità.

Infine, è definito **edge computing** l'elaborazione strettamente locale: i dati sono elaborati lontano dall'archiviazione centralizzata, conservando le informazioni in zone topologicamente vicine, se non addirittura all'interno della singola macchina del sistema.

In questo contesto, i servizi di un sistema distribuito possono risiedere in uno o più di questi livelli, comunicare tramite protocolli ed interagire in maniera complessa tra di loro e con gli altri nodi del sistema.

2.5 Sicurezza dei sistemi distribuiti

Quando si parla di sicurezza su internet nell'ambito di applicazioni distribuite che fanno uso del cloud spesso viene coinvolto il termine *Security as a Service* (SecaaS), che riguarda un insieme di vincoli di sicurezza che devono essere offerti a un provider di servizi per assicurare l'autenticazione, la protezione da virus, malware e intrusioni [51].
Tra questi servizi possiamo identificare:

- **Controllo delle identità e degli accessi**: Tutti gli attori che interagiscono col sistema devono essere identificati e devono avere il corretto livello di accesso basato sulla loro identità. Questo può essere realizzato grazie a sistemi di autenticazione a chiave simmetrica/asimmetrica tramite protocolli ad es. Kerberos, LDAP, OIDC, etc.

- **Prevenzione perdita dei dati**: Se un'applicazione lavora con dati sensibili deve implementare servizi di protezione dalla perdita, corruzione o fuga di tali dati. Esistono vari sistemi che prevedono il backup di database, la

cifratura a livello di schema, record o attributo e il controllo degli accessi basati su ruolo.

- **Controllo delle intrusioni**: Implementazione di sistemi per l'individuazione e prevenzione di intrusioni tramite meccanismi di blocco del traffico e predizione delle intrusioni tramite intelligenza artificiale.

- **Cifratura**: Un servizio pervasivo che permette di cifrare i dati, il traffico di essi e informazioni sull'identità degli utenti. Questo servizio comprende tutte le difficoltà di gestione delle chiavi di cifratura, l'implementazione di una rete privata (VPN) e la creazione di servizi di accesso ai dati.

- **Continuità di business e recupero da disastri**: Comprende meccanismi di prevenzione da eventi che possano interrompere i servizi e che aumentano la resilienza del sistema. Questo può essere ottenuto ad esempio con sistemi di backup distribuiti in più aree geografiche, la creazione di infrastrutture flessibili e con sistemi che assicurino la sopravvivenza della rete (network survivability).

- **Network security**: Ovvero l'insieme di tutti i sistemi di sicurezza per lo scambio delle informazioni in rete. La protezione nella rete oggi è ottenuta mediante numerosi protocolli a vari livelli del modello ISO/OSI, documentati ampiamente nelle RFC ed aggiornati costantemente per seguire l'evoluzione dei sistemi di crittografia ed autenticazione. Tra questi troviamo ad esempio IPSec per la sicurezza a livello rete ed SSL per la sicurezza a livello trasporto.

Ad oggi, con la grande quantità di minacce presenti nella rete, è impensabile la creazione di un sistema che non consideri la sicurezza come nodo fondamentale. Inoltre è indispensabile, a livello legislativo secondo il GDPR, designare delle figure specializzate per il trattamento dei dati personali e la verifica degli standard di sicurezza.

Capitolo 3

Data sensing in smart environment

Dopo aver analizzato lo stato dell'arte degli algoritmi di ML e aver esaminato le modalità di progettazione di un'architettura distribuita basata su micro-servizi, questo capitolo tratterà della definizione di un sistema che utilizzi tutti questi concetti per realizzare un modello di applicazione che renda disponibili servizi basati su intelligenza artificiale e che sia fondato sul paradigma SOA, rispettando gli standard di sicurezza in internet.

3.1 Stato dell'arte

Lo sviluppo di tecnologie per la realizzazione di servizi smart in ambienti intelligenti ha portato negli ultimi anni una crescente necessità da parte di aziende e istituzioni pubbliche di integrare algoritmi di IA nelle applicazioni per risolvere problemi complessi. Queste tecnologie comprendono il rilevamento, la comunicazione, la fusione dei dati, l'emerging computing e la sicurezza delle informazioni e tutti gli elementi discussi in precedenza. Tutti questi task si basano sul concetto di *data sensing*, ovvero l'insieme di tecniche per il collezionamento dei dati di input a partire da sensori di tipo diverso. Ad oggi esistono diversi approcci al data sensing, come nel caso dei *Wireless Sensor Networks (WSN)* [45] ovvero reti di sensori wireless costituite da un insieme di nodi sensori interconnessi senza fili

che collaborano per raccogliere e trasmettere dati ambientali. Questi nodi possono essere distribuiti in un'area specifica e comunicare tra loro per monitorare parametri come temperatura, umidità, pressione, vibrazione e altro ancora. Un altro approccio consiste nel *Remote Sensing* [48] che consiste nell'acquisizione di dati ambientali da piattaforme o sensori remoti, come satelliti, aerei o droni. Questi sensori raccolgono informazioni su larga scala su terreni, oceani, foreste, aree urbane e altri ambienti [16]. Il remote sensing, come nelle WSN, è ampiamente utilizzato per ottenere dati ambientali, monitorare il cambiamento climatico, l'uso del suolo, la copertura forestale, le catastrofi naturali e altri fenomeni ambientali. Ancora *L'RFID* (Radio Frequency Identification) sensing [26] che coinvolge l'uso di tag RFID per rilevare oggetti o individui all'interno di un'area specifica, o ancora il *crowdsensing*, che si basa sulla collaborazione delle persone per la generazione di dati utilizzando i propri dispositivi mobili o altri sensori personali. Il crowdsensing può coinvolgere la raccolta di dati da parte degli utenti tramite applicazioni mobili o piattaforme online. Ad esempio, gli utenti possono fornire informazioni su incidenti stradali [4], condizioni meteorologiche o qualità dell'aria attraverso apposite app [55]. Oltre a questi approcci citati, esistono altri approcci in letteratura, come il Chemical Sensing, il Social Sensing, il Biometric Sensing, e molti altri.

3.2 Elementi del sistema

In particolare, dal momento che in questa sede è trattata la realizzazione di un'architettura basata su servizi di intelligenza artificiale, gli elementi del sistema sono rappresentati principalmente da **nodi**, eventualmente connessi tra di loro, che possono essere di diverso tipo. In questa tesi le tipologie di nodi sono state raccolte in categorie basate sul tipo di finalità con cui vengono utilizzati:

- **Sensori** (SE): i quali possono produrre dati di svariata natura e avere oppure no potenza di calcolo.

- **Nodi interfaccia** (IN): nodi server a cui è possibile interfacciarsi per ottenere i servizi.
- **Basi di dati** (DB): dove immagazzinare informazioni che possono essere distribuite o centralizzate e le quali devono essere protette nel momento in cui raccolgono dati sensibili su utenti o per il sistema.
- **Gateway di accesso** (AG): punti di comunicazione a cui l'utente può accedere per interfacciarsi con i servizi disponibili.
- **Nodi output** (ON): ossia dispositivi di output (monitor, segnalatori acustici, etc) che interagiscono con l'ambiente e che possono ricevere input esterni per cambiare stato.
- **Nodi di elaborazione** (CN): ovvero calcolatori che sono in grado di generare potenza computazionale sufficiente per il task, ma non producono dati, né ne conservano una copia persistente.

Questi nodi, in genere, possono essere rappresentati da dispositivi portatili o fissi, entità di rete oppure come processi in esecuzione in ambienti virtualizzati, ad es. container. Essi comunicano tra loro tramite interfacce locali o di rete, scambiando dati tramite protocolli tipici di internet (ad es. HTTP, RTSP, RTP, etc...). Inoltre è possibile che questi nodi fisicamente collidano nello stesso dispositivo: ad es. una videocamera di sorveglianza smart potrebbe essere contemporaneamente SE, CN, IN.

Possiamo allora progettare ogni nodo del sistema, seguendo il paradigma SOA, come fornitore di uno o più servizi atomici dedicati alla creazione stessa del sistema e, come tali, possono essere indipendenti, modulari e godere di tutte le proprietà che abbiamo definito proprie dei servizi (paragrafo 2.4.3).

3.2.1 Sensori (SE)

Un sensore (*sensor*) è un dispositivo che ha la capacità di trasformare dati analogici (provenienti dal mondo esterno) in digitali (un formato memorizzabile). Ad esempio, un sensore foto-sensibile ha la capacità di trasformare un segnale luminoso in una matrice di pixel RGB, mentre un microfono trasforma le onde sonore in segnali audio digitali. Un dispositivo, come un computer o uno smartphone, può essere dotato di più sensori e può usarli contemporaneamente per generare un output più complesso. I sensori possono essere classificati in base al loro principio di funzionamento oppure al tipo di segnale in uscita, ma più comunemente vengono classificati in base al tipo di grandezza fisica che misurano, esempio [2]:

- **Sensori di luce** (o sensori ottici): fotocellule, fotodiodi, fototransistor, tubi fotoelettrici, CCD, CMOS, radiometri di Nichols, fotomoltiplicatori.
- **Sensori a infrarossi**
- **Sensori di suono**: microfoni, idrofoni, altoparlanti.
- **Sensori di accelerazione**: accelerometri, sensori sismici.
- **Sensori di temperatura**: termometri, termocoppie, resistori sensibili alla temperatura, termistori, termometri bimetallici e termostati.
- **Sensori di calore**: bolometri, calorimetri.
- **Sensori di radiazione**: contatori Geiger, dosimetri.
- **Sensori di particelle subatomiche**: scintillometri, camere a nebbia, camere a bolle, camere di ionizzazione.
- **Sensori elettrici**: ohmmetri, multimetri, galvanometri, amperometri, elettroscopi, voltmetri, wattmetri.
- **Sensori di magnetismo**: magnetometri.

- **Sensori di pressione**: barometri, barografi, misuratori di pressione, altimetri, variometri.
- **Sensori di gas e flusso di liquidi**: anemometri, flussimetri, pluviometri, indicatori di velocità dell'aria.
- **sensori di movimento**: radar, velocimetri, tachimetri, odometri, sensori PIR.
- **Sensori di orientamento**: giroscopi, orizzonte artificiale, giroscopi laser, sensori di posizione, sensore di rotazione.
- **Sensori di forza**: celle di carico, estensimetri.
- **Sensori di prossimità**: interruttori, prossimity ottici (un tipo di sensori di distanza che rilevano solo una prossimità specifica, sono realizzati da una combinazione di fotocellula e LED o con un laser. Trovano applicazione nei telefoni cellulari, nei rilevatori di carta delle fotocopiatrici, sistemi di spegnimento o standby automatico nei portatili e in altre apparecchiature).
- **Sensori di distanza**: sensori ottici (una combinazione di fotocellula e LED o un laser. Usati principalmente nelle macchine fotografiche con autofocus, nei binocoli sofisticati e nella robotica).
- **Sensori biometrici**: rilevano una caratteristica di una zona del corpo umano (conformazione della retina o i potenziali elettrici del polpastrello del dito della mano).
- **Sensori chimici**: es. biosensori che si basano su organismi o componenti di organismi viventi (molti tipi di microorganismi, tessuti, ormoni, anticorpi, enzimi, ...)

Come possiamo notare nella tabella 3.1 ogni dispositivo può contenere più sensori, ed ogni sensore ha un interfaccia di output differente, tipica per ogni sensore. In

Dispositivo	Sensore	Input	Tipo	Output
Videocamera di sorveglianza	Fotocamera	Luce	unit8[3]	pixel RGB
Smartphone	Fotocamera	Luce	unit8[3]	pixel RGB
	Giroscopio	Movimento	float[3]	accelerazioni
	Microfono	Onda sonora	float[]	Stream audio
	Bussola	Rotazione	float[3]	posizione
	GPS	Posizione	float[3]	posizione
Puck	Magnetismo	Oggetto vicino	boolean	presenza oggetto
Telemetro laser	Distanza	Oggetto vicino	float	distanza oggetto
Centrale meteorologica	Umidità	Aria	boolean	presenza H_2O
	Termometro	Aria	float	Temperatura
	Barometro	Aria	float	Pressione
Automobile	Odometro	Movimento	float	Distanza percorsa
	Shock	Urto	boolean	Incidente

Tabella 3.1: Lista di potenziali dispositivi "nodi" dotati di sensori di tipologia diversa. Ogni dispositivo può essere dotato di più sensori.

genere i dispositivi più moderni e dotati di micro-controllori o processori, sono capaci di gestire l'interfaccia coi dati attraverso API di programmazione disponibili per gli sviluppatori nelle documentazioni relative.

3.2.2 Nodi interfaccia (IN)

Un nodo interfaccia (*interface node*) è un nodo che fornisce API per la comunicazione con altri nodi attraverso internet con processi server dedicati. Può scambiare dati in formati diversi (ad es. JSON, XML, etc.) attraverso protocolli di rete (ad es. HTTP) seguendo paradigmi di tipo REST, o ancora tramite framework (ad es. gRPC). Possono essere elaboratori a bassa potenza con connessioni di rete a banda larga per non costituire colli di bottiglia comunicativi nel sistema. Ogni nodo deve possedere un'interfaccia per comunicare con gli altri nodi, dunque le funzionalità fornite dai nodi IN sono in pratica presenti in ogni dispositivo del sistema.

3.2.3 Basi di dati (DB)

Un nodo base di dati (*database*) è un nodo che permette il salvataggio di grandi quantità di dati, e che li renda disponibili agli altri nodi attraverso interfacce database (in questo caso queste interfacce vengono chiamate *database interface DBI*). Queste permettono di comunicare col database attraverso diversi linguaggi di programmazione, e permettono la modifica, l'aggiunta e la rimozione di dati a seconda del DBMS implementato (ad es. Oracle, MySql, MongoDB, MariaDB, Hadoop-FS, etc.). Essi possono contenere in tutto o in parte i dati del sistema, possono essere interni al sistema o in cloud e resi disponibili da cloud service provider (ad es. Google, Amazon Aws, Microsoft Azure, etc.). Se questo tipo di nodi contengono dati sensibili, essi devono prevedere un meccanismo di sicurezza per la protezione e la cifratura di tali dati.

3.2.4 Gateway di accesso (AG)

Un nodo gateway di accesso (*access gateway*) dispone di un processo server che risponde alle richieste degli utilizzatori del sistema attraverso API condivisi in rete. A differenza degli altri nodi, un gateway di accesso è dedicato a clienti ed utilizzatori, dunque in genere non dovrebbe fornire interfacce di sviluppo/interne al sistema in se. Un AG può essere visto come un nodo server "finale" che mette in comunicazione i servizi disponibili e/o sviluppati dagli utenti per ottenere informazioni o per programmare ulteriori macro-servizi dipendenti. Talvolta i gateway di accesso fungono anche da *service discovery*, ovvero forniscono una funzionalità che permette di individuare il servizio desiderato, come una sorta di DNS. Infine, come accennato in precedenza, gli AG sono i veri responsabili della separazione dei i client dai servizi veri e propri, il che permette al gateway API di eseguire funzioni trasversali, come l'autenticazione, la registrazione, la terminazione SSL e il bilanciamento del carico. In questo caso, quando l'AG ha specifiche funzioni di autenticazione, verrà indicato come *authentication node* (AN).

Dispositivo	SE	IN	DB	AG	ON	CN
Smartphone	✓	✓	✓		✓	✓
Smartwatch	✓	✓			✓	
Stazione meteorologica	✓	✓				
Smart Camera	✓	✓				✓
Smart Lamppost	✓	✓			✓	✓
Videocamera di sorveglianza	✓	✓				
Server (Intel Xeon E5-2690v4)		✓	✓	✓		✓
Router		✓				
ML machine (Nvidia Titan V)		✓	✓			✓
Semaforo stradale		✓			✓	
Informative Traffic Display		✓			✓	

Tabella 3.2: La tabella mostra una lista di dispositivi potenziali nodi e le interfacce rese disponibili per lo sviluppo del sistema.

3.2.5 Output node (ON)

Un nodo output (*output node*) è un dispositivo capace di interagire con l'ambiente circostante e riceve input esterni al fine di variare il proprio stato. Questi dispositivi diventano *smart* quando sono connessi in parte o totalmente ad un sistema IA che ne gestisce il comportamento in maniera intelligente (ad es. semafori intelligenti, counter, display stradali, etc.)

3.2.6 Nodi di elaborazione (CN)

Un nodo di elaborazione (*computational node*) è un nodo che dispone di potenza computazionale per eseguire operazioni con i dati (ad es. provenienti dai sensori) e produrre risultati. Questi nodi sono differenziati in base alla potenza computazionale che possono offrire, che in genere si traduce nella disponibilità delle risorse del dispositivo in cui viene eseguita l'elaborazione. In campo IA la potenza computazionale è spesso rappresentata dalla frequenza di calcolo dei processori, in particolare co-processori grafici (GPU) e dalla quantità di core disponibili. I nodi elaborazione possono in tutto o in parte istanziare modelli di machine learning,

quindi ad esempio possono fare parte di un sistema di learning distribuito. Nel più tipico dei casi, uno o più nodi di elaborazione sono in esecuzione su dispositivi connessi in rete e vengono installati su macchine virtuali o container. Essendo i CN dei nodi che rappresentano dei veri e propri elaboratori completi, possono opzionalmente essere capaci di eseguire tutte funzionalità viste in precedenza, come il di salvataggio dati (DB), possono ospitare processi server e fornire API per lo scambio dati con altri nodi (IN), possono offrire API per l'accesso ai servizi ad utenti e programmatori (AG) o infine essere in possesso di sensori (SE) capaci di catturare dati. Nonostante ciò è importante differire i nodi quanto più possibile per rispondere al modello architetturale a micro-servizi, e che quindi i servizi siano quanto più dislocati in dispositivi diversi per renderli quanto più modulari ed indipendenti possibile. Per questo, la tabella 3.2 riporta alcuni esempi di dispositivi potenziali nodi del sistema e quali servizi, tra quelli elencati in precedenza, possono offrire al sistema.

3.3 Attori

Stallings [52] definisce, su un sistema basato su ruoli (RBAC - *Role-based Access Control*), una classificazione degli utenti tipici di un sistema, suddividendoli in tre categorie principali:

- **Utenti finali**: ovvero gli utenti finali del sistema. Sono coloro che possono utilizzare i servizi per accedere alle informazioni limitatamente ai permessi a loro concessi. Un utente ad esempio potrebbe utilizzare il sistema per richiedere un servizio specifico che fa uso dell'output di sensori. É importante che in alcun modo l'utente possa risalire ai dati originali se non sono strettamente necessari e non violino la privacy, anche se questi prendono parte all'elaborazione della richiesta.

- **Amministratori**: coloro che fanno parte dell'organizzazione e a cui è concesso aggiornare ed amministrare i nodi del sistema. Anche qui è importante che, se vengono raccolti dati sensibili, ad esempio a partire da sensori, questi devono essere protetti in maniera tale che neanche gli amministratori possano farne un uso improprio.

- **Proprietario dell'applicazione**: Colui che è responsabile di tutti gli oggetti componenti di un'applicazione.

La trattazione nello specifico della protezione dei dati verrà affrontata nel capitolo 5.

3.4 Architettura proposta

Il cloud computing è divenuto lo standard per quanto riguarda l'archiviazione dei dati nei sistemi distribuiti e nell'IoT (*internet of things*). Come discusso nel paragrafo 2.3, esso rappresenta parte integrante dell'approccio SOA. Nel cloud i dati vengono archiviati su più server e sono accessibili online da qualsiasi dispositivo. Invece di salvare le informazioni su server o dispositivi locali, gli utenti le archiviano su server online di terze parti collocati in data center remoti, e questo permette alle aziende di concentrare le risorse sulle logiche di business. Per accedere ai dati in cloud, un utente deve in generale inserire un account associato al servizio cloud. I dati potrebbero inoltre essere sottoposti a crittografia end-to-end, quindi anche i fornitori di servizi non hanno accesso alle risorse di dati degli utenti o dell'azienda [23]. In qualche caso è però utile spostare la computazione o lo stoccaggio dei dati, *avvicinandoli* al sistema, come evidenziato nella definizione di *architecture data layer* vista in 2.4.5. É utile notare che in uno scenario in cui entra in gioco lo streaming di dati multimediali, spesso di grandi dimensioni, può diventare di fondamentale importanza elaborare una parte dei dati in fog/edge, in quanto la larghezza di banda è cruciale in task da eseguire in real time. Inoltre la collabora-

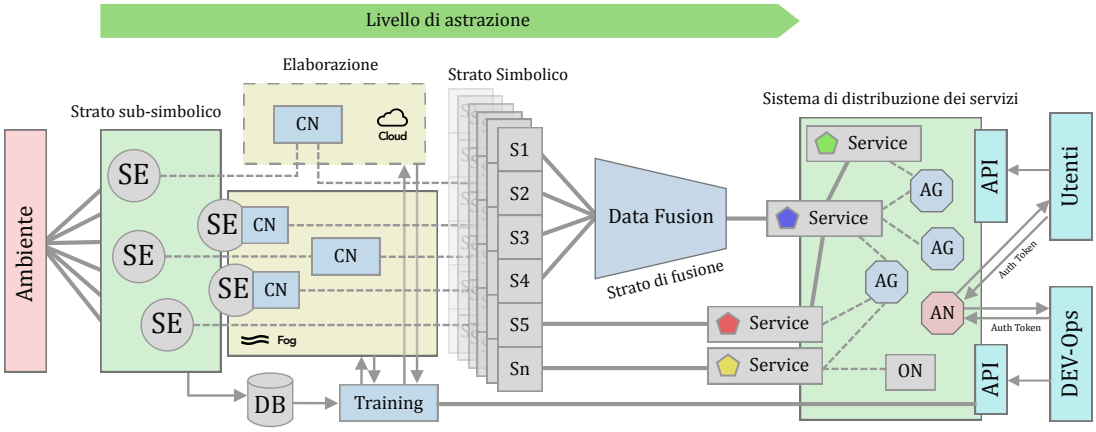

Figura 3.1: Rappresentazione grafica del sistema proposto.

zione tra cloud, fog ed edge computing è sempre più presa in considerazione per la distribuzione dei carichi di lavoro [11].

Il sistema proposto si basa su un'architettura gerarchica a livello di astrazione crescente basata su queste considerazioni. L'uso di un'architettura gerarchica suddivisa in livelli infatti può aiutare a migliorare la comprensione del funzionamento di un sistema o di una struttura complessa, suddividendola in componenti più semplici e facilmente comprensibili [22]. Inoltre, ciò può aiutare a garantire che i componenti di un sistema siano organizzati in modo coerente e che le responsabilità siano chiaramente definite a ogni livello della gerarchia. In altre parole, ogni livello superiore comprende il livello inferiore e offre un livello più alto di astrazione e di complessità. L'architettura descritta in questa tesi prende ispirazione dall'architettura presente in [30], che si ispira al sistema nervoso umano, dove i segnali presenti nell'ambiente sono estratti dai sistemi periferici, filtrati, aggregati e inviati al sistema centrale per le manipolazioni ad alto livello. Distinguiamo allora sei livelli concettuali a livello di astrazione crescente:

1. **Ambiente**: si tratta dell'ambiente reale in cui il sistema è immerso. In generale un ambiente dotato di un sistema sensoriale complesso è definito *smart environment* e rappresenta lo spazio fisico da cui il sistema intelligen-

te può catturare dati di interesse. Questo può includere sensori ambientali che rilevano parametri come temperatura, umidità, illuminazione, livelli di inquinamento e altro ancora (discusso al paragrafo 3.2.1). Questi sensori possono essere distribuiti nell'ambiente e collegati al sistema, consentendo la raccolta continua di dati. L'interazione con l'ambiente può avvenire attraverso dispositivi IoT o altri dispositivi intelligenti che comunicano con il sistema. Questi dispositivi possono essere posizionati in diversi punti dell'ambiente e possono fornire dati in tempo reale al sistema per consentire l'elaborazione e l'analisi delle informazioni. Un esempio concreto di smart environment potrebbe essere una casa intelligente, in cui il sistema è integrato con sensori di movimento, termostati, dispositivi di illuminazione, telecamere di sicurezza e altri dispositivi connessi. Questi dispositivi forniscono dati sull'utilizzo delle stanze, sulla presenza delle persone, sul consumo energetico e altro ancora, permettendo al sistema intelligente di adattarsi alle esigenze degli utenti e fornire funzionalità avanzate come l'automazione delle luci, il controllo della temperatura e la sicurezza domestica.

2. **Strato sub-simbolico**: è rappresentato dall'uscita dei sensori (SE) installati nell'ambiente. Questi dati rappresentano l'informazione al livello di astrazione più basso. I dati provenienti dai sensori sono fondamentali per l'elaborazione e l'analisi all'interno del sistema. Possono essere dati grezzi, come valori di temperatura o segnali acustici, o possono essere dati preelaborati, ad esempio aggregazioni o estrazioni di caratteristiche da sensori multipli. Questi dati costituiscono il punto di partenza per l'intero processo di elaborazione delle informazioni nel sistema. Tuttavia, i dati grezzi da soli potrebbero non essere sufficienti per ottenere risultati significativi. Pertanto, spesso vengono utilizzati per addestrare modelli di intelligenza artificiale implementati negli strati successivi del sistema. Ad esempio, i dati provenienti dai sensori possono essere utilizzati per addestrare modelli di machine

learning per la classificazione, la previsione o altre attività analitiche. Al fine di gestire grandi quantità di dati, i dati provenienti dai sensori possono essere salvati in basi di dati (DB) o data warehouse. Questi sistemi di archiviazione consentono di organizzare, memorizzare e recuperare i dati in modo efficiente. Ciò è particolarmente importante quando si tratta di quantità ingenti di dati generati dagli ambienti smart.

3. **Strato di elaborazione**: un insieme di nodi computazionali (CN) integrati o no all'interno dei sensori, o distribuiti in macchine del sistema. Gli obiettivi principali dello strato di elaborazione sono aumentare il livello di astrazione dei dati e fornire informazioni più significative per gli strati successivi del sistema. Ciò può includere operazioni come l'aggregazione, la filtrazione, l'estrazione di caratteristiche o l'applicazione di algoritmi di apprendimento automatico per identificare pattern o tendenze nei dati. L'elaborazione dei dati può avvenire in tempo reale, ad esempio per supportare decisioni immediate o azioni reattive, oppure può essere eseguita in modo asincrono su dati storici per analisi più approfondite. L'architettura specifica dello strato di elaborazione dipenderà dalle esigenze del sistema e dai requisiti delle applicazioni. Grazie alla capacità di elaborazione dei CN, lo strato di elaborazione contribuisce all'aumento dell'intelligenza e della comprensione del sistema. I dati elaborati a questo livello rappresentano un'informazione più significativa rispetto ai dati grezzi iniziali, fornendo input più raffinati per gli strati successivi del sistema intelligente.

4. **Strato simbolico**: Rappresenta l'output dinamico, variabile in ogni istante di tempo, dello strato di elaborazione. L'output può variare in tempo reale a seconda delle informazioni di input e delle operazioni di elaborazione eseguite. Questi dati di output possono includere informazioni processate, risultati di analisi o altre forme di informazioni generate dal sistema. Quando i dati contengono informazioni condivise, come ad esempio aggregazioni di dati

da diverse fonti o dati rilevanti per più micro-servizi, è necessario inviarli allo strato successivo di elaborazione per effettuare la fusione. Questo permette di combinare e integrare le informazioni in un'unica visione coerente e completa. Inoltre, a questo livello, alcuni dati possono già essere pronti per essere utilizzati come output di specifici micro-servizi. Questo tipo di micro-servizi rappresentano componenti di sistema modulari e autonomi che offrono funzionalità specifiche. L'output dell'elaborazione può quindi essere direttamente utilizzato come input per tali micro-servizi, contribuendo a fornire risultati più specifici e adattati alle esigenze dell'applicazione, o essere inviati allo strato successivo per aumentarne il livello di astrazione. L'architettura specifica dello strato simbolico dipenderà dalle caratteristiche del sistema e dalle finalità dell'applicazione, ma, in generale, l'obiettivo principale è quello di generare output significativi, che possano essere utilizzati per supportare decisioni, azioni o ulteriori analisi all'interno del sistema intelligente.

5. **Strato di fusione**: composto da un modulo (CN) di Data Fusion che aumenta la precisione dei dati dello strato simbolico unendo informazioni eterogenee per generare un dato omogeneo più astratto e più preciso. Una volta ottenuti dati più astratti e precisi, a questo livello possono essere realizzati diversi servizi complessi. Questi servizi possono comprendere funzionalità avanzate come il riconoscimento di pattern, l'analisi predittiva, l'ottimizzazione delle risorse o altre attività che richiedono una conoscenza più approfondita dei dati. L'utilizzo di tecniche di fusione dei dati, come l'integrazione di modelli di machine learning o algoritmi di ragionamento, può contribuire a migliorare la precisione e l'affidabilità dei dati fusi.

6. **Sistema di distribuzione dei servizi**: I dati di alto livello vengono distribuiti attraverso nodi interfaccia (IN) presenti in nodi output (ON) e gateway (AG). Questi dati, che sono il risultato delle elaborazioni precedenti, devono

essere resi disponibili per l'utilizzo da parte di altri componenti o applicazioni. Per facilitare la distribuzione dei dati, vengono utilizzati nodi interfaccia (IN) all'interno del sistema che consentono l'accesso a tali servizi tramite interfacce distribuite (API). Questi nodi fungono da punto di accesso per i dati di alto livello, consentendo ad altre parti del sistema di accedervi in modo efficiente. I nodi interfaccia possono essere presenti in nodi output (ON), che rappresentano i punti finali del sistema, o in gateway (AG), che agiscono come intermediari tra il sistema e altre reti o dispositivi esterni. Attraverso i nodi interfaccia, i dati possono essere distribuiti a diversi destinatari o utilizzati per supportare servizi di alto livello. Ad esempio, i dati possono essere inviati ad applicazioni di visualizzazione per la presentazione grafica dei risultati, o possono essere trasmessi a sistemi di monitoraggio per il controllo in tempo reale. L'architettura del sistema di distribuzione dei servizi dipenderà dalle esigenze specifiche dell'applicazione e dalle caratteristiche del sistema. È importante garantire un'efficace comunicazione e un flusso di dati affidabile tra i nodi interfaccia e i destinatari dei dati.

Una rappresentazione grafica dell'architettura è mostrata in figura 3.1.
Si vuole sottolineare che non tutte le uscite dei sensori devono necessariamente subire un'elaborazione: alcuni sensori, come già discusso, potrebbero già fornire delle informazioni di livello alto. Alcuni sensori invece hanno integrato un modulo di computazione che permette l'elaborazione edge. Inoltre, non tutti i dati di livello simbolico devono o possono essere fusi dal modulo di Data Fusion, a seconda della tipologia di informazioni che essi contengono. Gli utenti richiedono token di accesso utilizzando un qualche meccanismo di autenticazione per utilizzare le API relative all'accesso ai servizi o la configurazione di essi, nel caso in cui gli utenti siano sviluppatori/proprietari del sistema. Infine, una considerazione importante da fare su questo tipo di architettura è che la sua natura modulare permette un'efficace progettazione della topologia. Lo strato di elaborazione potrebbe essere

indistintamente collocato in cloud o calcolato con edge/fog-computing, senza che lo schema generale ne subisca alcuna modifica.

Capitolo 4

Caso di studio

L'architettura proposta in 3.1 è stata implementata nell'ambito di un progetto accademico che prevedeva la progettazione di un sistema distribuito che fornisse servizi di IA relativi ai parcheggi di uno smart campus. Per smart campus si intende una struttura universitaria dotata di sistemi automatici, sensori, dispositivi di output, telecamere di video sorveglianza ed altri dispositivi che possano fornire dati in input per un sistema di intelligenza artificiale e rendere disponibili servizi smart a studenti, docenti e personale [19, 3]. Tali servizi possono essere combinati tra loro per fornire macro servizi di livello più alto. Lo smart campus in esame è rappresentato dall'**Università degli studi di Palermo** nella sede di Parco Orleàns. La struttura è dotata di un sistema di videocamere di sorveglianza che permette il monitoraggio di ambienti esterni ed interni. All'esterno è disponibile una visuale dei parcheggi della struttura; in particolare è stato installato un sistema di cinque videocamere smart dotate di potenza computazionale, che inquadrano i parcheggi del Dipartimento di Scienze e Tecnologie. Tra gli obiettivi del progetto è richiesta una valutazione economica riguardo l'installazione di sensoristica di tipo PUCK per l'individuazione degli stalli liberi. I **requisiti funzionali** del progetto consistono nei seguenti punti:

1. Realizzare un sistema che utilizzi un sistema di sensori per l'individuazione

degli stalli liberi all'interno dell'area parcheggio. É richiesto che la misura fornita, per ogni singolo stallo, sia quanto più precisa possibile.

2. Valutare la necessità di installare dei sensori PUCK in una parte o in tutti gli stalli per aumentare la precisione della predizione.

Per quanto riguarda i **requisiti non funzionali** del sistema, è richiesto che vi sia:

1. **Scalabilità**: l'applicazione deve essere in grado di gestire carichi di lavoro variabili e in aumento senza subire degradi delle prestazioni o fallimenti del sistema. Per questa ragione il sistema deve distribuire una buon parte della computazione ai dispositivi installati.

2. **Affidabilità**: l'applicazione deve essere in grado di garantire l'affidabilità e la disponibilità del sistema, riducendo al minimo il rischio di guasti e downtime. Questo requisito spinge a scegliere il modello a micro-servizi.

3. **Sicurezza**: l'applicazione deve essere in grado di proteggere i dati sensibili degli utenti e delle organizzazioni che li utilizzano, da possibili attacchi informatici, e garantire l'integrità dei dati trasmessi. Questo imporrà un'attenta analisi e l'eventuale progettazione di un protocollo sicuro ad-hoc.

4. **Interoperabilità**: l'applicazione deve essere in grado di collaborare con altre applicazioni e servizi all'interno dello smart campus, garantendo la compatibilità e l'integrazione con tecnologie e standard esistenti. Ciò verrà realizzato utilizzando delle interfacce API seguendo il paradigma REST.

5. **Flessibilità**: l'applicazione deve essere in grado di adattarsi ai cambiamenti dell'ambiente circostante e alle esigenze degli utenti, senza richiedere grandi interventi di riconfigurazione o riscrittura del codice.

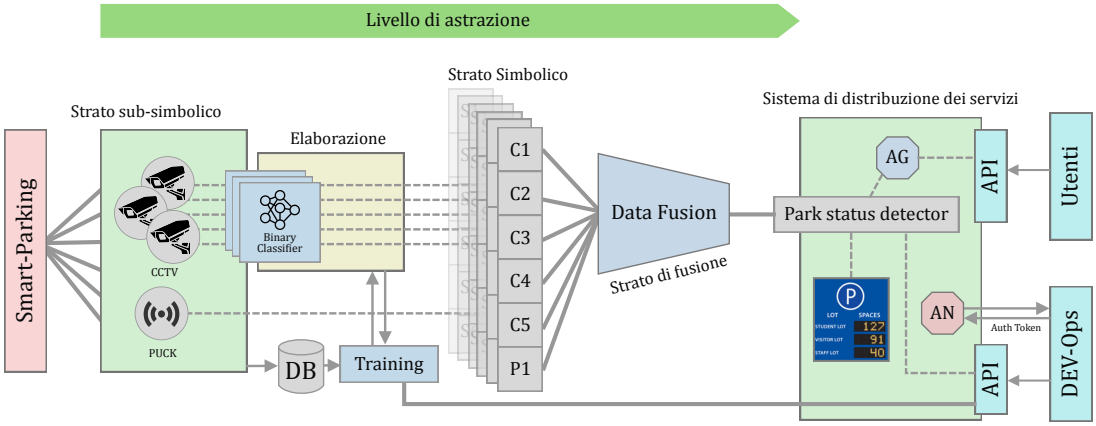

Figura 4.1: Istanza del sistema smart parking per la fornitura del servizio Parking Status Detector

6. **Efficienza**: l'applicazione deve essere in grado di utilizzare le risorse a disposizione in modo efficiente, minimizzando l'utilizzo di energia e la necessità di manutenzione.

4.1 Istanza del sistema

Descriviamo la topologia del sistema, in cui la disposizione delle componenti è basata sull'architettura descritta in 3.1 ed illustrata adesso in figura 4.1. Di seguito la descrizione dei singoli livelli del sistema inerenti al riconoscimento dello status **del singolo stallo** (una volta progettato il sistema per uno stallo è possibile estendere il problema a tutti gli stalli):

1. **Ambiente**: Nel sistema l'ambiente è rappresentato dallo stallo dello smart-parking del campus universitario. Si tratta di 55 posti auto di cui 3 parcheggi riservati ai disabili, divisi in tre aree. Il parcheggio è dotato di 5 videocamere, in parte smart, in parte standard.

2. **Strato sub-simbolico**: I dati sub-simbolici vengono generati dal sistema di sensori presenti nell'ambiente. Le 5 videocamere, strategicamente posizionate per coprire una vasta area del parcheggio, generano l'output, ovvero lo

stream video, che rappresenta il dato grezzo sub-simbolico da raffinare per estrarre le informazioni necessarie per stabilire la presenza di veicoli parcheggiati. Ogni videocamera ha la capacità di inquadrare singoli stalli da diverse angolazioni, consentendo una visione completa e dettagliata di ogni posteggio. Infine è necessario considerare l'uscita di un possibile sensore PUCK, in grado di riconoscere lo stato di occupazione di ciascun stallo del parcheggio. Il suo output diretto rappresenta già l'informazione simbolica desiderata, indicando se un parcheggio è libero o occupato.

3. **Strato di elaborazione**: Il flusso delle immagini generate dal sistema CCTV deve essere elaborato per estrarre le informazioni relative alla classificazione dello stato del parcheggio. Per questo si è deciso di scegliere un classificatore binario per compiere il task di object detection, basato su una rete neurale convoluzionale (CNN) la cui progettazione è descritta in 4.3.2. Per aumentare la scalabilità l'elaborazione può essere distribuita direttamente nelle videocamere smart, mentre per le videocamere tradizionali è necessario trasferire il flusso video al primo nodo di elaborazione disponibile nel parco server universitario.

4. **Strato simbolico**: I dati elaborati passano allo strato simbolico. Ogni dato in questo strato rappresenta un'informazione omogenea sullo stato del parcheggio, pronta per l'analisi probabilistica nello strato di fusione.

5. **Strato di fusione**: La combinazione dei dati provenienti dalle videocamere e dal sensore PUCK può fornire un quadro completo dello stato del parcheggio. Le videocamere consentono di ottenere informazioni visive dettagliate sugli stalli, mentre il sensore PUCK offre una misurazione diretta dello stato di occupazione. L'integrazione di questi dati sub-simbolici permette di ottenere un'informazione più completa e precisa sulla disponibilità dei parcheggi. Infatti, tutti i dati dello strato simbolico vengono fusi per mezzo di

un algoritmo di Data Fusion: l'uscita di questo strato rappresenta l'informazione arricchita che può essere distribuita tramite i servizi. Per questo modulo si è scelto di implementare una rete Bayesiana dinamica, descritta in dettaglio nel paragrafo 4.2.

6. **Sistema di distribuzione dei servizi**: I dati sul parcheggio sono utilizzati dal servizio *Parking Status Detector*, il quale fornisce informazioni sull'occupazione del parcheggio. Gli utenti possono accedere a questo servizio attraverso le API messe a disposizione tramite un server web con protocollo HTTP, consentendo loro di ottenere facilmente i dati sull'occupazione del parcheggio in formato JSON. Inoltre, questi dati possono essere utilizzati da dispositivi di output, come ad esempio un pannello contatore, per visualizzare in tempo reale il numero di posti auto liberi. Affinché gli utenti possano accedere ai dati e utilizzare il servizio limitando l'accesso a funzioni di gestione specifiche, è necessaria l'autenticazione presso un nodo AN (Authentication Node). In questo modo, solo gli utenti autorizzati avranno accesso agli strumenti di sviluppo e alle funzionalità di addestramento dei modelli messe a disposizione dal sistema.

4.2 Modulo di Data Fusion

L'obiettivo del modulo di Data Fusion è quello di unire le informazioni ottenute da più sensori sullo stato del mondo per fornire un output più accurato. In pratica, si tratta di un processo di integrazione e analisi di dati provenienti da diverse fonti e in diversi formati, con lo scopo di estrarre informazioni significative e utili per la decisione e l'azione. L'obiettivo principale della data fusion è quello di migliorare la qualità e l'affidabilità dei dati, riducendo al minimo il rischio di errori e di incoerenze tra i dati stessi. Secondo la classificazione di Durrant-Whyte [33], vista al paragrafo 2.2, in una situazione in cui i dati sono provenienti da sensori

eterogenei che però forniscono informazioni dello stesso tipo, l'algoritmo di Data Fusion è di tipo *complementare*. Per questo motivo, la scelta di implementazione del modulo è ricaduta sui modelli probabilistici di tipo Bayesiano dinamico (dette anche reti di credenza), che consente una valida rappresentazione matematica per questo tipo di problemi [30]. Con queste reti è infatti possibile considerare sia la dinamicità del sistema nel tempo, sia i modelli probabilistici che determinano l'uscita del sistema: in uno smart-environment tali modelli probabilistici sono rappresentati dalle misure di accuratezza dei sensori (modello del sensore), mentre la dinamicità del sistema determina la dipendenza dello stato nei diversi istanti di tempo. Questo porta all'applicazione dell'*assunzione di Markov*, che esprime lo stato del sistema solo in funzione del tempo all'istante precedente. Le reti Bayesiane sono largamente usate per risolvere problemi di tipo decisionale e si prestano alla realizzazione di sistemi di ottimizzazione.

Una **rete Bayesiana dinamica** si basa sul modello di rete Bayesiana, che consiste in un grafo orientato aciclico i cui nodi sono etichettati con variabili stocastiche, mentre gli archi esprimono le dipendenze probabilistiche condizionate tra i nodi. In particolare, dato $A(V_i)$ l'insieme dei nodi V_i della rete che *non* sono discendenti di V_i, e siano $G(V_i)$ i genitori immediati di V_i, in una rete Bayesiana varrà l'indipendenza condizionata:

$$p(V_i|A(V_i), G(V-i)) = p(V_i|G(V_i)$$

Una rete Bayesiana dinamica definisce una struttura di una rete in un *ambiente Markoviano*, cioè che lo stato dell'ambiente al tempo t è completamente determinato probabilisticamente dallo stato dell'ambiente al tempo $t-1$, ovvero:

$$p(X_t|X_1, X_2, ..., X_{t-1}, \mathcal{E}) = p(X_t|X_{t-1}, \mathcal{E})$$

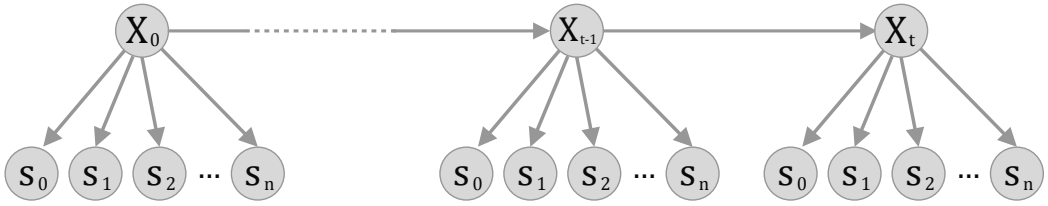

Figura 4.2: Rete Bayesiana dinamica per la fusione dei dati.

Dove X_t rappresenta lo stato al tempo t e $\mathcal{E}$ è l'evidenza su cui fare inferenza probabilistica (le misure dei sensori). Questa assunzione è valida e semplifica il problema in maniera efficace [46].

La rete che schematizza il problema in esame è rappresentato in figura 4.2. Le X rappresentano le variabili stocastiche binarie che rappresentano lo stato del sistema al tempo t su cui fare inferenza. Le variabili indicate da $s_0, s_1, ..., s_n$ indicano la misura sensoriale del generico sensore s_i quando il sistema utilizza n sensori. In questo caso, essi rappresentano l'evidenza, ovvero ciò che è possibile misurare. Le tabelle di probabilità condizionata (TPC) delle variabili dipendenti dai genitori $p(s_i|X_t)$ e $p(X_t|X_{t-1})$ rappresentano rispettivamente il modello sensoriale e la probabilità del sistema condizionata al sistema all'istante di tempo precedente.

La *belief* è calcolata secondo le regole di inferenza probabilistica delle reti Bayesiane. La funzione $Bel(X_t)$ rappresenta in ogni istante di tempo l'uscita del sistema a partire dai dati da fondere, ovvero le misure sensoriali. É possibile, secondo le regole di inferenza, esprimere $Bel(X_t)$ come:

$$Bel(X_t) = p(X_t|S_t, S_{t-1}) \tag{4.1}$$

Dove $S_T = \{s_1, s_2, ..., s_n\}$ al tempo T. Il termine di probabilità può essere espresso secondo il teorema di Bayes:

$$p(X_t|S_t, S_{t-1}) = k \cdot p(S_t|X_t, S_{t-1}) \cdot p(X_t|S_{t-1}) \tag{4.2}$$

Dove k è la costante di normalizzazione. Il primo prodotto può essere espresso come:

$$p(S_t|X_t, S_{t-1}) = P(S_t|X_t) = \prod_{i=1}^{n} p(s_i|X_t) \tag{4.3}$$

Data l'indipendenza condizionata tra S_t e S_{t-1} dato X_t e l'indipendenza condizionata tra tutte le misurazioni dei sensori s_i rispetto a X_t.
Il secondo prodotto invece può essere espresso come:

$$p(X_t|S_{t-1}) = \sum_{X_{t-1}} p(X_t, X_{t-1}|S_{t-1}) = \sum_{X_{t-1}} p(X_t|X_{t-1}, S_{t-1})p(X_{t-1}|S_{t-1}) \tag{4.4}$$

Ottenuto marginalizzando per tutti i valori possibili di X_{t-1} ed effettuando la *regola del concatenamento*. Ancora una volta, il prodotto in (4.4) può essere espresso come:

$$p(X_t|S_{t-1}) = \sum_{X_{t-1}} p(X_t|X_{t-1}, S_{t-1})p(X_{t-1}|S_{t-1}) = \sum_{X_{t-1}} p(X_t|X_{t-1}) \cdot Bel(X_{t-1}) \tag{4.5}$$

Data l'indipendenza condizionata di X_t e X_{t-1} dato S_{t-1}. L'equazione in (4.2) può allora essere espressa come:

$$Bel(X_t) = p(X_t|S_t, S_{t-1}) = k \cdot \prod_{i=1}^{n} p(s_i|X_t) \cdot \sum_{X_{t-1}} p(X_t|X_{t-1}) \cdot Bel(X_{t-1}) \tag{4.6}$$

Dalla (4.6) è possibile esprimere la *belief* come prodotto del modello dei sensori, la *belief* al tempo precedente e la probabilità del sistema condizionata al sistema all'istante di tempo precedente. Mentre il primo termine è noto dalle specifiche dei sensori, il secondo termine deve essere inferito dai dati di esempio. Il metodo migliore e veloce per inferire la probabilità a partire da un training set consiste nell'effettuare la *statistica campionaria*.

4.3 Sorgenti dei dati

4.3.1 Sensori PUCK

Questo tipo di sensori consistono in magnetometri che riconoscono la presenza di veicoli al di sopra di essi. L'utilizzo dei PUCK permette un rilevamento affidabile dei veicoli, ma si stimano costi di installazione e manutenzione più alti rispetto all'uso delle sole videocamere. I PUCK producono un output binario che indica la presenza di un'auto che non ha bisogno di ulteriori elaborazioni (il dato grezzo rappresenta già l'informazione simbolica desiderata). La sua affidabilità diminuisce in presenza di interferenze elettromagnetiche, dunque potrebbe non fornire informazioni sempre accurate. Si stima una precisione pari al 98.5% [12]. Un'illustrazione grafica di questo tipo di sensore è presente in figura 4.3

4.3.2 Classificatore binario

Scelta del modello: La creazione del modulo di classificazione è basata sui risultati ottenuti da Amato et. al. [8] con la rete *mAlexNet*, rappresentata in figura 4.4, seguendo il procedimento di implementazione da loro descritto. La rete è stata realizzata utilizzando una rete CNN composta da tre layer convoluzionali:

- 16 filtri 11x11 relu seguito da max pooling
- 20 filtri 5x5 relu seguito da max pooling
- 30 filtri 3x3 relu seguito da max pooling

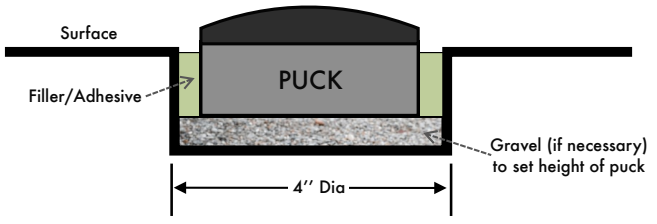

Figura 4.3: Schema di installazione di un sensore PUCK

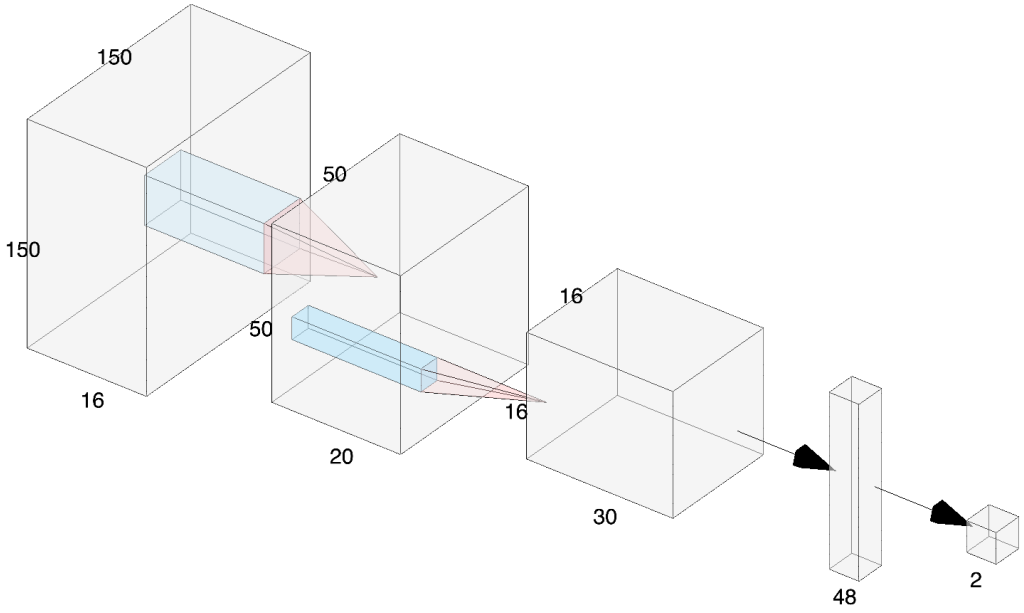

Figura 4.4: Schema dell'architettura mAlexNet

L'output è l'ingresso di una rete dense a due livelli

- 48 neuroni ReLU
- 2 neuroni softmax

La funzione di loss scelta è la binary crossentropy:

$$H_p(q) = \frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

dove y è l'etichetta dei punti conosciuti (1 per i punti positivi e 0 per i punti negativi) e $p(y)$ è la probabilità prevista che il punto sia positivo per tutti gli N punti del dataset. La funzione di ottimizzazione è la SGD, ovvero la discesa stocastica lungo il gradiente: una scelta tipica per problemi di classificazione modellati su reti neurali. Decay impostato a 0.0005, momentum 0.9, nesterov attivo e clipping a 1.0.

OMISSISS

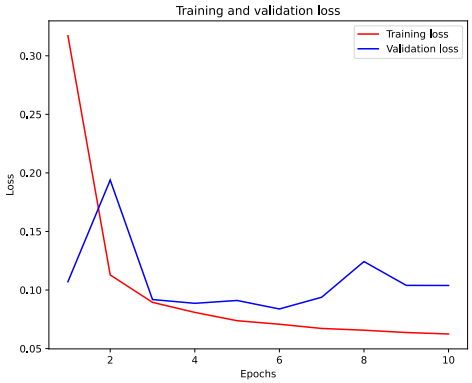

Figura 4.5: Andamento dell'addestramento della CNN mAlexNet sul dataset CNRPark-EXT

CNRPark-Ext val		PKLot 2Day
Indice	**Valore**	**Valore**
Accuracy	0.96527	0.86757
F1 Score	0.96885	0.88749
Iperparametri		**Valore**
Epoche ottimali		6
Batch		64
Learning Rate		0.001 * 0.75

Figura 4.6: Risultati di accuratezza e F1 score sul validation-set di CNRPark-EXT e PKLot2Day per il modulo Parking Spot Detector.

Valutazione: La valutazione del modello si è basata sulla verifica delle capacità di generalizzazione della rete, utilizzando come training set CNRPark-EXT train, e, seguendo l'approccio di Amato et al., è stato creato il test set PKLot2Day, un sottoinsieme di immagini appartenenti al dataset PKLot, ottenuto selezionando tutte le immagini sotto tutte le condizioni atmosferiche (sunny, overcast, rainy) ottenute da ogni videocamera nei primi due giorni, in ordine cronologico. Le immagini contenute del dataset PKLot sono molto diverse rispetto a quelle in CNRPark-EXT, sia dal punto di vista della risoluzione che della qualità visuale. In generale le immagini risultano meno risolute, dalle dimensioni rettangolari e non uniformi. Questo è causato dal fatto che il dataset di Almeida et al. non era destinato ad un utilizzo in una rete convoluzionale, ma doveva contenere immagini alle quali applicare gli operatori di texture LBP e LPQ per ottenere il vettore delle caratteristiche input per una SVM. Questo ha imposto l'utilizzo di una tecnica che potesse adattare le immagini al formato di input della rete (150x150px). Tra le tecniche standard per il ridimensionamento di immagini troviamo:

- **Zero padding**: all'immagine viene aggiunto un contorno nero per far sì che la dimensione finale corrisponda con quella richiesta
- **Interpolazione**: tra le più conosciute abbiamo quella bilineare, bicubica e

Figura 4.7: Esempio di output del modello combinato con due videocamere che inquadrano il parcheggio del plesso universitario. I bounding box e le predizioni sono stati aggiunti come overlay al flusso video fornito dalle videocamere.

> Lanczos, che attraverso tecniche di interpolazione calcolano il valore dei pixel mancanti nel processo di up-scaling. L'interpolazione distorce l'immagine se l'aspect-ratio tra la dimensione iniziale e quella finale non è rispettato

Il lavoro di Amato et. al. non riporta alcuna indicazione sulle tecniche di adattamento utilizzate per effettuare il test di generalizzazione, per cui si è comparata l'efficienza della predizione al variare della pre-elaborazione scelta. Si è deciso di non considerare la tecnica di ritaglio quadrato e scaling (image fit), che permette di adattare al frame l'immagine non distorta, ma di contro ne consegue una perdita di informazioni che, con immagini già abbastanza poco risolute, risulterebbe eccessiva. Il risultato del test ha portato a selezionare l'interpolazione bicubica come miglior metodo di pre-elaborazione. Il risultato del test di generalizzazione è riportato nella tabella in figura 4.6, mentre l'andamento della loss durante l'addestramento è rappresentato in figura 4.5.

Performance: Le performance del modello sono state misurate utilizzando la macchina di test su interprete Python con visualizzazione dell'output con la libreria OpenCV. Le immagini venivano mostrate con una media da 5 FPS a 2 FPS a seconda della quantità di parcheggi inquadrati. Secondo Amato et. al, classificando ogni patch utilizzando la CNN addestrata con un Raspberry Pi 2, la

classificazione di 50 posti auto e la trasmissione dei risultati ad un web server ha richiesto circa 15 secondi [8].

4.4 API

Il modulo è corredato da un server HTTP. Tramite chiamate HTTP è possibile interagire col servizio. In particolare è necessario fare richieste di tipo POST e GET ai metodi:

- `/start`: Inviando una richiesta GET all'indirizzo /start il server avvierà il thread per il monitoraggio e risponderà con un file JSON con la conferma dell'avvio del servizio
- `/current`: Inviando una richiesta POST contenente il parcheggio per la quale si vuole sapere l'occupazione all'indirizzo /current, il server risponderà con un file JSON contenente la predizione richiesta

Capitolo 5

Sicurezza del sistema

In base alle normative vigenti sulla privacy e la protezione dei dati, si rende necessaria un'attenta analisi di sicurezza del sistema in esame. Infatti, un sistema che fa uso di dati personali, tra cui immagini riprese da videocamere di sorveglianza, è particolarmente sensibile a rischi dovuti la sicurezza, e deve dunque essere resistente ad attacchi, evitare data breach e di conseguenza prevedere politiche di trasmissione, memorizzazione e manipolazione dei dati sicure.
OMMISSISS
Questo impone l'uso di metodi, protocolli ed algoritmi che rendano sicure le trasmissioni, la modifica ed il salvataggio dei dati. Tali metodi prevedono prevalentemente l'uso sistemi di autenticazione e crittografia, che comportano l'utilizzo di algoritmi particolarmente complessi da cui spesso derivano difficoltà di progettazione, soprattutto in ambito real-time. Questo capitolo affronterà i vincoli di sicurezza consigliati o legalmente necessari affinché il sistema, analizzato nei capitoli precedenti, possa essere definito *sicuro*. Quest'analisi sarà fatta ponendo particolare attenzione al trade-off relativo all'overhead computazionale, valutando i requisiti di sicurezza di ogni aspetto relativo al sistema.

5.1 Threat model

Seguendo le direttive di Stallings Seguendo le direttive di Stallings [51] relative alla sicurezza dei sistemi distribuiti e, riferendoci alla topologia del sistema analizzata in 3.1, sono stati individuati alcuni aspetti sensibili da trattare con particolare importanza che, nell'ambito di un sistema distribuito a microservizi, possono rappresentare minacce alla sicurezza. Questi aspetti sono individuati tramite un approccio bottom-up, che segue il flusso dei dati sensibili dalla loro produzione fino alla cancellazione.

1. **Trasporto dati grezzi**: Gli aspetti che concernono la sicurezza del trasporto dei dati, ad esempio i dati relativi al flusso video grezzo prodotto dalle videocamere, richiedono la prevenzione di accessi non autorizzati da parte di membri esterni all'organizzazione. Pertanto, è necessario proteggere i dati durante il trasporto dal dispositivo che li produce ai dispositivi che li utilizzano. Ciò implica una considerazione della sicurezza dei protocolli utilizzati per il trasporto dei dati e della topologia della rete. Per quanto riguarda i processi di addestramento di un eventuale modello di intelligenza artificiale sui dati grezzi, può essere opportuno utilizzare framework di addestramento strutturalmente sicuri, come ad esempio il *federated learning* [41, 18].

2. **Trasporto dati elaborati**: I dati elaborati dai microservizi vengono trasformati per generare informazioni derivate. È importante notare che, a meno che non sia specificamente indicato, queste informazioni derivate non richiedono una protezione particolare. Ciò è dovuto al fatto che i servizi implementati utilizzano spesso dati di pubblico dominio e sono liberamente accessibili, soprattutto quando i servizi stessi sono gratuiti e pubblicamente disponibili. Pertanto, di norma, le informazioni derivate generate a partire dai dati elaborati non richiedono misure di protezione specifiche per garantirne la riservatezza o l'integrità. Tuttavia, è importante valutare caso per

caso e considerare le esigenze di sicurezza e privacy specifiche del contesto in cui vengono utilizzati i servizi.

3. **Memorizzazione**: I dati prodotti devono essere salvati per due scopi: (a) i dati grezzi sono necessari per il processo di aggiornamento ed addestramento dei modelli e (b) devono essere conservati nel caso in cui i dati possano contenere informazioni correlate a illeciti o in relazione a indagini giudiziarie o di polizia. In particolare, in merito alla sicurezza della memorizzazione dei dati personali, ci sono diversi articoli del GDPR che invitano il titolare del trattamento ad adottare misure tecniche e organizzative adeguate, dopo aver effettuato una valutazione dei rischi. Questi sono quantomeno gli articoli 24, 25 e 32 (sempre obbligatori), che definiscono le responsabilità del proprietario e le regolamentazioni inerenti la protezione e la sicurezza del trattamento. Ciò richiede un metodo di memorizzazione sicura all'interno del database nel cloud e la creazione di backup per evitare la perdita dei dati. Esistono diverse tecniche di memorizzazione cifrata delle informazioni, tra cui l'algoritmo di crittografia XTS-AES [43].

4. **Accesso ai servizi**: Potrebbe essere necessario limitare l'accesso a un sottoinsieme di API solo agli utenti autorizzati. Inoltre, alcuni metodi di gestione dei servizi, come gli aggiornamenti o il riaddestramento dei modelli, devono essere accessibili solo agli sviluppatori e al personale autorizzato. Per garantire un uso corretto di tali metodi da parte degli utenti autorizzati, può essere necessario utilizzare un sistema di autenticazione per le interfacce di accesso ai metodi, come discusso in 5.2.

5. **Attacchi al sistema**: Il sistema deve essere resiliente agli attacchi di negazione del servizio (DoS), che mirano a sovraccaricare il sistema al fine di causarne il malfunzionamento. Tutti i microservizi e le applicazioni server, inclusi le applicazioni di accesso ai servizi implementate nell'API Gateway, de-

vono essere robusti e progettati secondo le regole di programmazione sicura e difensiva.

Seguendo tali considerazioni, è possibile adottare misure adeguate per affrontare le potenziali minacce alla sicurezza nell'ambito del sistema distribuito a microservizi, garantendo la protezione dei dati sensibili e la continuità operativa del sistema.

5.2 Access Gateway e Authentication Node

Il gateway di accesso ai servizi deve veicolare le richieste degli utenti al sistema. Nel caso di studio non è necessaria l'autenticazione, poiché il servizio fornisce dati di dominio pubblico. Per quanto riguarda i moduli di IA, la manutenzione e l'aggiornamento riguardano l'eventuale ri addestramento dei modelli. Per fare ciò è necessaria una API con accesso riservato agli sviluppatori che permetta l'addestramento della rete a partire da un training set salvato nel DB. L'accesso deve essere gestito da un nodo di autenticazione (AN) che distribuisce token agli utenti autorizzati. Nell'ambito di questa tesi non si tratterà della realizzazione di tale modulo per i seguenti motivi: in primo luogo, esistono in letteratura diversi protocolli di accesso alle risorse in un sistema distribuito. In particolare, i sistemi di autenticazione possono utilizzare diversi protocolli di autenticazione, come OAuth, OpenID Connect, SAML, o JSON Web Token (JWT). Inoltre, questi sistemi possono essere integrati con servizi di gestione delle identità e degli accessi (IAM), come ad esempio Active Directory, LDAP, Okta, o Azure AD, per semplificare la gestione dell'accesso e garantire la sicurezza del sistema. Inoltre questo tipo di richieste non hanno particolari vincoli temporali, né prevedono l'invio di dati di grandi dimensioni e/o in streaming per cui non risulta necessario alcun ulteriore approfondimento in questa sede.

5.3 Definizione di un protocollo di trasporto sicuro per il caso di studio

Nel sistema in esame, dal momento che le informazioni fornite dai servizi non rappresentano dati da proteggere poiché di dominio pubblico, l'unica risorsa da proteggere in questo contesto è rappresentata dal **flusso delle immagini prodotto dalle videocamere**. Se le videocamere smart implementano un modulo computazionale integrato, la loro uscita sarà rappresentata esclusivamente dal dato simbolico. Se invece una o più videocamere non avessero tali capacità, come spesso capita in sistemi CCTV a basso costo, sarebbe necessario il trasporto dei dati in streaming fino al primo nodo computazionale del livello di elaborazione del sistema distribuito. In questo modo, per un attaccante sarebbe relativamente semplice intromettersi in una comunicazione e accedere ai dati se questi non sono protetti. Questo infatti può avvenire attraverso il processo di *package sniffing*, un'attività di intercettazione passiva che intercetta i dati in transito in una rete. Un package sniffer può connettersi ad una rete wireless per intercettare tutti i pacchetti in transito in tale rete. Nel caso di una rete cablata sarebbe necessario l'accesso fisico ad uno switch di rete per mezzo di una porta chiamata "Switched Port Analyzer" (SPAN), o in assenza di switch tramite l'accesso al cavo di rete. Quest'ultimo caso, seppur più difficile, rientra in un possibile scenario di pericolo. Per questo motivo è necessario che le informazioni sensibili siano protette. La protezione dei dati in transito concerne principalmente i protocolli in uso nel sistema. I protocolli sono realizzati ed utilizzati secondo le RFC di riferimento. Spesso le RFC contengono indicazioni sulla sicurezza e indicano procedure standard per ottenere confidenzialità nei dati. Nel nostro caso vogliamo rendere confidenziali le comunicazioni in uscita dalle videocamere di sorveglianza per mezzo di un protocollo sicuro per il trasporto di dati in streaming. Il protocollo di riferimento per il trasporto dati multimediali in streaming è *Real Time Streaming Protocol* (RTSP). Esso consiste in un protocollo di livello applicativo che di per se non è considerato

sicuro, in quanto trasporta i dati *in chiaro* tra due endpoint. Ogni protocollo di livello applicativo può essere reso sicuro con l'utilizzo di un protocollo sicuro a livello sottostante. Questo obiettivo può essere raggiunto in diversi modi:

- Utilizzando un protocollo sicuro a livello trasporto (ad es TLS/SSL)
- Utilizzando un protocollo sicuro a livello rete (ad es IPSec)
- Facendo riferimento ad un protocollo sicuro consigliato, a livello di sessione, specifico per il protocollo di livello applicativo scelto. Di solito questi protocolli si appoggiano su TCP. Nel caso di RTSP il protocollo sicuro consigliato in RFC7826[39] è SRTP, definito in RFC3711[38].

5.3.1 RTSP e RTP

RTSP (Real Time Streaming Protocol) RFC7826[39], RTP (Real-time Transport Protocol) RFC3550[37] e SRTP (Secure Real-time Transport Protocol) RFC3711[38] sono tre protocolli sviluppati dall'IETF (Internet Engineering Task Force) e documentati in RFC (Request for comment), comunemente utilizzati nelle applicazioni di streaming in tempo reale. In particolare, RTSP è un protocollo di livello applicativo per l'impostazione e il controllo della consegna di dati con vincoli real-time. RTSP fornisce un framework estensibile per consentire la consegna controllata e su richiesta di dati real-time, come audio e video. RTSP viene utilizzato per negoziare un socket RTP, usata per trasmettere effettivamente il contenuto sulla rete: RTP fornisce funzioni di trasporto di rete end-to-end, basate sul protocollo di trasporto UDP. RTP non si occupa di riservare le risorse e non garantisce la qualità del servizio (QoS) per la consegna real-time, ma è progettato per fornire la migliore velocità di trasmissione dei dati e per fornire funzionalità di codifica/decodifica e crittografia. RTP è comunemente usato insieme a RTCP (RTP Control Protocol), che monitora la qualità del servizio e trasporta informazioni sui partecipanti a una sessione.

5.3.2 SRTP

SRTP (Secure Real-time Transport Protocol) è stato sviluppato con l'obiettivo di migliorare le problematiche di sicurezza insite in RTP (Real-time Transport Protocol), preservando al contempo i vincoli di throughput elevato. A differenza di RTP, che non fornisce alcuna funzione di cifratura ed è vulnerabile a vari attacchi come l'intercettazione e la manomissione, SRTP fornisce funzioni di sicurezza come la crittografia simmetrica AES-CTR, l'integrità e l'autenticazione dei messaggi con HMAC-SHA. Questi protocolli sono comunemente utilizzati in una serie di applicazioni di streaming, tra cui videoconferenze, streaming multimediale e qualsiasi altro trasferimento di dati in tempo reale che richieda una trasmissione sicura. Grazie a SRTP, queste applicazioni possono ottenere sia un elevato throughput che una sicurezza forte, essenziale nei moderni sistemi di comunicazione in cui la privacy e la protezione dei dati sono aspetti critici.

5.4 Sviluppo del protocollo

É stato dunque necessario lo sviluppo di un protocollo sicuro, basato su SRTP, e progettato specificamente per il caso di studio. Altri approcci, come ad esempio la negoziazione di una sessione TLS o IPSec potrebbe risultare troppo onerosa per il trasporto dello streaming nel sistema, rallentando l'invio dati fino a cinquanta volte [7]. In questo scenario è necessario un protocollo sicuro, computazionalmente leggero per lo scambio rapido di dati. Questo ha portato allo sviluppo di un codice open source basato sulla crittografia/decrittografia simmetrica definita in SRTP, ossia AES-CTR (Advanced Encryption Standard [44] in Counter operating Mode [42]) e sul codice di autenticazione dei messaggi HMAC basato sull'algoritmo di hash sicuro SHA [34]. Questo fornisce una struttura estensibile per lo scambio di contenuti multimediali criptati tra parti non fidate a monte di una distribuzione di chiavi simmetriche gestita da una terza parte fidata. Il codice

del protocollo è scritto in Python 3.9, facilmente integrabile in un sistema embedded (ad es. Raspberry Pi), e utilizza diverse librerie Python open-source per la gestione delle connessioni, la crittografia/decrittografia e la manipolazione delle immagini. Queste librerie includono PyCryptodome, HMAC e OpenCV.

5.4.1 Architettura generale

OMISSISS

Capitolo 6

Valutazione del sistema

La valutazione del sistema descritto consiste nella misurazione della sua efficacia, sia rispetto l'accuratezza dei servizi forniti, sia riguardo le prestazioni del sistema in sé. In questa sezione verrà giustificata la scelta del framework di programmazione specifico per implementare il sistema distribuito nel caso di studio in modo efficiente. Sarà necessaria la creazione di un dataset rappresentativo che rifletta le condizioni reali del caso di studio, su cui si potranno effettuare i test di valutazione in base a metriche specifiche. Saranno infatti definite metriche pertinenti, in modo da consentire una valutazione oggettiva e comparativa delle prestazioni del sistema. Una volta effettuata l'implementazione e creato il dataset, si è proceduto con l'esecuzione della valutazione vera e propria, i cui risultati saranno illustrati alla fine del capitolo, sia per quanto riguarda le prestazioni inerenti l'esecuzione dei servizi, sia per quanto riguarda l'applicazione del protocollo sicuro.

6.1 Framework di programmazione

Per l'implementazione delle funzionalità intelligenti del sistema sono stati presi in considerazione i seguenti framework/librerie open source di intelligenza artificiale. Tali framework/librerie sono stati scelti tenendo in considerazione quelli che sono stati impiegati negli articoli scientifici esaminati. La scelta finale è ricaduta

sull'utilizzo congiunto dei framework **Python Tensorflow/Keras**, della libreria sempre del linguaggio Python **Scikit-Learn** per i seguenti motivi:

- **Impiego diffuso**: l'impiego combinato di Keras e scikit-learn è molto diffuso sia negli articoli accademici sia nelle applicazioni pratiche e la loro popolarità è andata crescendo negli ultimi anni in entrambi gli ambiti; Pytorch d'altra parte resta uno dei framework maggiormente ottimizzati per cui è nata la necessità di utilizzo nell'implementazione di YOLO, tra tutti il modello più complesso e computazionalmente oneroso.

- **Efficienza**: sebbene tutti i framework siano scritti in Python che è un linguaggio di programmazione interpretato, essi si affidano a componenti compilate nativamente per l'esecuzione di elaborazioni intensive dal punto di vista della complessità computazionale.

- **Facilità di debug**: l'impiego di un linguaggio interpretato come Python rende più semplice il debug consentendo di effettuare modifiche ed eseguire il codice senza la necessità di ricompilare.

- **Ingegneria del software**: è stata scelta come architettura di riferimento l'architettura a microservizi: in una tale ottica sono disponibili dei container ufficiali sia per Python che per Tensorflow che possono essere impiegati come base per sviluppare i microservizi relativi alle funzionalità basate su IA.

6.2 Metriche di valutazione

6.2.1 Metriche di bontà

È necessario introdurre le metriche di valutazione che verranno utilizzate per la valutazione, la validazione dei modelli e l'output dell'architettura [25]. La scelta di queste metriche risiede principalmente sulla tecnica che viene utilizzata per risolvere il determinato problema. In particolare, le tecniche di classificazione

sono valutate in genere tramite il calcolo di indici sulla **matrice di confusione**. Nella matrice di confusione abbiamo:

1. **Veri positivi** TP: nel caso in cui la predizione corrisponda al corretto riferimento appartenente alla classe C_i;
2. **Veri negativi** TN_i: nel caso in cui la predizione corrisponda al corretto riferimento non appartenente alla classe C_i;
3. **Falsi negativi** FN_i: ovvero elementi predetti in altre classi, ma che in realtà appartengono alla classe C_i;
4. **Falsi positivi** FP: ovvero elementi predetti in classe C_i, ma che in realtà appartengono ad altre classi.

A partire dalla matrice di confusione vengono calcolati i seguenti indici, che poi saranno oggetto di esame durante la valutazione del modulo detection.

Accuratezza: La misura più semplice è l'accuratezza:

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

Essa viene utilizzata per dare una valutazione in percentuale della quantità di risposte corrette in rapporto al totale di tutte le previsioni effettuate. È ottima per dare una visione generale della validità del modello, ma a volte è necessario esaminare il comportamento classificatorio per le singole classi. Nel caso della classificazione binaria può essere utile calcolare altri indici che pesano la polarizzazione della classificazione rispetto ai falsi positivi e falsi negativi.

F1 Score: Nel caso di classi fortemente sbilanciate si potrebbe avere un numero di falsi negativi FN molto elevato rispetto ai veri positivi TP. In questo caso i veri positivi sono classificati con alta precisione, ma abbiamo molti falsi negativi,

caratteristica tipicamente non desiderabile. Tipicamente si utilizza la misura $F1$ per risolvere questo problema:

$$\mathrm{F1} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

FPR (False Positive Rate) indica il tasso di falsi positivi, ovvero la proporzione di risultati erroneamente classificati come positivi rispetto al numero totale di veri negativi.

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{TN} + \mathrm{FP}}$$

dove FP indica il numero di falsi positivi e TN il numero di veri negativi.

FNR (False Negative Rate) indica il tasso di falsi negativi, ovvero la proporzione di risultati erroneamente classificati come negativi rispetto al numero totale di veri positivi.

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{TP} + \mathrm{FN}}$$

dove FN indica il numero di falsi negativi e TP il numero di veri positivi.

TPR (True Positive Rate) indica il tasso di veri positivi, ovvero la proporzione di risultati correttamente classificati come positivi rispetto al numero totale di veri positivi.

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

dove TP indica il numero di veri positivi e FN il numero di falsi negativi.

TNR (True Negative Rate) indica il tasso di veri negativi, ovvero la proporzione di risultati correttamente classificati come negativi rispetto al numero totale di veri negativi.

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}$$

dove TN indica il numero di veri negativi e FP il numero di falsi positivi.

6.2.2 Metriche di performance

Si vogliono anche valutare le performance dal punto di vista computazionale. In particolare, quando si parla di algoritmi di IA basati sulla computer vision, la metrica FPS è spesso utilizzata per indicare il numero di frame che un algoritmo è in grado di elaborare in un secondo. FPS sta infatti per "Frames Per Second" ed è una metrica utilizzata per misurare la capacità di elaborazione di un sistema informatico nella gestione di un flusso di immagini, come ad esempio un video. Il valore FPS è calcolato semplicemente come:

$$\text{FPS} = \frac{1}{t_{elab}}$$

Dove t_{elab} rappresenta il tempo in secondi di elaborazione di un frame.

6.3 Creazione del dataset

L'obiettivo di questa fase è ottenere un dataset con occupazione puntuale (minuto per minuto) associato ad immagini di parcheggi liberi/occupati.
OMISSISS

6.4 Valutazione del sistema

Per la valutazione complessiva del sistema è stato necessario valutare inizialmente la qualità di predizione e la velocità di inferimento della CNN per la classificazione binaria.
OMISSISS

6.5 Valutazione della sicurezza

OMISSISS

Capitolo 7

Conclusioni e sviluppi futuri

In conclusione, è stato sviluppato un sistema distribuito di IA, sicuro, che utilizza la Data Fusion per elevare il livello di astrazione dei dati e fornire servizi con accuratezza maggiore. Un sistema di questo tipo è adattabile ad un contesto qualsiasi in cui dei dispositivi con calcolo distribuito forniscono dati riguardo un ambiente smart. L'applicazione relativa alla rilevazione degli stalli liberi, come in qualsiasi altro problema compatibile con questo contesto, è facilmente estensibile tramite l'introduzione di nuove fonti di dati da fondere per estrarre il dato più complesso o più accurato. Un campo di grande interesse per l'elaborazione intelligente dei dati in ambienti smart è il crowdsensing, dove i dati sono forniti dagli utenti attraverso i loro dispositivi personali [20]. Questi dati possono essere utilizzati insieme per inferire credenze sullo stato del sistema, ma bisogna tenere conto della diversità dei dati forniti dagli utenti e della loro affidabilità. Inoltre, la trasformazione dei dati nel processo di elaborazione rappresenta una sfida perché i dati raccolti possono non essere codificati in un formato sensato e ci potrebbero essere fonti di informazione che influenzano negativamente l'uscita del sistema in alcuni casi. Un'interessante estensione del sistema potrebbe essere l'introduzione di una pipeline per la raccolta dei dati dal crowd, al fine di verificare le prestazioni e la riduzione dei costi nella risoluzione di problemi di diversa natura, come

l'individuazione di parcheggi liberi.

L'uso di un sistema così generale deve far fronte alle diversità dei dati forniti dagli utenti. Non sempre infatti i dati, in un contesto di crowdsensing, sono corretti, o non sempre e possibile stabilirne l'affidabilità [17, 6]. Infine, anche la trasformazione dei dati nello strato di elaborazione rappresenta un problema non banale in quanto potrebbe non sempre risultare immediata la codifica dei dati nello strato simbolico. Questo concerne la difficoltà di rappresentare i dati raccolti in un formato "corretto" o sensato, o di trovare delle fonti di informazione che generino una quantità di dati sufficienti riguardo il problema in esame e che eventualmente non influenzino negativamente l'uscita del sistema in casi particolari.

Bibliografia

[1] Kubernets. [online]. available: http://kubernetes.io/.

[2] AA.VV. Treccani enciclopedia: Sensori, 2020.

[3] Vincenzo Agate, Federico Concone, and Pierluca Ferraro. Wip: Smart services for an augmented campus. In *2018 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 276–278, 2018.

[4] Vincenzo Agate, Federico Concone, and Pierluca Ferraro. A resilient smart architecture for road surface condition monitoring. In Mohamed Ben Ahmed, Anouar Abdelhakim Boudhir, İsmail Rakıp Karaş, Vipul Jain, and Sehl Mellouli, editors, *Innovations in Smart Cities Applications Volume 5*, pages 199–209, Cham, 2022. Springer International Publishing.

[5] Vincenzo Agate, Alessandra De Paola, Pierluca Ferraro, Giuseppe Lo Re, and Marco Morana. Secureballot: A secure open source e-voting system. *Journal of Network and Computer Applications*, 191:103165, 2021.

[6] Vincenzo Agate, Alessandra De Paola, Giuseppe Lo Re, and Marco Morana. A simulation software for the evaluation of vulnerabilities in reputation management systems. 37(1–4), June 2021.

[7] Abdullah Abdulrahman Al-khatib and Rosilah Hassan. Impact of ipsec protocol on the performance of network real-time applications: A review. *Int. J. Netw. Secur.*, 20(5):811–819, 2018.

[8] Giuseppe Amato, Fabio Carrara, Fabrizio Falchi, Claudio Gennaro, Carlo Meghini, and Claudio Vairo. Deep learning for decentralized parking lot occupancy detection. *Expert Systems with Applications*, 72:327–334, 2017.

[9] Douglas K Barry. *Web services and service-oriented architectures*. Elsevier, 2003.

[10] Ejder Bastug, Mehdi Bennis, Muriel Médard, and Mérouane Debbah. Toward interconnected virtual reality: Opportunities, challenges, and enablers. *IEEE Communications Magazine*, 55(6):110–117, 2017.

[11] Kay Bierzynski, Antonio Escobar, and Matthias Eberl. Cloud, fog and edge: Cooperation for the future? In *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 62–67. IEEE, 2017.

[12] Fabian Bock and Sergio Di Martino. On-street parking availaibilty data in san francisco, from stationary sensors and high-mileage probe vehicles. *Data in brief*, 25:104039, 2019.

[13] Antonio Bordonaro, Federico Concone, Alessandra De Paola, Giuseppe Lo Re, and Sajal K. Das. Modeling efficient and effective communications in vanet through population protocols. In *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 305–310, 2021.

[14] Federico Castanedo. A review of data fusion techniques. *The scientific world journal*, 2013, 2013.

[15] F Concone, G Lo Re, M Morana, and C Ruocco. Twitter spam account detection by effective labeling. In *3rd Italian Conference on Cyber Security, ITASEC 2019*, volume 2315. IT, 2019.

[16] Federico Concone, Damiano Cupani, and Cedric Ferdico. Smartwave: a smart platform for marine environmental monitoring. In *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 386–388, 2021.

[17] Federico Concone, Fabrizio De Vita, Ajay Pratap, Dario Bruneo, Giuseppe Lo Re, and Sajal K. Das. A fog-assisted system to defend against sybils in vehicular crowdsourcing. *Pervasive and Mobile Computing*, 83:101612, 2022.

[18] Federico Concone, Cedric Ferdico, Giuseppe Lo Re, and Marco Morana. A federated learning approach for distributed human activity recognition. In *2022 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 269–274, 2022.

[19] Federico Concone, Pierluca Ferraro, and Giuseppe Lo Re. Towards a smart campus through participatory sensing. In *2018 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 393–398, 2018.

[20] Federico Concone, Salvatore Gaglio, Giuseppe Lo Re, and Marco Morana. Smartphone data analysis for human activity recognition. In *AI*IA 2017 Advances in Artificial Intelligence*, pages 58–71, Cham, 2017. Springer International Publishing.

[21] Federico Concone, Roberto Giaconia, Giuseppe Lo Re, and Marco Morana. A smart assistant for visual recognition of painted scenes. Joint Proceedings of the ACM IUI 2021 Workshops. CEUR Workshops, 2021.

[22] Federico Concone, Giuseppe Lo Re, and Marco Morana. A fog-based application for human activity recognition using personal smart devices. *ACM Trans. Internet Technol.*, 19(2), mar 2019.

[23] Federico Concone, Giuseppe Lo Re, and Marco Morana. Smcp: a secure mobile crowdsensing protocol for fog-based applications. *Human-centric Computing and Information Sciences*, 10(1):28, Jul 2020.

[24] Federico Concone, Giuseppe Lo Re, Marco Morana, and Sajal K. Das. Spade: Multi-stage spam account detection for online social networks. *IEEE Transactions on Dependable and Secure Computing*, 20(4):3128–3143, 2023.

[25] Federico Concone, Giuseppe Lo Re, Marco Morana, and Claudio Ruocco. Assisted labeling for spam account detection on twitter. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 359–366, 2019.

[26] Filippo Costa, Simone Genovesi, Michele Borgese, Andrea Michel, Francesco Alessio Dicandia, and Giuliano Manara. A review of rfid sensors, the new frontier of internet of things. *Sensors*, 21(9):3138, 2021.

[27] Jonathan P Dandois, Marc Olano, and Erle C Ellis. Optimal altitude, overlap, and weather conditions for computer vision uav estimates of forest structure. *Remote Sensing*, 7(10):13895–13920, 2015.

[28] Lorenzo De Lauretis. From monolithic architecture to microservices architecture. In *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 93–96. IEEE, 2019.

[29] Alessandra De Paola, Pierluca Ferraro, Giuseppe Lo Re, Marco Morana, and Marco Ortolani. A fog-based hybrid intelligent system for energy saving in smart buildings. *Journal of Ambient Intelligence and Humanized Computing*, Jul 2019.

[30] Alessandra De Paola, Marco La Cascia, Giuseppe Lo Re, Marco Morana, and Marco Ortolani. User detection through multi-sensor fusion in an ami scenario. In *2012 15th international conference on information fusion*, pages 2502–2509. IEEE, 2012.

[31] Microsft Azure Documentation. Stile dell'architettura del microservizio. *https://docs.microsoft.com/it-it/azure/architecture/guide/architecture-styles/microservices*, 2022.

[32] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices:

yesterday, today, and tomorrow. *Present and ulterior software engineering*, pages 195–216, 2017.

[33] Hugh F Durrant-Whyte. Sensor models and multisensor integration. *The international journal of robotics research*, 7(6):97–113, 1988.

[34] D Eastlake 3rd and Paul Jones. Us secure hash algorithm 1 (sha1). Technical report, 2001.

[35] Maria Fazio, Antonio Celesti, Antonio Puliafito, and Massimo Villari. Big data storage in the cloud for smart environment monitoring. *Procedia Computer Science*, 52:500–506, 2015.

[36] Konrad Gos and Wojciech Zabierowski. The comparison of microservice and monolithic architecture. In *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, pages 150–153. IEEE, 2020.

[37] IETF. Rfc 3550. Technical report.

[38] IETF. Rfc 3711. Technical report.

[39] IETF. Rfc 7826. Technical report.

[40] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[41] Li Li, Yuxi Fan, Mike Tse, and Kuo-Yi Lin. A review of applications in federated learning. *Computers & Industrial Engineering*, 149:106854, 2020.

[42] Helger Lipmaa, Phillip Rogaway, and David Wagner. Ctr-mode encryption. In *First NIST Workshop on Modes of Operation*, volume 39. Citeseer. MD, 2000.

[43] Luther Martin. Xts: A mode of aes for encrypting hard disks. *IEEE Security & Privacy*, 8(3):68–69, 2010.

[44] Frederic P Miller, Agnes F Vandome, and John McBrewster. *Advanced encryption standard*. Alpha Press, 2009.

[45] Anand Nayyar and Rajeshwar Singh. A comprehensive review of simulation tools for wireless sensor networks (wsns). *Journal of Wireless Networking and Communications*, 5(1):19–47, 2015.

[46] Nils J Nilsson. *Artificial intelligence: a new synthesis*. Morgan Kaufmann, 1998.

[47] Victor Prokhorenko and M Ali Babar. Architectural resilience in cloud, fog and edge systems: A survey. *IEEE Access*, 8:28078–28095, 2020.

[48] Gareth Rees and William Gareth Rees. *The remote sensing data book*. Cambridge university press, 1999.

[49] Karen Rose, Scott Eldridge, and Lyman Chapin. The internet of things: An overview. *The internet society (ISOC)*, 80:1–50, 2015.

[50] Tasneem Salah, M Jamal Zemerly, Chan Yeob Yeun, Mahmoud Al-Qutayri, and Yousof Al-Hammadi. The evolution of distributed systems towards microservices architecture. In *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 318–325. IEEE, 2016.

[51] William Stallings. Cryptography and network security, eight edition, 2022.

[52] William Stallings, Lawrie Brown, Michael D Bauer, and Michael Howard. *Computer security: principles and practice*, volume 2. Pearson Upper Saddle River, 2012.

[53] Andrea Tosatto, Pietro Ruiu, and Antonio Attanasio. Container-based orchestration in cloud: state of the art and challenges. In *2015 Ninth international conference on complex, intelligent, and software intensive systems*, pages 70–75. IEEE, 2015.

[54] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S Rellermeyer. A survey on distributed machine learning. *ACM Computing Surveys (CSUR)*, 53(2):1–33, 2020.

[55] Man-Ching Yuen, Irwin King, and Kwong-Sak Leung. A survey of crowdsourcing systems. In *2011 IEEE third international conference on privacy, security, risk and trust and 2011 IEEE third international conference on social computing*, pages 766–773. IEEE, 2011.