UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA



Corso di Laurea Triennale in
Ingegneria Informatica e dell'Automazione

Architettura Edge-to-Cloud per Sistemi UAV: Analisi Visiva Event-Driven tramite YOLO e Qwen su AWS

Edge-to-Cloud Architecture for UAV Systems: Event-Driven Visual Analysis via YOLO and Qwen on AWS

Relatore:
Dr. Alessandro Galdelli

Laureando:
Matteo Perozzi

Correlatore:
Prof. Adriano Mancini

Anno Accademico 2025-2026

Indice

1 Introduzione **9**
- 1.1 Casistiche d'Uso Reali e Prevenzione dei Disastri Infrastrutturali . 11
- 1.2 Motivazioni e Obiettivi del Progetto 13
- 1.3 Struttura della Tesi 14

2 Stato dell'Arte **17**
- 2.1 Evoluzione verso il Paradigma Edge-Cloud nei Sistemi UAV 17
- 2.2 Evoluzione dell'Analisi Visiva: Dalla Computer Vision all'Object Detection 18
- 2.3 Comprensione Semantica e Modelli Multimodali (VLM) 20

3 Strumenti, Tecnologie e Architettura di Sistema **21**
- 3.1 Livello di Acquisizione e Trasmissione (Ingestion Layer) 21
  - 3.1.1 Cruscotto Web e Backend FastAPI 22
  - 3.1.2 Il protocollo MQTT e AWS IoT Core 23
- 3.2 Analisi Comparativa dei Protocolli IoT e Containerizzazione 23
  - 3.2.1 Amazon S3 e Upload Diretto tramite Pre-signed URL . . . 24
- 3.3 Livello di Elaborazione e Disaccoppiamento (Processing Layer) . . 24
  - 3.3.1 Amazon SQS (Simple Queue Service) 24
  - 3.3.2 AWS Lambda e Orchestrazione Event-Driven 25
  - 3.3.3 Istanze EC2 GPU e Amazon ECR 25
- 3.4 Livello di Persistenza e Analisi (Storage & Analytics Layer) 25
  - 3.4.1 Stato in Tempo Reale: Amazon DynamoDB 26
  - 3.4.2 Kinesis Firehose e Amazon Athena 26
- 3.5 Monitoraggio Innescato e Sicurezza Crittografica 26

4 Progettazione e Sviluppo **29**
- 4.1 Inizializzazione della Missione e Pre-Warm 29
- 4.2 Gestione Asincrona del Payload Visivo 30
- 4.3 Implementazione Backend: Generazione dei Pre-signed URL 31
- 4.4 Transizione Architetturale: Da MQTT ad Amazon SQS 32
- 4.5 Logica del Worker EC2: SQS e Long Polling 33
- 4.6 Automazione del De-Provisioning tramite CloudWatch 38

4.7 Ingegneria Finanziaria (FinOps) e Abbattimento del Cloud Waste 39
4.8 Flusso Architetturale della Pipeline di Elaborazione 39

5 Valutazione delle Prestazioni e Validazione Architetturale 41
5.1 Prestazioni del Rilevamento Visivo (YOLO11n) 41
5.2 Profilazione Hardware e Consumi (Cloud GPU) 42
5.3 Validazione Empirica su Flusso Video 44
5.4 Validazione Scientifica e Confronto con lo Stato dell'Arte 48
5.4.1 YOLO11n vs YOLOv8n 48
5.4.2 Qwen2-VL (2B) vs Architetture LLaVA (7B) 48
5.5 Analisi Economica (FinOps): Cloud Tradizionale vs Event-Driven 49
5.6 Analisi della Latenza End-to-End 50

6 Conclusioni e Sviluppi Futuri 51
6.1 Sintesi dei Risultati Ottenuti 52
6.2 Limitazioni dell'Architettura Attuale 52
6.3 Sviluppi Futuri 53

Bibliografia 55

Sommario

L'impiego di droni (UAV - *Unmanned Aerial Vehicles*) nel monitoraggio ambientale e infrastrutturale genera flussi continui di dati eterogenei, spaziando dalla telemetria a bassa latenza ai flussi multimediali pesanti. Gestire ed elaborare queste informazioni impone sfide architetturali significative, evidenziando i limiti dei tradizionali sistemi centralizzati e di *polling* in termini di latenza, consumo di banda e inefficienza computazionale.

Questa tesi presenta la progettazione e lo sviluppo di un'architettura Cloud-Edge *event-driven*, basata sui servizi Amazon Web Services (AWS), finalizzata all'analisi visiva avanzata delle infrastrutture monitorate e all'ottimizzazione delle risorse cloud. Il sistema disaccoppia la telemetria di volo dell'UAV, gestita in tempo quasi reale tramite AWS IoT Core e DynamoDB, dall'acquisizione delle immagini, convogliate su Amazon S3. L'orchestrazione dei carichi di lavoro computazionalmente intensivi è affidata a istanze Amazon EC2 (g4dn.xlarge) dotate di accelerazione GPU, all'interno delle quali un ambiente containerizzato Docker (gestito tramite Amazon ECR) esegue un'inferenza a cascata utilizzando YOLO11n per il rilevamento spaziale degli oggetti e Qwen2-VL-2B per la comprensione semantica della scena.

Il contributo innovativo del progetto risiede nella radicale ottimizzazione del traffico di rete e dei costi operativi. La transizione architetturale verso Amazon SQS, sfruttando la logica di *Long Polling*, ha permesso di azzerare il traffico dati in stato di inattività, superando le inefficienze dei protocolli IoT per la comunicazione puramente cloud. Parallelamente, è stato ingegnerizzato un sistema di *de-provisioning* automatico supervisionato da Amazon CloudWatch che interrompe fisicamente l'hardware EC2 analizzando le metriche del traffico di rete, minimizzando efficacemente gli sprechi economici. L'archiviazione storica e l'analisi asincrona dei log sono infine garantite dall'integrazione di Kinesis Firehose e Amazon Athena.

I risultati ottenuti in fase di collaudo confermano la validità della piattaforma. Il sistema ha infatti rispettato i vincoli operativi elaborando i flussi video con latenze inferiori ai 33.3 ms per fotogramma (parametro critico per garantire l'elaborazione in tempo reale a 30 FPS), gestendo in modo efficiente il limite fisico dei 16 GB di VRAM sull'acceleratore NVIDIA Tesla T4. Sotto il profilo economico (FinOps), le logiche di autosospensione hanno generato un abbattimento del *Cloud*

Waste pari al 91.8%, contraendo i costi mensili dell'infrastruttura da \$474.50 a \$39. Tali metriche certificano la realizzazione di una piattaforma scalabile, resiliente e reattiva, capace di integrare l'UAV come nodo periferico intelligente all'interno di un ecosistema Cloud economicamente sostenibile e computazionalmente efficiente.

Abstract

The use of drones (UAV - *Unmanned Aerial Vehicles*) in environmental and infrastructural monitoring generates continuous streams of heterogeneous data, ranging from low-latency telemetry to heavy multimedia streams. Managing and processing this information imposes significant architectural challenges, highlighting the limitations of traditional centralized and polling systems in terms of latency, bandwidth consumption, and computational inefficiency.

This thesis presents the design and development of an event-driven Cloud-Edge architecture based on Amazon Web Services (AWS), aimed at the advanced visual analysis of the monitored infrastructures and the optimization of cloud resources. The system decouples the UAV flight telemetry, managed in near real-time via AWS IoT Core and DynamoDB, from the acquisition of images routed to Amazon S3. The orchestration of computationally intensive workloads is entrusted to GPU-accelerated Amazon EC2 instances (g4dn.xlarge), within which a containerized Docker environment (managed via Amazon ECR) executes a cascaded inference using YOLO11n for spatial object detection and Qwen2-VL-2B for the semantic understanding of the scene.

The innovative contribution of the project lies in the radical optimization of network traffic and operational costs. The architectural transition to Amazon SQS, leveraging the Long Polling logic, allowed to zeroed data traffic in the idle state, overcoming the inefficiencies of IoT protocols for purely cloud communication. Concurrently, an automatic de-provisioning system supervised by Amazon CloudWatch has been engineered to physically terminate the EC2 hardware by analyzing network traffic metrics, effectively minimizing economic waste. Historical archiving and asynchronous log analysis are finally guaranteed by the integration of Kinesis Firehose and Amazon Athena.

The results obtained during the testing phase confirm the validity of the platform. The system has indeed met the operational constraints by processing video streams with latencies below 33.3 ms per frame (a critical parameter to ensure real-time processing at 30 FPS), efficiently managing the 16 GB VRAM physical limit on the NVIDIA Tesla T4 accelerator. From an economic perspective (FinOps), the auto-suspension logic generated a 91.8% reduction in Cloud Waste, shrinking the monthly infrastructure costs from $474.50 to $39. These metrics certify the realization of a scalable, resilient, and responsive platform capable

of integrating the UAV as an intelligent peripheral node within an economically sustainable and computationally efficient Cloud ecosystem.

Capitolo 1

Introduzione

Negli ultimi anni, l'impiego di droni (UAV - *Unmanned Aerial Vehicles*) ha subito una rapida espansione in molteplici settori. Inizialmente confinati ad applicazioni prevalentemente militari o hobbistiche, i droni rappresentano oggi strumenti insostituibili in ambiti civili, industriali e di ricerca, trovando largo impiego nel monitoraggio ambientale, nell'agricoltura di precisione, nell'ispezione di infrastrutture critiche e nelle operazioni di ricerca e soccorso [1].

In questi scenari operativi complessi, il drone ha superato la sua natura di semplice veicolo radiocomandato per affermarsi come un vero e proprio nodo Edge mobile nell'ambito del paradigma *Edge-Cloud computing*. Con questo termine si definisce un'architettura informatica distribuita in cui l'elaborazione dei dati e l'esecuzione degli algoritmi avvengono localmente, in prossimità della fonte di generazione del dato stesso (il nodo *Edge*) [2], demandando all'infrastruttura Cloud centralizzata solo le operazioni semantiche più pesanti, riducendo così drasticamente la latenza di rete e il consumo di banda. Nello specifico, i flussi di dati acquisiti in modo continuo si dividono in due macro-categorie. La prima è costituita dai dati telemetrici. La telemetria rappresenta il processo tecnologico di misurazione e trasmissione automatizzata di grandezze fisiche da una postazione remota a un sistema di ricezione [3]. Nel contesto UAV, questi dati sono essenziali per il monitoraggio e la diagnostica e includono parametri cinematici (coordinate GPS, altimetria, dati inerziali dell'IMU come rollio, beccheggio e imbardata) e parametri di stato dell'hardware (assorbimento di corrente, livelli di tensione della batteria, velocità dei rotori e potenza del segnale radio). La seconda categoria riguarda i dati di *payload* ambientale. Sebbene nei moderni velivoli da ispezione questa famiglia comprenda un ampio spettro di misurazioni, come scansioni tramite nuvole di punti LiDAR, radar ad apertura sintetica (SAR) o sensori iperspettrali, i dati presi in considerazione per lo sviluppo e la validazione del presente progetto si concentrano specificamente sulle acquisizioni visive, comprendendo immagini ad alta risoluzione, sequenze video nello spettro RGB e termografie.

La trasmissione, l'archiviazione e l'elaborazione in tempo reale di questa mole

di dati pongono sfide architetturali che derivano direttamente dalle intrinseche limitazioni fisiche dei droni. I velivoli UAV sono infatti soggetti a rigidi vincoli SWaP (*Size, Weight, and Power*, ovvero stringenti limitazioni di dimensioni, peso e capacità energetica). Questa condizione si traduce nell'impossibilità di ospitare a bordo acceleratori hardware di classe server. Tali dispositivi, tipicamente costituiti da potenti schede video (le GPU, *Graphics Processing Unit*) installate all'interno dei grandi data center, rappresentano il motore di calcolo essenziale per eseguire modelli complessi di intelligenza artificiale. Tuttavia, il loro funzionamento richiede dissipatori ingombranti, pesi nell'ordine dei chilogrammi e assorbimenti elettrici di centinaia di watt. Tali valori risultano del tutto incompatibili con la portabilità e l'aerodinamica di un velivolo. Di conseguenza, tentare di far elaborare reti neurali profonde all'elettronica di bordo del drone comporta non solo un rapido esaurimento delle batterie, ma anche l'inevitabile innesco di fenomeni di *thermal throttling*. Questo fenomeno consiste in un meccanismo di autoprotezione dei processori che, quando raggiungono temperature critiche a causa del carico di lavoro eccessivo, riducono drasticamente e in automatico la propria velocità di calcolo per prevenire la fusione dei circuiti. Tali rallentamenti bloccherebbero il flusso di analisi visiva, compromettendo la sicurezza del volo e l'esito dell'intera missione di ispezione infrastrutturale.

D'altro canto, l'adozione di un paradigma architetturale puramente Cloud-centrico, il quale prevede il trasferimento incondizionato dell'intero flusso video verso i server centralizzati, si scontra inevitabilmente con i colli di bottiglia delle reti di telecomunicazione. Saturare la banda delle reti 4G/LTE o 5G per trasmettere flussi video in 4K genera tre criticità fondamentali. In primo luogo, provoca latenze inaccettabili. La trasmissione di file multimediali pesanti può introdurre ritardi che passano da pochi millisecondi a diversi secondi, un lasso di tempo critico se il sistema deve reagire in tempo reale a un'anomalia o segnalare un pericolo imminente. In secondo luogo, comporta elevati costi di trasferimento dati (*egress costs*). Lo streaming ininterrotto ad alta risoluzione genera facilmente decine di gigabyte di traffico orario, esaurendo rapidamente i piani tariffari cellulari commerciali e incrementando esponenzialmente le tariffe di *ingestion* richieste dai provider cloud. Infine, espone il sistema a gravi vulnerabilità operative in aree con scarsa copertura di rete. L'ispezione di infrastrutture schermate o remote, come viadotti in cemento armato, gallerie ferroviarie o piattaforme *offshore*, è soggetta a improvvisi cali di segnale cellulare che disconnetterebbero il velivolo dal motore di inferenza semantica, rendendo il sistema temporaneamente cieco [4].

Al fine di cercare di risolvere queste problematiche, il presente elaborato propone la progettazione e la prototipazione di una piattaforma informatica basata sul paradigma ibrido Cloud-Edge. L'idea centrale è realizzare un'infrastruttura capace di separare e orchestrare in modo efficiente due specifiche tipologie di traffico dati. Da un lato vi sono i cosiddetti "flussi leggeri", che comprendono la telemetria continua del drone (dati di navigazione e diagnostica). Dall'altro lato

si collocano i "flussi pesanti", costituiti dai fotogrammi ad alta risoluzione e dalle sequenze video estratte dal *payload* ambientale, che vengono isolati e gestiti in modo strettamente asincrono. Come mostrato in Figura 1.1, il sistema delega i carichi di lavoro computazionalmente intensivi, come l'inferenza semantica multimodale, a servizi cloud scalabili. Questa architettura di calcolo distribuito viene attivata esclusivamente al verificarsi di un trigger (*event-driven*), ottimizzando così il bilanciamento tra reattività al suolo, potenza elaborativa e sostenibilità economica (FinOps).

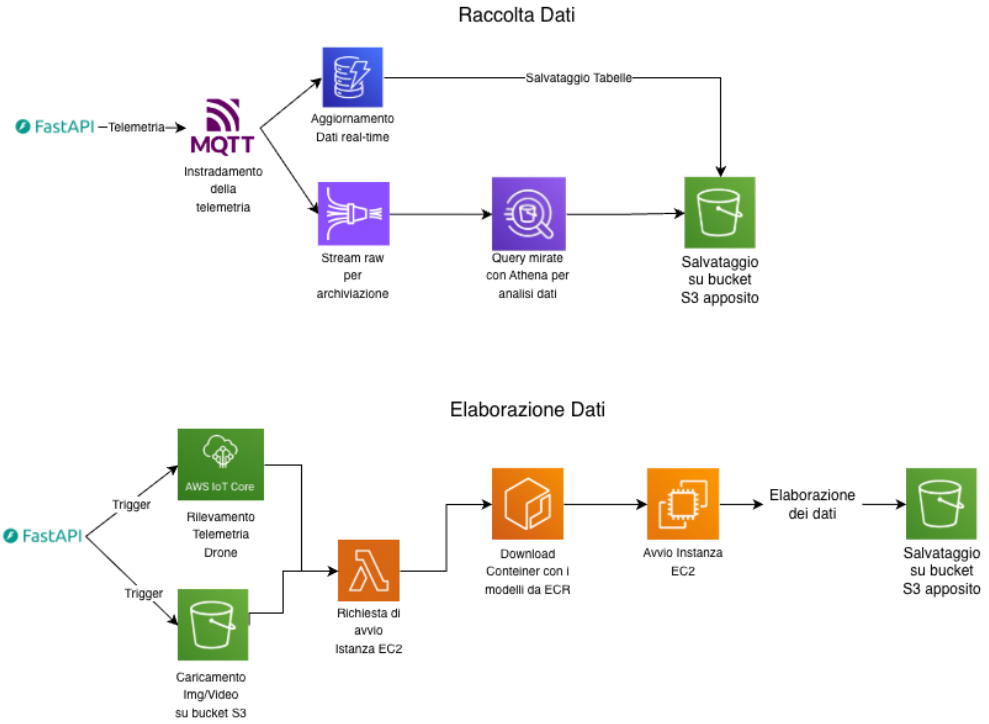


Figura 1.1: Schema generale dell'architettura Edge-to-Cloud event-driven per sistemi UAV.

1.1 Casistiche d'Uso Reali e Prevenzione dei Disastri Infrastrutturali

Negli ultimi anni, il monitoraggio automatizzato tramite UAV ha dimostrato un'efficacia tangibile nella prevenzione dei collassi strutturali e nella diagnostica di opere civili critiche, trovando applicazione diretta in settori quali la viabilità (ponti e viadotti), le infrastrutture ferroviarie e gli impianti energetici (dighe e reti di trasmissione). Come evidenziato dalla letteratura scientifica nel campo dello *Structural Health Monitoring* (SHM) [5], l'impiego dei droni sta progressivamente sostituendo i metodi tradizionali di ispezione basati sull'intervento diretto

di operatori su fune, cestelli elevatori o ponteggi temporanei. Questa transizione metodologica non solo mitiga drasticamente i rischi legati alla sicurezza umana in ambienti ostili o di difficile accesso, ma garantisce anche l'acquisizione di flussi di lavoro ripetibili, misurabili e ad altissima densità di dati.

Un caso di studio del 2025 ha evidenziato come un drone equipaggiato con sensori ad alta risoluzione abbia documentato l'intera struttura di un ponte in sole 2 ore, rispetto alle 8 ore necessarie a una squadra tradizionale su camion con cestello, registrando una riduzione del 75% del tempo impiegato sul campo [6].

L'analisi visiva guidata dall'Intelligenza Artificiale, nello specifico attraverso l'implementazione di reti neurali come le varianti del modello YOLO [7], è in grado di identificare micro-lesioni strutturali, comunemente note in letteratura con il termine *fatigue cracks*, nell'acciaio o nel calcestruzzo, individuando anomalie sub-millimetriche che sfuggirebbero all'occhio umano [8]. L'importanza di questa architettura emerge drasticamente negli scenari di gestione delle emergenze (*disaster response*). Come riportato in diverse rassegne sulle ispezioni civili automatizzate [9, 10, 11], l'impiego di sistemi UAV abbinati al *Deep Learning* permette di mappare i nodi strutturali critici (come i piloni dei ponti) nelle fasi di post-emergenza sismica. Questo approccio garantisce una valutazione dell'agibilità in poche ore anziché in giorni, svincolando le squadre di soccorso dai rischi di un'ispezione manuale e accelerando la ripresa sicura della viabilità [9]. Analogamente, in contesti costieri, l'uso di fotocamere multispettrali abbinate all'Edge AI permette di monitorare gradienti di ossidazione e velocità di corrosione salina invisibili alle fotocamere standard, fornendo alle municipalità dati diagnostici oggettivi per pianificare o posticipare interventi di restauro in totale sicurezza.

L'esigenza di un'analisi automatizzata in tempo reale diventa un requisito imprescindibile alla luce delle recenti normative europee sul volo dei droni. Il regolamento EASA (European Union Aviation Safety Agency) disciplina le operazioni in modalità BVLOS (*Beyond Visual Line of Sight*) [12]. In questi scenari, il pilota remoto perde il contatto visivo diretto con l'aeromobile, affidandosi unicamente alla telemetria e ai flussi video. Un'architettura in grado di analizzare proattivamente la scena, allertando l'operatore solo in caso di effettive anomalie strutturali o ostacoli imprevisti, si configura come una vera e propria "tecnologia abilitante". Essa rende tecnicamente e legalmente realizzabili operazioni prolungate altrimenti impossibili, automatizzando l'estrazione delle informazioni utili e trasformando l'operatore umano da osservatore passivo e ininterrotto dei flussi video a supervisore strategico degli eventi critici. Tale transizione metodologica è di importanza fondamentale poiché previene il sovraccarico cognitivo (*cognitive overload*) del pilota remoto [13]. Nelle missioni BVLOS, infatti, l'esame continuo di molteplici monitor porta a un rapido degrado dell'attenzione e a una severa perdita della *situational awareness* (consapevolezza situazionale). Delegando l'onere dell'ispezione visiva continua all'Intelligenza Artificiale, si garantisce il livello di affidabilità e sicurezza necessario per l'integrazione definitiva degli UAV nello spazio aereo civile condiviso.

1.2 Motivazioni e Obiettivi del Progetto

Dal punto di vista tecnologico, l'idea deriva dalla necessità di risolvere la limitata capacità di calcolo a bordo del velivolo e i costi insostenibili di una trasmissione video continua. Esiste una zona grigia tra le potenzialità teoriche dei moderni modelli multimodali e la loro effettiva messa in produzione su macchine con vincoli energetici severi. Risolvere questa lacuna attraverso l'automazione infrastrutturale significa rendere finalmente scalabili e sostenibili le ispezioni visive automatizzate.

Dal punto di vista accademico, il progetto nasce dalla forte volontà di applicare sul campo le competenze multidisciplinari maturate nel corso di laurea. L'intento è creare un ponte tangibile tra il mondo dell'automazione robotica (il nodo UAV) e le architetture informatiche distribuite (il Cloud Computing), ingegnerizzando una soluzione su Amazon Web Services (AWS) che rispetti rigorosi criteri di efficienza.

L'obiettivo principale del presente lavoro di tesi è progettare, implementare e collaudare un'architettura *cloud-native* per la gestione avanzata dei flussi UAV e l'inferenza semantica distribuita, valutandone analiticamente le prestazioni, la resilienza e l'efficienza economica.

Nel dettaglio, lo studio mira a raggiungere i seguenti obiettivi specifici:

- L'acquisizione e la trasmissione sicura della telemetria tramite protocollo MQTT avvalendosi del broker gestito AWS IoT Core, garantendo l'autenticazione tramite certificati X.509.

- L'archiviazione dello stato corrente del drone su un database NoSQL a bassa latenza (Amazon DynamoDB) per consentire l'alimentazione di interfacce e *dashboard* operative. Questo approccio permette alle applicazioni client di interrogare lo stato della flotta in tempo reale, disaccoppiando la lettura dei dati dal flusso di pubblicazione sul broker MQTT.

- Il caricamento asincrono e scalabile degli *asset* visivi all'interno di un *Data Lake* basato su Amazon S3, ovvero un repository di archiviazione centralizzato progettato per accogliere enormi volumi di dati grezzi nella loro forma nativa [14]. Questo processo sfrutta URL pre-firmati (*pre-signed URL*), permettendo al drone di caricare i file direttamente nello *storage* e aggirando i classici colli di bottiglia dei backend tradizionali causati dal transito sincrono di file multimediali pesanti tramite standard HTTP.

- L'azzeramento del traffico di rete cloud in stato di inattività tramite l'impiego di code Amazon SQS configurate in modalità *Long Polling*. Questo paradigma mantiene la connessione aperta fino all'effettivo arrivo di un messaggio o allo scadere di un *timeout* preimpostato (fino a 20 secondi), svincolando i microservizi dall'infrastruttura di ricezione continua e riducendo la latenza complessiva alla comparsa dell'evento.

- L'elaborazione automatica delle immagini tramite container Docker isolati gestiti su Amazon ECR. La *pipeline* visiva impiega un'inferenza a cascata, utilizzando YOLO11n per il rilevamento spaziale a bassa latenza e il modello multimodale Qwen2-VL per la descrizione semantica, eseguiti su istanze di calcolo Amazon EC2 appartenenti alla famiglia `g4dn` ed equipaggiate con acceleratori grafici NVIDIA Tesla T4.

- L'automazione dello spegnimento dell'hardware tramite il monitoraggio delle metriche di traffico di Amazon CloudWatch. Questo sistema di *de-provisioning* intelligente garantisce un rigoroso contenimento dei costi di esecuzione cloud.

- L'acquisizione continua dei log storici e dei risultati inferenziali tramite Kinesis Data Firehose, organizzandoli per consentirne l'interrogazione *on-demand* e *serverless* tramite Amazon Athena. Il paradigma senza server esonera lo sviluppatore dal partizionamento dell'infrastruttura sottostante, permettendo di interrogare i dati grezzi scrivendo semplici *query* in linguaggio SQL standard [15].

1.3 Struttura della Tesi

La trattazione si articola nei seguenti capitoli, strutturati in modo logico e consequenziale per illustrare i fondamenti teorici, le fasi di progettazione e la validazione empirica del prototipo:

- **Capitolo 2 - Stato dell'Arte:** Analizza il contesto tecnologico di riferimento, esplorando l'evoluzione dei paradigmi Cloud ed Edge Computing per l'IoT e lo sviluppo dei modelli di Intelligenza Artificiale per l'analisi visiva (dalle architetture *Single-Stage* come YOLO ai recenti *Vision-Language Models*).

- **Capitolo 3 - Tecnologie Utilizzate e Architettura Software:** Introduce il set di strumenti di sviluppo (Python, FastAPI, Docker) e la vasta gamma di servizi AWS impiegati, delineando la macro-architettura del sistema e le giustificazioni ingegneristiche dietro ogni scelta tecnologica.

- **Capitolo 4 - Progettazione e Sviluppo:** Descrive analiticamente l'implementazione tecnica della *pipeline event-driven*. Rappresentando il nucleo ingegneristico dello studio, questa sezione motiva dettagliatamente le scelte architetturali più complesse, quali la transizione a SQS per il disaccoppiamento interno e l'automazione dei *trigger* hardware per i nodi GPU.

- **Capitolo 5 - Risultati:** Valuta le performance del prototipo esaminando i log operativi, i tempi di latenza dell'inferenza visiva e l'effettivo risparmio economico ottenuto tramite la configurazione degli allarmi CloudWatch.
- **Capitolo 6 - Conclusioni:** Sintetizza il raggiungimento degli obiettivi prefissati, evidenziando le limitazioni strutturali dell'attuale versione e proponendo concrete traiettorie per lo sviluppo futuro del progetto.

Capitolo 2

Stato dell'Arte

Il presente capitolo delinea il contesto tecnologico e scientifico in cui si colloca l'architettura ingegnerizzata in questo progetto. Verranno esplorate l'evoluzione delle architetture di calcolo distribuito, le sfide intrinseche dei sistemi UAV e le fondamenta teoriche dell'Intelligenza Artificiale applicata all'analisi visiva, al fine di giustificare in modo logico e consequenziale le scelte progettuali adottate nei capitoli successivi.

2.1 Evoluzione verso il Paradigma Edge-Cloud nei Sistemi UAV

Il paradigma alla base del Cloud Computing affonda le sue radici concettuali nel 1961, quando l'informatico John McCarthy ipotizzò che la potenza di calcolo potesse un giorno essere erogata come un servizio pubblico (*utility computing*) [16]. Durante gli anni '90, questa visione iniziò a prendere forma attraverso il *Grid Computing*, che permetteva di distribuire carichi di lavoro computazionali su reti di macchine geograficamente distanti, seppur presentando ancora limiti significativi in termini di standardizzazione dei protocolli e sicurezza dei dati [17].

Il definitivo punto di svolta per l'infrastruttura informatica moderna è avvenuto a metà degli anni 2000, segnando il passaggio dai modelli *on-premise*, basati sull'acquisto e il mantenimento di server fisici locali, al modello *Pay-as-You-Go*. Questo approccio ha abbattuto i costi di capitale a favore delle spese operative, garantendo una capacità di allocazione dinamica delle risorse computazionali su larga scala. Nel corso degli anni, l'offerta Cloud si è stratificata in servizi IaaS (*Infrastructure as a Service*), PaaS (*Platform as a Service*) e SaaS (*Software as a Service*), evolvendo recentemente nel modello *Serverless*, in cui l'infrastruttura esegue porzioni di codice esclusivamente in risposta a specifici *trigger* (approccio *event-driven*), sollevando lo sviluppatore dalla gestione dei server sempre attivi.

Parallelamente, l'evoluzione delle telecomunicazioni ha favorito l'espansione dell'Internet of Things (IoT). Nelle sue prime fasi, l'infrastruttura IoT basava la propria operatività su modelli architetturali strettamente centralizzati; i nodi periferici acquisivano i dati e li inviavano integralmente ai data center per l'elaborazione. Tuttavia, con l'impiego di droni all'interno di reti di monitoraggio critico (come l'ispezione di reti elettriche e infrastrutture civili), questo approccio ha rivelato limiti stringenti [4]. Gli UAV moderni generano infatti flussi video ad altissima risoluzione e dati telemetrici ad alta frequenza; trasmettere incondizionatamente questa mole di informazioni verso server remoti comporta latenze insostenibili sulle reti mobili, elevati costi operativi legati al consumo di banda (*egress costs*) e gravi vulnerabilità in aree geografiche remote prive di copertura cellulare.

Oltre ai limiti di rete, i velivoli UAV sono soggetti a severi vincoli fisici. L'esecuzione locale di modelli di *Deep Learning* da miliardi di parametri richiederebbe l'installazione a bordo di *System-on-Chip* (SoC), ovvero circuiti integrati complessi che racchiudono processore centrale, memoria e acceleratore grafico in un unico microchip. Sebbene performanti, dispositivi come l'NVIDIA Jetson introducono un carico utile (*payload*) aggiuntivo e assorbimenti elettrici incompatibili con la portabilità del mezzo. Fisicamente, ogni incremento della massa e dell'assorbimento elettrico sul drone incide negativamente sull'autonomia di volo, riducendo drasticamente i tempi operativi e compromettendo la missione [18].

Per ovviare a queste problematiche, la letteratura scientifica recente ha formalizzato i limiti del calcolo interamente cloud-centrico, promuovendo il paradigma dell'*Edge Computing* [19]. L'architettura *Edge-Cloud Collaborative* sviluppata nel presente progetto si fonda su questi recenti *framework* accademici [20]. Il drone agisce come nodo sensoriale periferico (Edge) instradando in tempo reale solo i dati a basso *bitrate* (come la telemetria continua), mentre l'elaborazione dei flussi video pesanti viene delegata in modo asincrono alle istanze GPU del Cloud, aggirando così i limiti hardware del velivolo.

2.2 Evoluzione dell'Analisi Visiva: Dalla Computer Vision all'Object Detection

L'impiego degli UAV per le ispezioni ha generato un volume di informazioni visive tale da rendere obsolete le tradizionali metodologie di analisi. Storicamente, l'estrazione di informazioni dai flussi video aerei era un processo manuale in cui operatori umani visionavano ore di riprese per identificare anomalie. I primi tentativi di automazione si basavano su algoritmi di *Computer Vision* deterministici, come l'algoritmo *Canny edge detection* o l'estrazione di *feature* locali quali SIFT (*Scale-Invariant Feature Transform*) e SURF (*Speeded Up Robust Features*). Tali sistemi, sebbene efficaci in ambienti controllati, fallivano dinanzi alla naturale

variabilità del mondo reale (cambi di illuminazione, occlusioni o instabilità della telecamera), producendo un numero inaccettabile di falsi positivi [21].

L'approccio all'analisi visiva è stato radicalmente trasformato dall'introduzione delle reti neurali artificiali profonde (*Deep Learning*). L'efficacia di questa metodologia è stata sancita nel 2012 dal successo dell'architettura AlexNet, che ha dimostrato scientificamente l'enorme potenziale dell'addestramento distribuito su acceleratori GPU per compiti di classificazione complessi [22]. Per l'elaborazione di dati bidimensionali, l'architettura di riferimento è divenuta la Rete Neurale Convoluzionale (CNN). Una CNN impiega un'operazione matematica di convoluzione in cui filtri (*kernel*) scorrono sull'immagine per estrarne in modo gerarchico caratteristiche spazialmente invarianti. L'utilizzo della condivisione dei pesi (*weight sharing*) permette di ridurre i parametri da ottimizzare, mitigando il rischio di *overfitting* e incrementando l'efficienza computazionale [23].

L'integrazione delle CNN nei droni ha richiesto modelli sempre più rapidi. Fino a pochi anni fa, il rilevamento degli oggetti (*Object Detection*) si affidava a reti a due stadi come *Faster R-CNN* [24], eccessivamente lente per l'uso in tempo reale. L'introduzione della famiglia YOLO (*You Only Look Once*) ha rivoluzionato il settore unificando la localizzazione dei *bounding box* e la classificazione in un'unica elaborazione *feed-forward* [7]. YOLO discretizza l'immagine in una griglia, assegnando la responsabilità della predizione alla cella in cui ricade il centro dell'oggetto. Matematicamente, l'affidabilità della previsione è espressa dal punteggio di confidenza

$$\text{Confidence} = \Pr(\text{Object}) \times \text{IoU}_{\text{pred}}^{\text{truth}}$$

dove l'IoU (*Intersection over Union*) quantifica la sovrapposizione geometrica tra l'area stimata e quella reale [7]. Negli scenari operativi UAV odierni, versioni ottimizzate come YOLOv8-Nano e YOLOv10-Nano hanno dimostrato compromessi ideali tra velocità di inferenza e occupazione di memoria su *dataset* aerei di riferimento, quali VisDrone [25]. Alla luce di questa evoluzione tecnologica, il presente lavoro di tesi propone un'architettura di analisi strutturata in due fasi operative sequenziali, concepita per ottimizzare le risorse computazionali. Il flusso di elaborazione (*pipeline*) prevede l'impiego del modello YOLO11n come primo stadio di rilevamento spaziale ad alta velocità. Questo componente ha il compito di ispezionare i fotogrammi grezzi, isolare istantaneamente le coordinate di reale interesse e scartare il rumore di fondo visivo. Solamente i ritagli di immagine (*crop*) contenenti i bersagli validati vengono successivamente inoltrati a un secondo stadio di natura semantica, governato da un modello multimodale avanzato (il Vision-Language Model Qwen2-VL). Tale approccio a cascata garantisce che le elaborazioni più onerose vengano innescate esclusivamente sulle frazioni di scena rilevanti, gettando le basi per l'architettura Cloud-Edge che verrà dettagliata nei capitoli successivi.

2.3 Comprensione Semantica e Modelli Multimodali (VLM)

Mentre i modelli di rilevamento spaziale come YOLO forniscono esclusivamente coordinate e classi isolate, la necessità di redigere automaticamente referti ispettivi complessi ha spinto la ricerca verso l'integrazione del linguaggio naturale. Un avanzamento metodologico decisivo è avvenuto con l'affermazione dell'architettura *Transformer*. Introdotta da Vaswani et al. [26], questa architettura si basa su meccanismi di attenzione per elaborare sequenze di dati, ponendo le basi per i moderni *Large Language Models* (LLM).

Per estendere tali capacità percettive al dominio visivo, la ricerca ha concepito i *Vision-Language Models* (VLM). Questi sistemi associano un *Vision Encoder* a un LLM decodificatore, impiegando moduli di allineamento per proiettare le informazioni visive nel medesimo spazio vettoriale dei *token* testuali [27]. La letteratura recente documenta lo sviluppo di numerose varianti. Architetture pionieristiche come CLIP allineano semanticamente immagini e testo, GPT-4V permette ragionamenti complessi su input ad alta risoluzione, mentre LLaVA adotta un approccio efficiente sfruttando proiezioni lineari per connettere encoder visivi a LLM open-source [28]. L'implementazione di queste reti sui flussi UAV rappresenta l'attuale frontiera della ricerca, permettendo di interrogare le scene riprese direttamente in linguaggio naturale.

Nella progettazione della piattaforma oggetto di questa tesi, il ruolo di motore inferenziale semantico è stato assegnato al modello multimodale Qwen2-VL-2B-Instruct [29]. La scelta di questa specifica variante dimensionale (2 miliardi di parametri) rappresenta un compromesso ingegneristico ottimale tra l'avanguardia algoritmica del modello, tra i più recenti e performanti allo stato dell'arte, e i rigorosi vincoli infrastrutturali dell'architettura. L'adozione di varianti dimensionali maggiori avrebbe infatti inevitabilmente saturato i 16 GB di VRAM dell'acceleratore Tesla T4, rendendo impossibile l'allocazione dinamica della cache KV (*Key-Value*) e forzando la migrazione verso istanze Cloud di fascia superiore, con un inaccettabile incremento dei costi operativi (FinOps).

Ricevendo in input i ritagli precedentemente isolati da YOLO, tale modello non si limita alla banale classificazione, ma deduce attivamente il contesto strutturale della scena. L'innovazione tecnologica che ne giustifica le altissime prestazioni, pur operando su un nodo computazionale così vincolato, è il meccanismo M-RoPE (*Multimodal Rotary Position Embedding*). A differenza delle architetture precedenti che perdevano le relazioni spaziali appiattendo l'immagine in matrici monodimensionali, M-RoPE scompone le informazioni posizionali in componenti temporali, spaziali orizzontali e verticali [29]. Questo permette alla rete di preservare fedelmente le relazioni geometriche degli elementi ispezionati e di elaborare reportistica testuale altamente dettagliata, abilitando la completa automazione del processo di monitoraggio ingegneristico.

Capitolo 3

Strumenti, Tecnologie e Architettura di Sistema

La realizzazione di una piattaforma distribuita Cloud-Edge richiede la selezione accurata di uno *stack* tecnologico capace di garantire sicurezza, disaccoppiamento, scalabilità e contenimento dei costi. L'intero sistema è stato sviluppato avvalendosi dei servizi gestiti di Amazon Web Services (AWS) e framework *open-source* moderni, architettando la *pipeline* logica su tre macro-livelli: Acquisizione, Elaborazione e Persistenza.

Come illustrato nello schema architetturale completo (Figura 3.1), il flusso operativo si biforca fin dalla sorgente. Il nodo Edge (l'UAV) trasmette la telemetria tramite protocollo MQTT ad AWS IoT Core, che ne storicizza lo stato su Amazon DynamoDB. Parallelamente, l'operatore web interagisce con il backend FastAPI per ottenere i permessi necessari al caricamento diretto dei flussi multimediali su Amazon S3. Tale caricamento genera un evento (*ObjectCreated*) che popola la coda Amazon SQS, la quale a sua volta alimenta, tramite *Long Polling*, i nodi di elaborazione Amazon EC2. All'interno di questi ultimi, un ambiente isolato Docker orchestra l'inferenza a cascata sui modelli YOLO11n e Qwen2-VL, i cui risultati vengono salvati nuovamente su S3. L'intero ciclo di vita delle istanze di calcolo è infine supervisionato attivamente da Amazon CloudWatch per imporre le policy di spegnimento automatico.

3.1 Livello di Acquisizione e Trasmissione (Ingestion Layer)

Il livello di acquisizione ha la responsabilità di porsi come ponte bidirezionale tra il mondo fisico (il velivolo e l'operatore) e l'infrastruttura Cloud. Il sistema differenzia intrinsecamente le tecnologie di ricezione in base al *payload* del dato. Da un lato vi sono i pacchetti telemetrici, ovvero stringhe di testo di pochi byte contenenti coordinate GPS, orientamento spaziale IMU e stato della batteria.

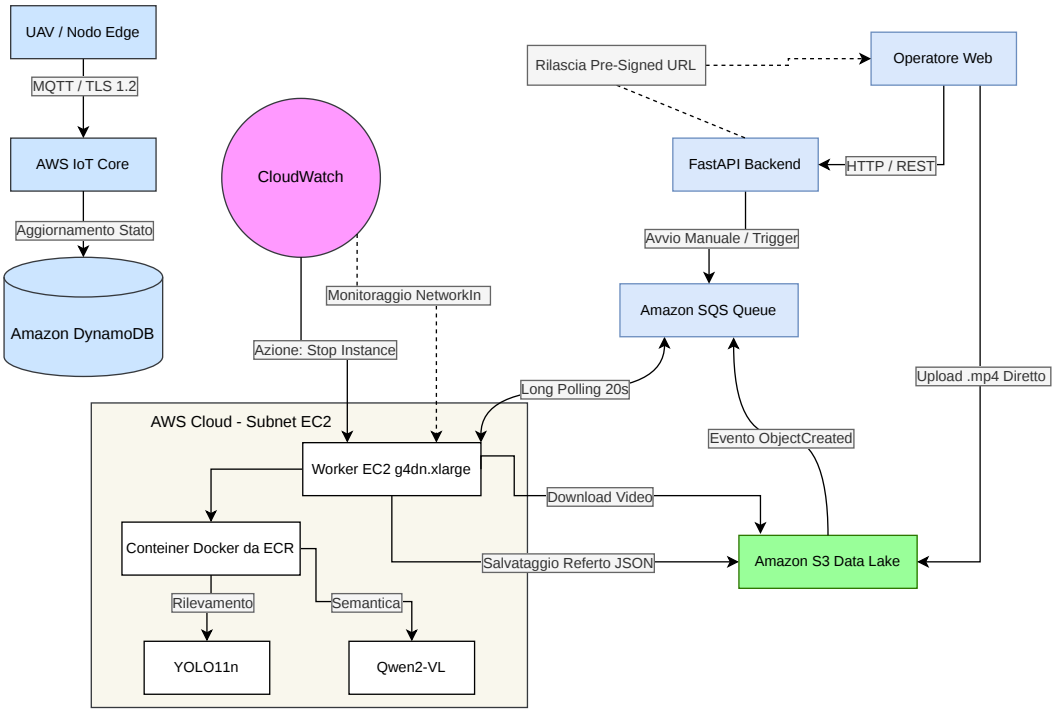


Figura 3.1: Schema architetturale dell'infrastruttura Cloud-Edge: i flussi telemetrici e multimediali generati dall'UAV (sinistra) vengono disaccoppiati logicamente e instradati verso i rispettivi livelli di acquisizione (IoT Core/S3), elaborazione asincrona (SQS/EC2) e archiviazione (DynamoDB/Athena).

Dall'altro lato si trovano i flussi multimediali composti da fotogrammi e video ad alta risoluzione, il cui peso impone canali di trasmissione ad alta capacità.

3.1.1 Cruscotto Web e Backend FastAPI

L'interfaccia di comando per l'operatore, denominata *Mission Control Dashboard*, è stata progettata e sviluppata appositamente per questo progetto. Lato client, l'applicativo è strutturato come una *Single Page Application* (SPA). Una SPA è un'applicazione web che carica un singolo documento HTML e aggiorna dinamicamente solo i frammenti di pagina necessari tramite JavaScript, offrendo un'esperienza utente fluida e priva dei tempi morti tipici del ricaricamento completo delle pagine tradizionali.

Il *backend* della dashboard è stato sviluppato in linguaggio Python impiegando FastAPI. Questo framework è stato selezionato per le sue altissime prestazioni, paragonabili a quelle dei server NodeJS e Go [30]. Tali performance derivano dall'implementazione dello standard ASGI (*Asynchronous Server Gateway Interface*) tramite il server *Uvicorn* che, a differenza dei server tradizionali

sincroni, gestisce le richieste in modo non bloccante (*non-blocking concurrency*), permettendo al sistema di gestire migliaia di connessioni simultanee.

Il *backend* espone specifiche API REST (interfacce di programmazione) fondamentali per l'operatività del sistema:

- **Simulazione telemetrica:** Un *endpoint* che inietta dati di volo fittizi nel sistema, permettendo di collaudare la tenuta dell'infrastruttura logica senza dover fisicamente lanciare in volo un drone.
- **Pre-Warm hardware:** Una rotta API che ordina preventivamente l'accensione delle macchine virtuali Cloud e il caricamento delle reti neurali in memoria prima che il drone invii le immagini, azzerando i ritardi operativi (*cold-start*).
- **Autorizzazione al caricamento:** Un *endpoint* crittografico che fornisce al drone un lasciapassare temporaneo per depositare i video in Cloud in totale sicurezza.

3.1.2 Il protocollo MQTT e AWS IoT Core

La trasmissione in tempo reale dei dati cinematici dell'UAV è governata dal protocollo MQTT (*Message Queuing Telemetry Transport*). A differenza del classico protocollo HTTP basato su un rigido meccanismo di *request/response*, MQTT utilizza un paradigma *publish/subscribe*. Il drone "pubblica" la propria posizione su un canale tematico (denominato *topic*) e chiunque sia interessato a quei dati (come il cruscotto di controllo) si "iscrive" al canale per riceverli passivamente, senza dover interrogare continuamente il server.

Il ruolo di *Broker* centrale, ovvero l'ente che riceve i messaggi e li smista agli iscritti, è affidato ad AWS IoT Core. Questo servizio gestito scala automaticamente per supportare milioni di dispositivi connessi. AWS IoT Core è stato impiegato nel progetto per due funzioni cardine, ovvero garantire la sicurezza delle connessioni (verificando l'identità del drone) e applicare regole logiche di instradamento (*Rules Engine*). Il Rules Engine, infatti, intercetta il traffico MQTT e utilizza istruzioni simil-SQL per salvare automaticamente la posizione del drone nel database, senza richiedere la scrittura di codice di intermezzo.

3.2 Analisi Comparativa dei Protocolli IoT e Containerizzazione

La selezione del protocollo MQTT per la telemetria è dettata da severi vincoli di rete riscontrabili nell'aviazione *unmanned*. Diverse misurazioni presentate in studi accademici [31, 32] dimostrano che l'overhead di un'intestazione HTTP standard supera spesso i 400 byte, contro i soli 2 byte fissi dell'intestazione MQTT. In

contesti operativi remoti dove la banda passante cellulare si degrada a pochissimi Kilobit per secondo (Kbps), l'uso di HTTP causerebbe congestioni. MQTT, al contrario, offre livelli di garanzia di consegna chiamati *Quality of Service* (QoS). Sfruttando il QoS 1 (*At least once*) o il QoS 2 (*Exactly once*), il sistema assicura che il pacchetto telemetrico arrivi a destinazione anche a fronte di gravissime instabilità radio, memorizzando il messaggio e ritentandone l'invio alla prima riconnessione utile.

Parallelamente, la distribuzione dell'applicativo sul nodo di inferenza si avvale della containerizzazione Docker. Questa scelta risponde alla necessità di isolare rigidamente l'ambiente di runtime del codice dalle configurazioni del sistema operativo sottostante. La letteratura documenta come i progetti basati sul *Deep Learning* siano proni al cosiddetto *dependency hell* [33], ovvero conflitti tra versioni incompatibili di driver NVIDIA CUDA, librerie PyTorch e pacchetti Python che portano al blocco del software. Containerizzando la *pipeline*, le dipendenze di YOLO11n e Qwen2-VL risultano logicamente incapsulate, garantendo la totale immutabilità e riproducibilità dell'ambiente in Cloud.

3.2.1 Amazon S3 e Upload Diretto tramite Pre-signed URL

I flussi video catturati bypassano deliberatamente il server FastAPI per impedire l'esaurimento della banda passante del backend. Il sistema adotta il *pattern* architetturale dei Pre-signed URL. Il client richiede ad AWS un token crittografico con scadenza temporale (es. 60 secondi); una volta generata la firma, il drone esegue un *upload* diretto verso Amazon S3 (Simple Storage Service) [34]. S3 agisce come *Data Lake* transitorio e innesca la catena *event-driven*. L'avvenuto caricamento del file fa scattare l'accodamento del *task* su Amazon SQS e la successiva elaborazione neurale su Amazon EC2.

3.3 Livello di Elaborazione e Disaccoppiamento (Processing Layer)

Il livello di elaborazione costituisce il nucleo operativo dell'infrastruttura, poiché assorbe interamente il carico computazionale dell'inferenza spaziale e semantica. La sua architettura deve scalare per gestire picchi di ispezione imprevisti, garantendo contemporaneamente che nessuna risorsa venga fatturata nei momenti di inattività (*idle*).

3.3.1 Amazon SQS (Simple Queue Service)

Per garantire il disaccoppiamento logico e temporale tra la fase di acquisizione dei dati e l'elaborazione intensiva su GPU, il sistema si avvale di Amazon SQS.

Tale servizio di messaggistica immagazzina in modo asincrono e duraturo i *task* pendenti, agendo da *buffer* elastico per assorbire eventuali picchi di carico. I dettagli architetturali relativi all'implementazione del paradigma di *Long Polling* e alle logiche di orchestrazione dei messaggi saranno oggetto di trattazione approfondita nel Capitolo 4.

3.3.2 AWS Lambda e Orchestrazione Event-Driven

L'orchestrazione dei flussi asincroni è governata da AWS Lambda. Lambda è un servizio di calcolo *Function-as-a-Service* (FaaS) che permette di eseguire porzioni di codice logico in risposta a determinati eventi, senza allocare o gestire alcun server virtuale. Nel progetto, una funzione Lambda viene istanziata per generare le autorizzazioni di S3, validare i formati in ingresso e compilare i messaggi da destinare alle code SQS, agendo da vero e proprio controllore di volo del traffico dati.

3.3.3 Istanze EC2 GPU e Amazon ECR

L'elaborazione delle reti neurali avviene su istanze Amazon EC2 della famiglia g4dn.xlarge. Queste macchine integrano acceleratori grafici NVIDIA Tesla T4 equipaggiati con *Tensor Cores*, componenti hardware specializzati che accelerano esponenzialmente le moltiplicazioni di matrici e permettono l'inferenza a precisione mista (FP16/INT8), vitale per eseguire modelli massivi come Qwen2-VL a velocità accettabili.

Per garantire la riproducibilità, il sistema operativo delle istanze non viene mai manipolato manualmente. Le immagini Docker, complete dei pesi delle reti neurali, risiedono in Amazon ECR (*Elastic Container Registry*) ossia il registro privato e protetto di AWS per i container. Al momento del *Pre-Warm*, ovvero la fase di riscaldamento preventivo del server, le cui logiche architetturali verranno dettagliate nella Sezione 4.1, l'istanza autentica la propria identità tramite il framework IAM (*Identity and Access Management*), scarica il container da ECR e demanda il calcolo vettoriale direttamente all'hardware NVIDIA.

3.4 Livello di Persistenza e Analisi (Storage & Analytics Layer)

I dati prodotti vengono instradati in tecnologie di archiviazione eterogenee, differenziate in base alla latenza richiesta in fase di lettura.

3.4.1 Stato in Tempo Reale: Amazon DynamoDB

Per alimentare la *dashboard* di monitoraggio con la posizione istantanea dell'UAV, è indispensabile conoscere l'ultimo stato noto del drone con latenze bassissime. Si impiega pertanto Amazon DynamoDB, un database NoSQL serverless progettato per garantire tempi di risposta a singola cifra di millisecondo. DynamoDB memorizza lo stato transiente della missione sovrascrivendo l'ultima entry per ciascun `drone_id`.

3.4.2 Kinesis Firehose e Amazon Athena

La totalità dei referti semantici e della telemetria storica confluisce nel *Data Lake* principale su S3, un *repository* centralizzato capace di archiviare moli infinite di dati nel loro formato originale (JSON). L'aggregazione e la consegna di questi file è operata in modo continuo da Amazon Kinesis Data Firehose.

Per estrarre conoscenze strutturate ed effettuare *query* sui voli passati, il sistema sfrutta Amazon Athena. Athena adotta un paradigma denominato *Schema-on-Read* il quale, diversamente dai database SQL tradizionali che impongono di formattare i dati prima del salvataggio (*Schema-on-Write*), proietta una struttura a tabelle sui file grezzi di S3 soltanto nel momento in cui l'operatore esegue l'interrogazione, fatturando l'operazione in base ai byte effettivamente analizzati.

3.5 Monitoraggio Innescato e Sicurezza Crittografica

Il controllo granulare dell'infrastruttura è affidato ad Amazon CloudWatch. Questo servizio raccoglie dati operativi in tempo reale sotto forma di log, metriche ed eventi. Nel contesto dell'architettura proposta, CloudWatch funge da supervisore attivo. Esso non si limita alla visualizzazione grafica del consumo di risorse (CPU, Rete, Disco), ma è in grado di innescare allarmi e azioni reattive per modificare fisicamente lo stato delle macchine EC2, introducendo logiche di automazione infrastrutturale (*Infrastructure as Code*). Nel caso specifico è configurato per innescare script di spegnimento delle macchine EC2 qualora il traffico visivo si arresti, introducendo una solida automazione FinOps.

Infine, trattando ispezioni potenzialmente critiche, la *pipeline* implementa rigidi standard di crittografia, classificabili in due categorie conformi alle direttive del *National Institute of Standards and Technology* (NIST).

- **Data in Transit (Dati in transito):** Il flusso di telemetria IoT viaggia cifrato tramite TLS 1.2 (*Transport Layer Security*), convalidato dall'autenticazione mutua X.509 [35]. Le API FastAPI e il traffico verso S3 sono parimenti blindati sul canale HTTPS/TLS 1.3, proteggendo le informazioni da tentativi di intercettazione (*man-in-the-middle*).

- **Data at Rest (Dati a riposo):** Le immagini, i video e i referti residenti sul Data Lake S3 vengono crittografati lato server tramite lo standard SSE-S3 (*Server-Side Encryption*), impiegando l'algoritmo a chiave simmetrica AES-256 gestito intrinsecamente dall'infrastruttura AWS [36].

Capitolo 4

Progettazione e Sviluppo

Il presente capitolo documenta le fasi di ingegnerizzazione e implementazione del prototipo. L'obiettivo primario di questa fase è stato tradurre i requisiti teorici in un'architettura software funzionante, ponendo particolare enfasi sulla creazione di una *pipeline* interamente *event-driven*. Tale paradigma permette al sistema di scalare in modo autonomo e di minimizzare i costi operativi in assenza di carico computazionale. Applicando i principi di ottimizzazione FinOps, l'infrastruttura provvede allo spegnimento automatico dei costosi nodi GPU al termine delle ispezioni, evitando l'addebito orario per risorse inattive [37].

4.1 Inizializzazione della Missione e Pre-Warm

Il ciclo di vita di una missione UAV inizia dal pannello operativo web. Come illustrato nella Figura 4.1, l'interfaccia denominata "Centro di Controllo UAV" fornisce all'operatore i comandi diretti per orchestrare sia il flusso dei dati, tramite la sezione "Connessione Telemetrica", sia lo stato dell'hardware tramite la sezione "Inizializzazione Infrastruttura".

Al fine di ottimizzare i tempi di risposta complessivi, l'architettura prevede che le procedure di preparazione fisica del drone e l'allocazione delle risorse Cloud avvengano in stretta esecuzione parallela. Prima ancora che il velivolo effettui il decollo e inizi a trasmettere le immagini, l'operatore preme il pulsante "Accendi Sistema (Pre-Warm)". Questa azione innesca una chiamata REST che il backend in FastAPI riceve e traduce in un comando diretto (eseguito tramite le librerie AWS SDK) per risvegliare l'istanza EC2 g4dn.xlarge designata all'inferenza.

Questa operazione di accensione preventiva si rende ingegneristicamente indispensabile per mascherare le latenze di inizializzazione (*initialization overhead*) intrinseche all'uso di modelli di *Deep Learning* massivi. L'avvio di una macchina virtuale isolata, il caricamento (*pull*) dei container aggiornati da Amazon ECR e il trasferimento di svariati gigabyte di pesi neurali (YOLO11n e Qwen2-VL) dal-

lo *storage* alla memoria VRAM dell'acceleratore grafico richiedono tempistiche nell'ordine di alcuni minuti [38].

Se questa gravosa procedura di allocazione venisse innescata in modo puramente reattivo, ovvero solo all'arrivo del primo video dal drone, il sistema subirebbe un ritardo inaccettabile, perdendo il requisito fondamentale dell'analisi in tempo reale. Sfruttando l'innesco parallelo del *Pre-Warm*, invece, l'infrastruttura completa la fase di caricamento in *background* mentre il velivolo raggiunge l'altitudine operativa. In questo modo, l'istanza Cloud risulta già attiva e i modelli matematici sono pienamente residenti in memoria nell'esatto istante in cui il primo fotogramma raggiunge il *Data Lake*.



Figura 4.1: Cruscotto operativo (Mission Control Dashboard) per l'interazione con l'architettura event-driven.

4.2 Gestione Asincrona del Payload Visivo

Una delle sfide architetturali più critiche ha riguardato il trasferimento dei file multimediali dal nodo Edge all'infrastruttura Cloud, garantendo che tale operazione non portasse alla saturazione della banda passante del server di controllo. In un'architettura web tradizionale, l'inoltro di *asset* multimediali prevede che il client invii il video direttamente al *backend* applicativo. Questa operazione avviene solitamente attraverso la codifica del file binario in formato testo Base64, una conversione che introduce tuttavia un *overhead* critico incrementando la dimensione del *payload* di circa il 33% [39], oppure mediante richieste HTTP di

tipo *multipart/form-data*. In entrambi gli scenari, il server è costretto ad allocare memoria (RAM) per ricevere, frammentare e processare i blocchi di dati prima di poterli inoltrare verso lo *storage* finale. Questo paradigma trasforma il *backend* in un severo collo di bottiglia prestazionale. La ricezione sincrona di file pesanti causa infatti l'esaurimento dei *thread* di elaborazione del server (blocco dell'I/O), impedendo al sistema di gestire le richieste degli altri droni (in caso di utilizzo parallelo) e degradando drasticamente l'efficienza dell'infrastruttura.

Per superare queste limitazioni, la *pipeline* di *upload* è stata riprogettata implementando il *pattern* architetturale dei Pre-signed URL (URL pre-firmati). Questo meccanismo esclude totalmente il server dal transito dei file, delegando il trasferimento direttamente ai server di archiviazione distribuiti di Amazon. Il processo transazionale che regola questo schema di sicurezza si articola in tre fasi sequenziali:

- **Richiesta di autorizzazione:** Il client effettua una richiesta HTTP GET minimale al *backend* FastAPI, comunicando unicamente l'intenzione di caricare un file e specificandone i metadati (es. nome file e formato).
- **Firma crittografica (Signing):** Il *backend*, essendo già autenticato nel Cloud tramite un ruolo IAM (*Identity and Access Management*), invoca la libreria AWS SDK per Python (*Boto3*). Utilizzando l'algoritmo *AWS Signature Version 4*, il sistema genera un *hash* HMAC-SHA256 che combina le chiavi private del server con i dettagli della richiesta [40]. Il risultato è un URL univoco che incorpora la firma crittografica, il quale viene restituito al client. Per mitigare i rischi di sicurezza legati a potenziali intercettazioni (*token hijacking*), la validità temporale della firma è rigorosamente limitata a una finestra di 5 minuti.
- **Caricamento diretto (Direct Upload):** Sfruttando l'URL pre-firmato appena ricevuto, il client esegue una richiesta HTTP POST (o PUT) caricando il file `.mp4` direttamente all'interno del *bucket* Amazon S3 designato. Attraverso questo *handshake*, il transito del pesante flusso video avviene esclusivamente in modalità *peer-to-peer* tra l'UAV e lo *storage* Cloud, preservando la larghezza di banda e le risorse computazionali del *backend* FastAPI.

L'arrivo del file su S3 genera un evento `s3:ObjectCreated` che funge da innesco (*trigger*) per l'intera coda di elaborazione successiva.

4.3 Implementazione Backend: Generazione dei Pre-signed URL

Per implementare il caricamento diretto su Amazon S3 bypassando il backend, è stato sviluppato un *endpoint* dedicato all'interno dell'applicazione FastAPI. Il

codice sfrutta la libreria ufficiale AWS `boto3` per interfacciarsi con il servizio IAM e generare un URL crittografato.

Come si evince dallo Script 4.1, la funzione `generate_presigned_post` riceve in input il nome del file e imposta un parametro `ExpiresIn` configurato a 300 secondi (5 minuti). Questa finestra temporale è sufficiente per consentire al drone o al cruscotto web di avviare il caricamento del video, garantendo al contempo che l'URL non possa essere riutilizzato per scopi malevoli una volta scaduto.

```
import boto3
from fastapi import FastAPI, HTTPException
from botocore.exceptions import ClientError

app = FastAPI()
s3_client = boto3.client('s3', region_name='eu-south-1')
BUCKET_NAME = "drone-video-datalake"

@app.get("/api/upload-url")
async def get_presigned_url(filename: str):
    """Genera un URL temporaneo per l'upload diretto su S3"""
    try:
        response = s3_client.generate_presigned_post(
            Bucket=BUCKET_NAME,
            Key=f"inbox/{filename}",
            ExpiresIn=300 # Scadenza token: 5 minuti
        )
        return {"url_data": response}
    except ClientError as e:
        raise HTTPException(status_code=500, detail="Errore AWS IAM")
```

Snippet 4.1: Endpoint FastAPI per la generazione del Pre-signed URL.

4.4 Transizione Architetturale: Da MQTT ad Amazon SQS

Durante le prime fasi di prototipazione, la comunicazione tra i microservizi cloud, in particolare il meccanismo di notifica impiegato per segnalare all'istanza EC2 la disponibilità di un nuovo video da analizzare, era gestita interamente tramite il protocollo MQTT. Sebbene questa tecnologia risulti eccellente per il transito della telemetria in tempo reale, essa si fonda su sessioni TCP persistenti. Per impedire che i *firewall* o i dispositivi NAT (*Network Address Translation*) interrompano le connessioni inattive, lo standard MQTT impone l'invio costante di pacchetti di *Keep-Alive* [41]. Questo meccanismo di controllo obbliga il client (il *worker* Python residente su EC2) a trasmettere periodicamente messaggi di `PINGREQ`, ai quali il *Broker* (AWS IoT Core) risponde con messaggi di `PINGRESP`, garantendo così il mantenimento del *socket* di rete.

Tuttavia, questo fisiologico traffico di rete generato a riposo (*idle traffic*) inquinava irrimediabilmente le metriche di monitoraggio. La costante presenza di pacchetti di controllo, seppur leggerissimi, impediva agli algoritmi di supervisione di determinare con esattezza quando l'istanza avesse realmente terminato di

elaborare i dati visivi. Di conseguenza, si è optato per una radicale transizione architetturale, sostituendo MQTT con Amazon SQS (Simple Queue Service) per la gestione e l'orchestrazione dei carichi di lavoro computazionalmente intensivi.

Il *worker* neurale in esecuzione sull'istanza EC2 adotta la tecnica del Long Polling. A differenza del *polling* tradizionale, la richiesta HTTP di lettura della coda SQS rimane aperta fino a un massimo di 20 secondi. Se nessun messaggio (ovvero la notifica di un nuovo video) giunge in questo intervallo, la connessione si chiude in modo pulito e ne viene immediatamente aperta una successiva. Questo paradigma garantisce due vantaggi architetturali fondamentali per la stabilità del sistema:

- **Tolleranza ai guasti e certezza del task:** Quando un messaggio viene prelevato dal *worker*, SQS non lo elimina immediatamente, ma lo nasconde al resto del sistema attivando un *Visibility Timeout* [42]. Se il container neurale dovesse arrestarsi in modo anomalo (*crash* dell'applicativo o esaurimento della RAM) durante l'analisi del video e prima di poter inviare il comando di cancellazione definitiva, il *timeout* scadrebbe e il *task* tornerebbe visibile in coda. Questo meccanismo costituisce un vantaggio cruciale, poiché assicura la *fault tolerance* dell'architettura garantendo che nessuna ispezione vada mai persa a fronte di errori sistemici.

- **Azzeramento del traffico passivo per il de-provisioning automatico:** L'assenza dei pacchetti *Keep-Alive* di MQTT si traduce in un silenzio di rete assoluto. Durante i 20 secondi di attesa del *Long Polling*, la metrica `NetworkIn` dell'interfaccia virtuale scende a zero. Tale condizione fornisce un segnale inequivocabile dello stato di inattività del server. Non essendovi rumore di fondo, un valore pari a zero certifica in modo deterministico che la macchina non sta ricevendo nuovi dati. Questo segnale netto (traffico elevato contro zero assoluto) costituisce il presupposto logico essenziale per configurare in totale sicurezza gli allarmi CloudWatch, permettendo di innescare lo spegnimento fisico della GPU senza il rischio di falsi positivi che interromperebbero un'elaborazione in corso.

4.5 Logica del Worker EC2: SQS e Long Polling

Il cuore dell'elaborazione asincrona risiede nello script Python eseguito in *background* all'interno del container Docker su Amazon EC2. Questo *worker* ha il compito di interrogare continuamente la coda SQS per prelevare i messaggi di notifica relativi ai *task* di inferenza visiva.

Nello specifico, poiché le code di messaggistica Cloud sono progettate per latenze bassissime e impongono un limite tassativo di 256 KB per singolo messaggio [43], il *task* prelevato non contiene il file multimediale stesso, bensì un

payload testuale leggero formattato in JSON. Tale documento funge da puntatore (*Claim-Check pattern*) che contiene i metadati essenziali, quali il nome del *bucket* Amazon S3 e la chiave identificativa univoca (*Object Key*) del video appena caricato dal drone. Acquisite queste coordinate da SQS, il *worker* provvede a scaricare temporaneamente il video sulla memoria dell'istanza EC2 per poi sottoporlo alla *pipeline* neurale.

L'implementazione del *Long Polling*, descritta nello Script 4.2, si basa sull'impostazione del parametro `WaitTimeSeconds=20`. Questa istruzione blocca l'esecuzione del ciclo `while` lato server AWS per 20 secondi prima di restituire una risposta vuota, azzerando di fatto il consumo di banda di rete dell'istanza EC2 durante l'attesa e consentendo ai meccanismi di automazione di rilevare lo stato di inattività.

```
import boto3
import time

sqs = boto3.client('sqs', region_name='eu-south-1')
QUEUE_URL = "https://XXXXXX/DroneTaskQueue"

def run_worker():
    print("Avvio Worker SQS. In attesa di task...")
    while True:
        # Chiamata bloccante (Long Polling)
        response = sqs.receive_message(
            QueueUrl=QUEUE_URL,
            MaxNumberOfMessages=1,
            WaitTimeSeconds=20
        )

        if 'Messages' in response:
            msg = response['Messages'][0]
            receipt_handle = msg['ReceiptHandle']
            print(f"Task ricevuto. Avvio inferenza su metadati: {msg['Body']}")

            # Esecuzione del modello visivo (omessa per brevità)
            # video_path = download_from_s3(msg['Body'])
            # process_video_with_yolo(video_path)

            # Rimozione esplicita del messaggio per evitare rielaborazioni
            sqs.delete_message(
                QueueUrl=QUEUE_URL,
                ReceiptHandle=receipt_handle
            )
        else:
            # Nessun task ricevuto dopo 20 secondi. Ripresa del ciclo.
            pass

if __name__ == "__main__":
    run_worker()
```

Snippet 4.2: Worker Python per l'estrazione asincrona dei task tramite Long Polling.

L'esecuzione asincrona di questo frammento di codice si inserisce in una più ampia architettura a cascata. Per rispondere in modo esaustivo alle dinamiche di elaborazione dei dati, la Figura 4.2 espone lo schema a blocchi di tipo semantico che traccia l'esatto ciclo di vita dell'informazione, dall'avvio del nodo computazionale fino alla memorizzazione del risultato finale.

Il diagramma scompone l'interazione in quattro stadi operativi sequenziali. La prima fase gestisce l'innesco e l'accodamento, dove il caricamento nel *bucket* dei dati grezzi su Amazon S3 genera l'evento `s3:ObjectCreated` che popola la coda SQS per la generazione del *task*. La seconda fase descrive l'inizializzazione dell'infrastruttura, partendo dal *trigger* di avvio dell'istanza EC2 per procedere con l'avvio del nodo computazionale, il download del container Docker da ECR e l'attivazione del *worker* Python. La terza fase rappresenta il nucleo della *pipeline* di elaborazione semantica. Il *worker* riceve il messaggio tramite *Long Polling* di 20 secondi ed estrae i metadati del *payload* per avviare il download del video da S3. Inizia quindi l'estrazione dei *frame* e l'inferenza spaziale tramite YOLO11n. I ritagli d'immagine vengono successivamente inoltrati a Qwen2-VL per l'analisi contestuale profonda. Infine, la quarta fase chiude il ciclo curando la conclusione e il salvataggio. Il sistema genera il referto conclusivo, lo deposita nel *bucket* S3 di destinazione in formato JSON e rimuove definitivamente il *task* dalla coda SQS per impedire rielaborazioni accidentali.

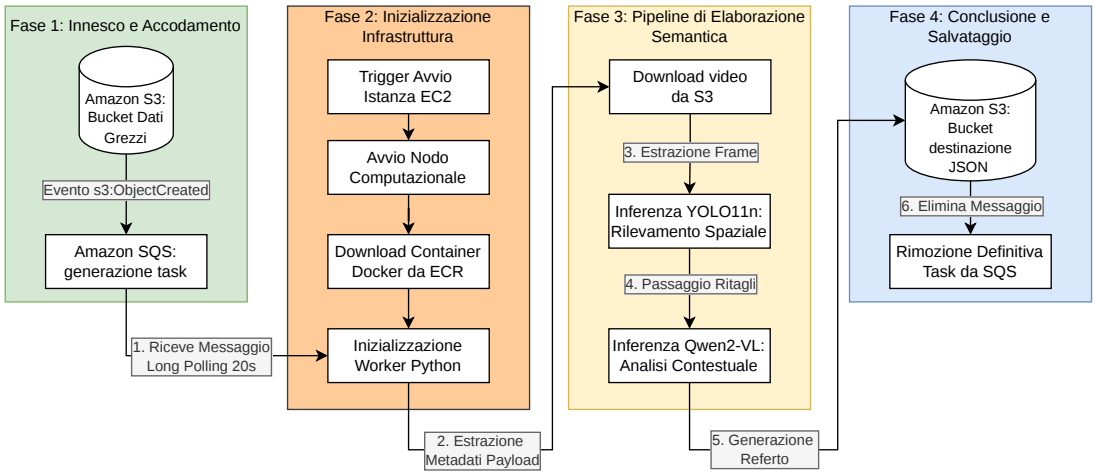


Figura 4.2: Schema a blocchi degli stadi operativi della pipeline semantica. La figura dettaglia i passaggi interni di elaborazione dati, evidenziando le quattro macro-fasi logiche di innesco, inizializzazione, inferenza a cascata e salvataggio finale dei risultati.

Per formalizzare il ciclo di vita del processo in *background*, il diagramma a stati UML illustrato in Figura 4.3 modella analiticamente le transizioni operative del container Docker [44].

Entrando nel dettaglio, lo schema definisce i seguenti stati e le relative azioni:

- **Inizializzazione e Attesa:** Il sistema entra nello stato attivo tramite l'evento di Avvio (Pre-Warm), transitando nel blocco macroscopico di SQS_Long_Polling. Qui, il sistema mantiene la Connessione Aperta (20s) rimanendo nello stato "In_Attesa" di un Messaggio SQS.

- **Esecuzione della Pipeline:** Al verificarsi dell'evento Task Ricevuto, il sistema interrompe la fase di polling e avvia sequenzialmente il Download_Video dal *Data Lake*, per poi passare all'Estrazione Frame.

- **Inferenza a Cascata:** Le immagini estratte vengono sottoposte allo stato di YOLO11n_Inference. Il diagramma prevede a questo punto una biforcazione condizionale fondamentale. Se non vi è nulla di rilevante nella scena (Nessun Target Frame Vuoto), si passa direttamente all'archiviazione. Qualora invece venga identificata un'anomalia (Target Rilevato), si attiva lo stato di "Qwen2_VL_Inference" per generare il referto semantico approfondito.

- **Completamento e Riavvio:** Dopo il Salvataggio_S3, l'avvenuta elaborazione innesca l'evento di Task Completato (Cancellazione Messaggio) per evitare rielaborazioni duplicate, riportando in *loop* il sistema nello stato di ascolto SQS iniziale.

Come si evince dallo schema logico, la terminazione definitiva del processo non è programmata tramite un'istruzione software interna al codice Python (come un comando exit()), bensì è delegata a un fattore infrastrutturale esterno. È infatti la configurazione di Amazon CloudWatch ad assumere il controllo dell'arresto. Qualora i sensori registrino un traffico di rete in ingresso inferiore a 100 kB per un periodo continuativo di 15 minuti (Allarme CloudWatch), scatta un *trigger* architetturale che forza lo Spegnimento_EC2 [45]. Questa interruzione hardware (Stop Fisico Istanza) garantisce la disattivazione sicura del *worker*, concretizzando i principi di ottimizzazione dei costi precedentemente discussi.

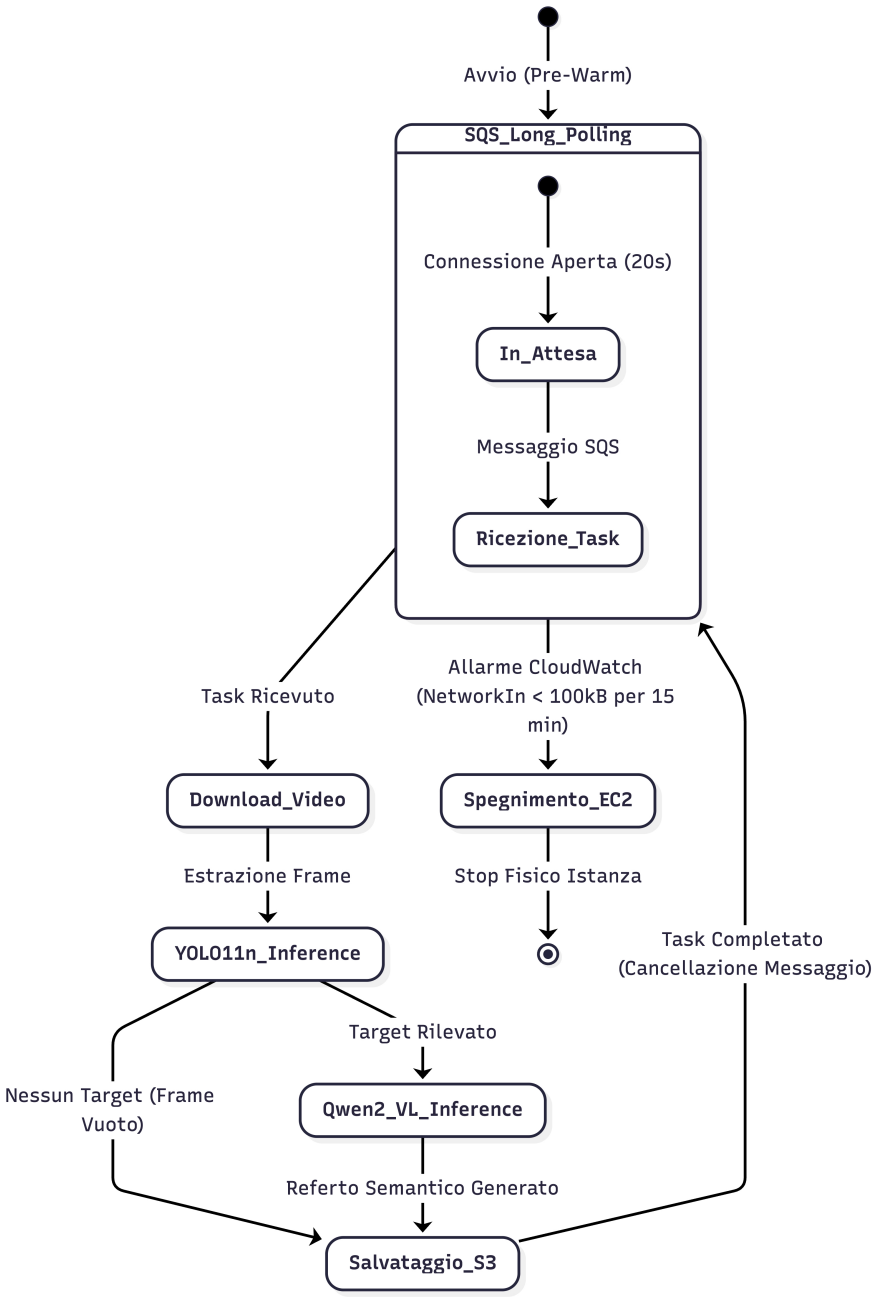


Figura 4.3: Diagramma a stati UML del *worker* EC2. Lo schema evidenzia la biforcazione logica dell'inferenza visiva (YOLO e Qwen2-VL) e la terminazione esterna del ciclo di vita delegata all'allarme di inattività CloudWatch.

4.6 Automazione del De-Provisioning tramite CloudWatch

L'efficienza economica di un'architettura Cloud-Edge dipende strettamente dalla capacità di deallocare le risorse computazionali onerose al termine del loro impiego. Mantenere attiva un'istanza GPU in assenza di flussi video da analizzare configura infatti il fenomeno noto nella letteratura ingegneristica e gestionale come *Cloud Waste* (spreco di risorse cloud), una criticità architetturale che incide severamente sui costi operativi [46].

Sfruttando l'azzeramento del traffico passivo garantito dal paradigma *Long Polling* su SQS, si è progettata una logica di *de-provisioning* deterministica e interamente automatizzata, demandandone la supervisione ad Amazon CloudWatch. Nello specifico, i sensori di telemetria infrastrutturale sono stati configurati per campionare costantemente la metrica `NetworkIn`, ovvero il volume di traffico di rete in ingresso all'interfaccia virtuale dell'istanza EC2. La configurazione dei parametri logici di monitoraggio è strutturata come segue:

- **Soglia di tolleranza:** Fissata a 100 kByte, al fine di filtrare e ignorare il fisiologico rumore di fondo della rete, come i micro-flussi generati dai pacchetti TCP di controllo, dagli aggiornamenti del demone SSH o dai log di sistema.

- **Periodo di valutazione (*Datapoint*):** Il traffico aggregato viene campionato e valutato a intervalli regolari di 5 minuti.

- **Regola di innesco (Condizione di allarme):** Nel contesto dei sistemi di monitoraggio Cloud, la condizione di allarme rappresenta l'espressione logico-matematica che determina il passaggio di stato del sistema da `OK` ad `ALARM` [47]. In questa specifica architettura, la condizione stabilisce che l'allarme si inneschi esclusivamente se il traffico rilevato permane ininterrottamente al di sotto della soglia di tolleranza per 3 periodi di valutazione consecutivi. Questa finestra temporale di 15 minuti complessivi funge da meccanismo di *debouncing* (anti-rimbalzo), evitando lo spegnimento accidentale della macchina dovuto a temporanee cadute di connessione radio dell'UAV.

Al raggiungimento di tale condizione, CloudWatch invia un segnale di sistema invocando l'azione nativa arn:aws:automate:region:ec2:stop. Questo comando forza l'arresto fisico della macchina virtuale (transizione allo stato `stopped`), concretizzando il *de-provisioning* automatico e l'interruzione della fatturazione senza richiedere alcun intervento manuale da parte dell'operatore.

4.7 Ingegneria Finanziaria (FinOps) e Abbattimento del Cloud Waste

L'architettura *event-driven* progettata non risponde solo a requisiti puramente prestazionali, ma affronta una delle sfide più critiche nell'adozione dei servizi Cloud, ovvero l'ottimizzazione e il contenimento dei costi. Per governare tale aspetto, la piattaforma integra nativamente i principi del *Cloud FinOps*, un vero e proprio *framework* metodologico che unisce ingegneria, finanza e automazione per massimizzare il ritorno sull'investimento dell'infrastruttura [37].

La letteratura scientifica e i report di settore [46, 48] evidenziano come una percentuale significativa della spesa Cloud globale si traduca in *Cloud Waste* (superfluo), causato primariamente dal mantenimento in esecuzione di istanze orfane o pesantemente sottoutilizzate. Le macchine accelerate da hardware, come le istanze EC2 della famiglia `g4dn`, presentano infatti costi orari elevati che gravano sul bilancio anche quando si trovano in semplice stato di attesa.

L'implementazione del blocco combinato SQS (*Long Polling*) e degli allarmi CloudWatch trasforma un'infrastruttura tradizionalmente statica in un ecosistema *Zero-Idle* (o *Scale-to-Zero*). In ingegneria del software, tale paradigma definisce un sistema capace di azzerare i tempi di inattività. Non appena la coda dei *task* si svuota, l'architettura scala autonomamente rimuovendo le risorse computazionali [49]. L'arresto fisico della macchina, innescato al calo della metrica `NetworkIn`, elimina alla radice l'errore umano legato alla mancata disattivazione manuale da parte dell'operatore. Ciò garantisce che l'addebito orario avvenga esclusivamente in corrispondenza dell'effettiva esecuzione di carichi di lavoro utili all'ispezione visiva, rigettando in via definitiva l'inefficiente modello dei server *always-on*.

4.8 Flusso Architetturale della Pipeline di Elaborazione

Al fine di sintetizzare le interazioni e le dipendenze logiche tra i vari microservizi del sistema (FastAPI, Amazon S3, SQS, AWS Lambda e il nodo EC2), è stato tracciato lo schema conclusivo riportato in Figura 4.4.

Il diagramma illustra i due filoni paralleli della *pipeline*, ovvero il rilevamento della telemetria tramite AWS IoT Core e il caricamento dei file multimediali su Amazon S3. Entrambi gli eventi agiscono da innesco (*trigger*) per i livelli successivi, provocando rispettivamente la richiesta di avvio dell'istanza EC2 (tramite AWS Lambda) e la creazione del *task* in coda (tramite Amazon SQS).

Questa ramificazione logica evidenzia il completo disaccoppiamento spaziale e temporale tra il livello di acquisizione dei dati e il livello di calcolo GPU. Come formalizzato nella letteratura dell'Ingegneria del Software, tale disaccoppiamento

costituisce la validazione teorica della natura puramente *event-driven* dell'architettura [50]. In questo paradigma architetturale, infatti, i componenti produttori (come S3 o FastAPI) si limitano ad emettere un evento verso un intermediario, ignorando del tutto lo stato, l'identità o i tempi di esecuzione del servizio consumatore finale (il *worker* sull'istanza EC2).

Questa rigorosa separazione garantisce che il transito dei dati non generi mai chiamate sincrone bloccanti. Il drone e il cruscotto web non devono mai attendere in linea che la rete neurale finisca il proprio lavoro. Al contrario, essi scaricano asincronamente il proprio *payload* e si liberano immediatamente, assicurando la totale reattività del sistema e l'isolamento dei guasti (*fault isolation*) tra i moduli [50].

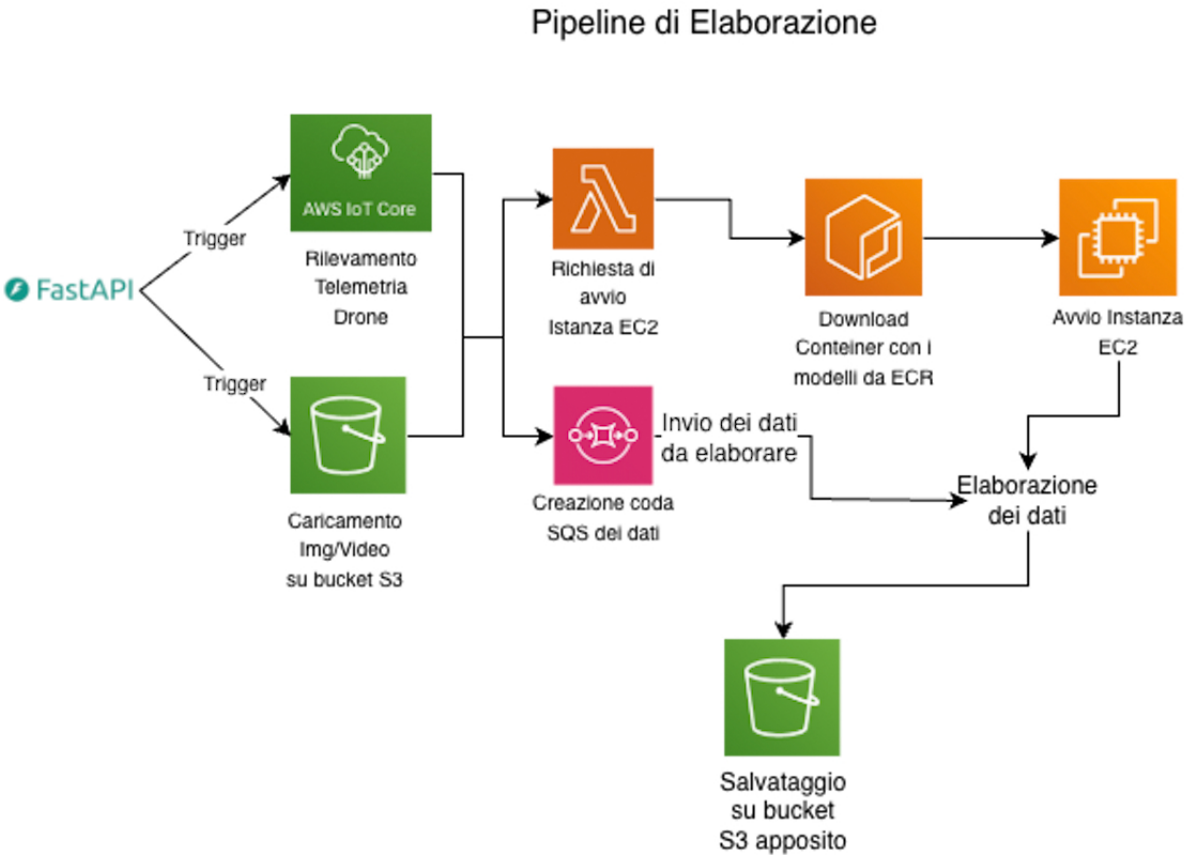


Figura 4.4: Schema logico della *Pipeline* di Elaborazione. Il diagramma evidenzia come gli inneschi (*Trigger*) di FastAPI scatenino l'elaborazione parallela: da un lato l'accodamento logico su SQS, dall'altro la preparazione infrastrutturale di EC2 tramite Lambda e il download da ECR, isolando totalmente l'acquisizione dal calcolo pesante.

Capitolo 5

Valutazione delle Prestazioni e Validazione Architetturale

Il presente capitolo espone l'analisi quantitativa e qualitativa della *pipeline* Cloud-Edge, validando le scelte ingegneristiche effettuate durante la fase di progettazione. I test sperimentali sono stati condotti impiegando un'istanza Amazon EC2 `g4dn.xlarge` equipaggiata con acceleratore grafico NVIDIA Tesla T4 (16 GB di VRAM), focalizzando l'attenzione sui tempi di inferenza, sull'ottimizzazione delle risorse hardware e sul mantenimento dei rigidi vincoli di latenza imposti dalle operazioni UAV in tempo reale.

5.1 Prestazioni del Rilevamento Visivo (YOLO11n)

Il primo stadio della *pipeline*, deputato all'identificazione dei *target* nel flusso video grezzo, impiega l'architettura neurale YOLO11n. Al fine di valutare le prestazioni architetturali pure della piattaforma Cloud-Edge, il modello è stato implementato nella sua versione *off-the-shelf*, caricando direttamente i pesi pre-addestrati sul *dataset* COCO (*Common Objects in Context*) [51]. Non è stato pertanto eseguito alcun addestramento aggiuntivo (*fine-tuning*) su immagini aeree, affidandosi alla robustezza intrinseca della rete per il riconoscimento delle classi di base.

Storicamente, l'invio di flussi video continui da droni verso stazioni di terra ha sempre rappresentato un severo collo di bottiglia critico, soggetto a perdite di pacchetti e saturazione della banda in contesti di emergenza. L'impiego di YOLO11n funge da filtro semantico preventivo. Analizzando la scena visiva a una velocità di inferenza teorica superiore agli 80 *frame* al secondo (FPS) [52], il modello scarta istantaneamente i fotogrammi privi di anomalie, instradando esclusivamente le immagini contenenti *target* confermati.

Per poter essere processati in parallelo dai *Tensor Cores* della GPU, i fotogrammi estratti dal flusso video vengono preventivamente ridimensionati alla

risoluzione di 640×384 pixel. Questa riproiezione spaziale risponde a uno stringente vincolo matematico delle reti convoluzionali della famiglia YOLO, le quali richiedono che l'altezza e la larghezza del tensore in ingresso siano multipli esatti dello *stride* massimo della rete (tipicamente pari a 32) [53]. Fissando la larghezza standard a 640 pixel, l'altezza proporzionale originale di 360 pixel (tipica del rapporto 16:9) viene compensata con 24 pixel aggiuntivi di *padding* neutro (*letterboxing*) per raggiungere i 384 pixel. Questo accorgimento rispetta l'allineamento della memoria (384 è perfettamente divisibile per 32) ed evita qualsiasi distorsione geometrica degli oggetti inquadrati, che degraderebbe l'accuratezza del rilevamento.

L'efficienza computazionale del modello è direttamente quantificabile confrontando i tempi di esecuzione con la soglia critica del *real-time*. Come si evince dai log di sistema (Tabella 5.1), la latenza complessiva si attesta a ∼11.1 millisecondi per fotogramma. Considerando che un flusso video standard a 30 FPS impone di concludere ogni elaborazione entro un limite massimo di 33.3 ms, il sistema opera a un terzo del tempo limite consentito.

Fase di Elaborazione (YOLO11n)	**Tempo Medio (ms)**
Pre-processing (Ridimensionamento e normalizzazione)	1.6
Inference (Calcolo su tensori GPU)	8.3
Post-processing (NMS e Bounding Box rendering)	1.2
Latenza Totale per Fotogramma	**∼11.1**

Tabella 5.1: Tempi di inferenza medi registrati sull'acceleratore NVIDIA T4. Il totale di 11.1 ms certifica un ampio margine rispetto al limite teorico di 33.3 ms per flussi a 30 FPS.

Questo margine di tolleranza, superiore a 20 ms, risulta vitale. Esso funge da *buffer* per assorbire eventuali fluttuazioni di latenza del sistema operativo (*OS jitter*) e assicura la fluidità ininterrotta dell'elaborazione anche nel caso in cui le frequenze di *clock* dell'hardware subiscano un declassamento automatico dovuto all'eccessivo calore (*thermal throttling*) durante missioni di volo prolungate.

5.2 Profilazione Hardware e Consumi (Cloud GPU)

L'orchestrazione parallela di un modello di *Object Detection* e di un *Vision-Language Model* (VLM) impone vincoli hardware stringenti. Architetture monolitiche del passato tentavano di eseguire reti neurali complesse direttamente su elaboratori CPU. Ad esempio, l'elaborazione di reti convoluzionali profonde (come le prime VGG16) per il riconoscimento visivo richiedeva svariati secondi o addirittura minuti per l'inferenza di un singolo fotogramma ad alta risoluzione in

assenza di accelerazione parallela, rendendo impossibile l'analisi in tempo reale [54].

Sfruttando invece l'architettura Turing della GPU NVIDIA Tesla T4, il sistema garantisce latenze minime. Rispetto alle generazioni precedenti (come l'architettura Pascal), l'architettura Turing si differenzia per l'introduzione dei *Tensor Core* di seconda generazione: unità di calcolo hardware specializzate che permettono l'esecuzione di operazioni matriciali a precisione mista (FP16 e INT8), massimizzando il *throughput* specifico per l'inferenza delle reti neurali senza compromettere l'accuratezza [55].

Metrica Operativa (Tesla T4)	**Valore Rilevato**	**% su VRAM (15360 MiB)**
YOLO11n (Filtro spaziale)	242 MiB	1.5%
Qwen2-VL-2B (VLM semantico)	9432 MiB	61.4%
Memoria Totale Allocata (Picco)	**9677 MiB**	**63.0%**
Utilizzo Compute GPU	**100%**	-
Assorbimento Energetico	**71 W**	**Max (70W)**

Tabella 5.2: Profilazione reale del consumo di memoria e potenza durante l'esecuzione combinata dei modelli. Il valore di 15360 MiB indica la porzione di memoria video (VRAM) effettivamente indirizzabile e rilevata dal sistema operativo, a fronte dei 16 GB fisici nominali della scheda.

Come mostrato in Tabella 5.2, il sistema è stato profilato tramite l'utility di diagnostica `nvidia-smi` durante la fase di picco (*Inference Peak*). L'enorme divario di occupazione della memoria non è indice di inefficienza di un modello rispetto all'altro, ma riflette la radicale differenza di scopo e progettazione architetturale dei due software. YOLO11n è una rete convoluzionale specializzata e ridotta all'essenziale per l'estrazione di *bounding box*, pensata per operare su dispositivi *Edge*, richiedendo un'allocazione di memoria di soli 242 MiB. Al contrario, Qwen2-VL-2B è un modello fondazionale ad alta capacità parametrica basato su architettura Transformer (dotato di oltre 2 miliardi di parametri), il quale necessita di allocare stabilmente 9.4 GB di VRAM unicamente per pre-caricare i propri pesi neurali prima dell'avvio dei calcoli.

L'utilizzo del 100% delle risorse di calcolo e l'assorbimento di 71 W (al limite del *Total Graphics Power* della scheda) confermano che la macchina `g4dn.xlarge` rappresenta il taglio infrastrutturale minimo indispensabile per sorreggere questo carico di lavoro. Lasciare un *buffer* di circa 5.6 GB di VRAM libera è un requisito architetturale imperativo. Come intrinseco in tutti i modelli linguistico-visivi basati su Transformer, il meccanismo di calcolo dell'attenzione, che in Qwen utilizza l'algoritmo *M-RoPE* (*Multimodal Rotary Positional Embedding*), presenta una complessità spaziale che scala in modo quadratico, definita come $O(N^2)$, rispetto al numero di *token* in ingresso [26].

All'aumentare della risoluzione nativa dell'immagine, esplode geometricamente il numero di *patch* visive da processare. Questa espansione costringe la GPU ad allocare gigabyte di memoria temporanea per archiviare la cache KV (*Key-Value*) e le matrici di attenzione durante il *forward pass*. L'assenza del margine di 5.6 GB condurrebbe a fatali errori di *Out-Of-Memory* (OOM) ed al conseguente crash dei microservizi containerizzati. Per mitigare tale rischio, la *pipeline* implementa un ridimensionamento dinamico del *frame* visivo prima di sottoporlo al VLM per l'inferenza multimodale (ovvero la generazione del referto descrittivo testuale).

5.3 Validazione Empirica su Flusso Video

Per valutare la robustezza della *pipeline* Edge-to-Cloud in uno scenario operativo reale, il sistema è stato testato elaborando un flusso video continuo della durata di 63 secondi, relativo a un ambiente costiero e industriale. L'infrastruttura ha completato l'intero ciclo di *ingestion*, filtraggio spaziale tramite YOLO11n e inferenza semantica complessa tramite Qwen2-VL in un tempo assoluto di 110 secondi.

Dal punto di vista metrico, dividendo il tempo di elaborazione totale per la durata nativa del frammento multimediale, si ottiene un *Real-Time Factor* (RTF) pari a $\sim$1.75 [56]. In ingegneria dei sistemi multimediali, questo rapporto esplicita matematicamente che per ogni secondo effettivo di video catturato dall'UAV, l'infrastruttura Cloud necessita di 1.75 secondi per portare a termine l'intera complessa filiera di analisi neurale.

Poiché il valore è superiore a 1 (il che indica un'elaborazione più lenta dello scorrere del tempo fisico), il transito diretto e sincrono dei fotogrammi causerebbe inevitabilmente il blocco del server per esaurimento delle risorse (saturazione delle code di *I/O*). Tuttavia, questo risultato empirico valida matematicamente la scelta architetturale di interporre Amazon SQS; operando come un *buffer* elastico [57], la coda assorbe e gestisce asincronamente i picchi di latenza generati dai modelli fondazionali visivo-linguistici. Il disaccoppiamento previene così il sovraccarico e il conseguente blocco (*throttling*) del *worker* EC2, confermando che l'infrastruttura è perfettamente idonea e stabile per missioni di ispezione asincrone.

Il filtro *Edge* (YOLO11n) ha dimostrato un'eccellente efficienza selettiva, innescando il massiccio trasferimento multimodale esclusivamente in corrispondenza del rilevamento di classi predefinite. Di seguito vengono analizzati nel dettaglio i risultati semantici generati, partendo dal comportamento nominale del sistema fino all'esame metodologico dei casi limite e delle relative allucinazioni del modello linguistico.

Risultati della Pipeline: Comprensione del Contesto



Figura 5.1: Estrazione di un frame: ispezione nominale dello stabilimento balneare.

Output del Cruscotto (Edge-to-Cloud):

- **Oggetti (YOLO):** `PERSON`
- **Semantica (Qwen):**

> *"La foto mostra una spiaggia con sedie da sole in ordine rettilinea lungo il mare. Sul lato sinistro della spiaggia c'è un sentiero per le persone che possono camminare o fare attività sportive. Al centro della spiaggia ci sono alcuni bagnanti e alcune sedie vuote... Il mare è tranquillo e riflette la luce del sole su uno specchio d'acqua brillante. Non vi sono evidenti anomalie nella fotografia."*

Il report generato in corrispondenza della Figura 5.1 illustra la capacità del sistema di operare in condizioni standard. Il *trigger* visivo (`PERSON`) elaborato dal nodo Edge attiva l'inferenza del modello Qwen2-VL, il quale mappa spazialmente la topologia dell'infrastruttura inquadrata. Come formalizzato nella letteratura inerente la valutazione dei modelli fondazionali [58], l'esito di una singola inferenza visiva non possiede la valenza statistica necessaria per costituire una certificazione formale di affidabilità universale. Tuttavia, la corretta deduzione finale sull'assenza di anomalie operative o contestuali, quali la presenza di ostacoli lungo i percorsi, danni alle strutture balneari o potenziali situazioni di pericolo per i bagnanti, rappresenta un solido riscontro empirico a favore del corretto funzionamento dell'architettura, fornendo un'ulteriore validazione qualitativa sull'efficacia del modello come strumento di monitoraggio passivo.

Risultati della Pipeline: Ambiguità Geometrica e Sovrastima



Figura 5.2: Estrazione di un frame: area ferroviaria e lettini accatastati.

Output del Cruscotto (Edge-to-Cloud):

- **Oggetti (YOLO):** `CAR`
- **Semantica (Qwen):**

> *"La foto mostra un'area urbana con una strada asfaltata, ferrovie in posizione verticale e alcuni veicoli parcheggiati... Anomalie notevoli includono: 1. Il numero di veicoli apparentemente troppo elevato rispetto alla quantità reale di auto."*

Nel caso illustrato in Figura 5.2, Qwen2-VL contestualizza adeguatamente l'ambiente ferroviario, ma deduce anomalie inesistenti sovrastimando il numero di veicoli. Questo errore è imputabile alla complessa geometria dei lettini balneari impilati in basso a sinistra, che il modello multimodale confonde con le geometrie ripetute di vetture parcheggiate.

Risultati della Pipeline: Casi Limite e Allucinazioni a Cascata



Figura 5.3: Estrazione di un frame: sistemazione dell'area ristoro esterna.

Output del Cruscotto (Edge-to-Cloud):

- **Oggetti (YOLO):** `MOTORCYCLE, CHAIR, PERSON`
- **Semantica (Qwen):**

> *"La scena mostra un ambiente esterno... Sullo sfondo c'è uno spazzacamino rosso che sembra essere utilizzato per pulire le sedie. Anomalia notevole è l'esistenza del "motorcycle" (motocicletta), ma questo oggetto non risulta presente nell'immagine reale."*

Un fenomeno di allucinazione logica emerge analizzando la Figura 5.3. Il rilevatore Edge classifica erroneamente l'idropulitrice rossa come `MOTORCYCLE`. Ricevendo questo *prompt* visivo distorto, Qwen2-VL tenta di razionalizzare l'oggetto descrivendolo come uno "spazzacamino rosso". Successivamente rileva la discrepanza testuale segnalando l'assenza della motocicletta come anomalia. Questa dinamica conferma che la precisione semantica nel Cloud è strettamente vincolata all'accuratezza della pre-elaborazione all'Edge.

5.4 Validazione Scientifica e Confronto con lo Stato dell'Arte

Le scelte implementative sono state guidate da un'attenta comparazione con i modelli attuali, bilanciando il *trade-off* tra capacità di generalizzazione e costo computazionale [59].

5.4.1 YOLO11n vs YOLOv8n

Sebbene YOLOv8 rappresenti uno standard industriale ampiamente documentato in letteratura [60], la release YOLO11 introduce un design ottimizzato dei blocchi C3k2 e *Spatial Pyramid Pooling Fast* (SPPF). A parità di parametri totali rispetto al predecessore, YOLO11n riduce del 22% il numero di operazioni in virgola mobile (FLOPs) pur registrando un lieve incremento della *Mean Average Precision* (mAP) sui dataset pubblici come COCO. Nel dominio aerospaziale, questa evoluzione algoritmica è determinante: minori calcoli tensoriali si traducono direttamente in un abbattimento del consumo energetico sulle batterie dei sistemi Edge.

5.4.2 Qwen2-VL (2B) vs Architetture LLaVA (7B)

Nel panorama emergente dei *Vision-Language Models*, architetture pioniere come LLaVA [61], basate su *backbone* linguistici complessi come Vicuna o LLaMA, richiedono file di pesi superiori ai 7 miliardi di parametri. Il caricamento di tali modelli satura quasi completamente i 16 GB di una T4, costringendo l'impiego di hardware di fascia superiore (come le istanze basate su NVIDIA A10G) e raddoppiando di conseguenza i costi operativi.

Qwen2-VL-2B-Instruct riduce il carico computazionale integrando il modulo *ViT M-RoPE* (Multimodal Rotary Positional Embedding) [62]. A differenza dei sistemi tradizionali che comprimono l'immagine a risoluzioni quadrate fisse causando la perdita di dettagli visivi fondamentali, Qwen analizza il *frame* nativo in proporzioni arbitrarie convertendolo in *token* visivi dinamici. Questo approccio preserva fedelmente la geometria spaziale dell'inquadratura e le reali distanze relative tra gli elementi. Di conseguenza, il modello non si limita a riconoscere la classe degli oggetti presenti, ma acquisisce una profonda consapevolezza topologica della scena. Tale caratteristica risulta essenziale per l'individuazione di anomalie legate alla distribuzione spaziale o alla densità degli elementi monitorati. Valutando il posizionamento relativo e la proporzione degli ingombri, il VLM è infatti in grado di dedurre incongruenze di natura logica o quantitativa, garantendo una capacità di analisi semantica paragonabile a quella di reti tre volte più grandi.

5.5 Analisi Economica (FinOps): Cloud Tradizionale vs Event-Driven

Oltre alle performance computazionali, l'architettura è stata validata dal punto di vista finanziario. L'istanza `g4dn.xlarge` impiegata (basata su architettura x86 e acceleratore NVIDIA T4) presenta un costo *On-Demand* nella regione `eu-south-1` (Milano) pari a circa $0.65 all'ora.

In un'architettura Cloud tradizionale di tipo *always-on*, in cui il server dedicato all'inferenza rimane costantemente allocato in attesa dei flussi video, il costo operativo mensile (calcolato su una media di 730 ore) ammonterebbe a circa $474.50.

Grazie al paradigma *event-driven* orchestrato tramite Amazon SQS e CloudWatch, il sistema massimizza l'efficienza del modello tariffario *pay-as-you-go* (pagamento a consumo). L'addebito per le risorse computazionali viene innescato esclusivamente durante l'effettivo svolgimento della missione visiva e il successivo smaltimento dei *task* in coda [63]. Ipotizzando uno scenario operativo realistico in cui una flotta UAV esegua ispezioni per un totale di 2 ore giornaliere, il tempo di allocazione dell'hardware accelerato si riduce drasticamente a sole 60 ore mensili.

Come evidenziato nel grafico comparativo in Figura 5.4, questa ottimizzazione dei tempi di esecuzione fa crollare la spesa infrastrutturale a soli $39 mensili. A livello analitico, ciò si traduce in un abbattimento del *Cloud Waste* pari a circa il 91.8%, validando quantitativamente la sostenibilità economica (FinOps) dell'intera soluzione ingegnerizzata.

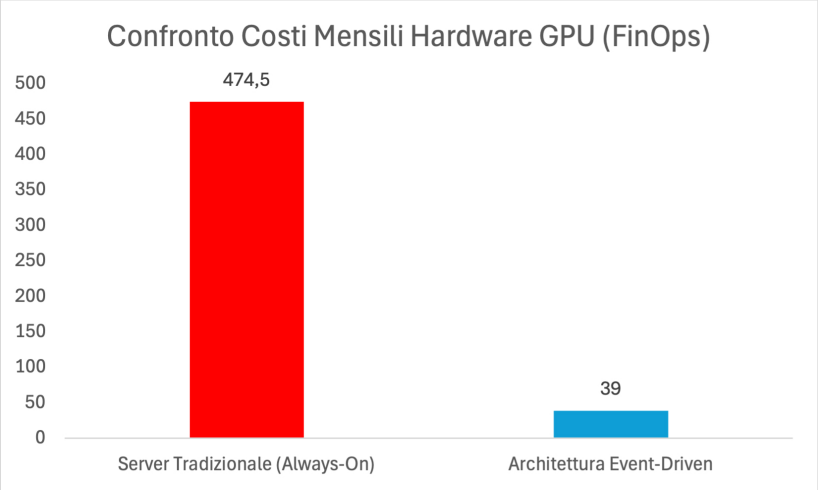


Figura 5.4: Confronto dei costi mensili hardware GPU (FinOps): l'architettura *always-on* a confronto con la *pipeline event-driven*.

5.6 Analisi della Latenza End-to-End

L'efficacia del paradigma *event-driven* si misura nel tempo totale di attraversamento del dato. Sistemi distribuiti basati su chiamate sincrone (REST) o meccanismi di *polling* diretti tra microservizi tendono a fallire catastroficamente in caso di micro-interruzioni della rete cellulare del drone [64].

Il disaccoppiamento garantito da Amazon SQS trasforma la vulnerabilità della rete in una tolleranza nativa ai guasti. Dal momento esatto in cui il video completa l'upload asincrono su Amazon S3 tramite *Pre-signed URL*, il *worker* EC2 viene innescato dalla coda. L'intera esecuzione della pipeline comprendente il download interno al Cloud, l'estrazione dei frame, il filtraggio YOLO, la generazione semantica Qwen e il salvataggio del referto JSON si conclude in soli 10 secondi circa. Questo tempo di reazione, unito all'affidabilità semantica dimostrata, convalida l'utilizzo dell'architettura in scenari operativi reali di sorveglianza e risposta rapida.

Capitolo 6

Conclusioni e Sviluppi Futuri

L'obiettivo principale di questo lavoro di tesi è stato superare le limitazioni intrinseche dei sistemi UAV tradizionali, progettando un'architettura ibrida Cloud-Edge capace di coniugare la flessibilità del volo autonomo con la potenza di calcolo delle infrastrutture distribuite. In fase di progettazione sono stati delineati molteplici obiettivi specifici, i quali sono stati tutti conseguiti e validati con successo durante la fase di collaudo.

Nello specifico, la gestione della telemetria è stata implementata in modo sicuro tramite il protocollo MQTT e AWS IoT Core, mentre l'archiviazione dello stato corrente dell'UAV su Amazon DynamoDB ha garantito latenze di lettura minime per le interfacce operative. Il problema del caricamento asincrono degli *asset* visivi è stato risolto adottando il pattern dei Pre-signed URL su Amazon S3, aggirando efficacemente i colli di bottiglia dei backend tradizionali. Per quanto concerne l'orchestrazione, l'impiego di Amazon SQS in modalità *Long Polling* ha azzerato il traffico passivo, ponendo le basi per il contenimento dei costi. L'elaborazione visiva automatizzata ha raggiunto i requisiti di tempo reale (elaborazione entro i limiti per i 30 FPS) tramite l'inferenza a cascata di YOLO11n e Qwen2-VL all'interno di container Docker su istanze EC2 accelerate da GPU. Parallelamente, le logiche FinOps sono state soddisfatte implementando un *de-provisioning* automatico supervisionato da Amazon CloudWatch, che ha drasticamente abbattuto gli sprechi economici. Infine, l'infrastruttura di analisi a posteriori è stata completata con successo orchestrando Kinesis Data Firehose e Amazon Athena per l'interrogazione *serverless* dei log operativi.

Sotto il profilo formativo, lo sviluppo di questo progetto ha rappresentato una fondamentale occasione di crescita accademica e professionale. Il lavoro ha permesso di applicare e consolidare sul campo i principi avanzati di Architetture dei Calcolatori e Cloud Computing. Si è presentata l'opportunità di approfondire l'ecosistema Amazon Web Services, traducendo la teoria architettonica in soluzioni ingegneristiche pratiche. È stata inoltre acquisita un'esperienza diretta e concreta sulla gestione fisica e logica delle macchine virtuali, sull'orchestrazione di container su hardware accelerato da GPU e sull'integrazione di modelli di

intelligenza artificiale multimodali all'avanguardia.

Proprio l'integrazione di tali modelli ha richiesto lo sviluppo di una profonda consapevolezza critica in fase di valutazione. L'impiego operativo di Qwen2-VL ha infatti confermato che il processo di decodifica autoregressiva puramente probabilistica dei *token* può occasionalmente portare il sistema a dedurre la presenza di anomalie inesistenti. Questo fenomeno, noto in letteratura come *hallucination* [65], genera descrizioni strutturali plausibili ma fittizie. Considerare questo limite è essenziale per valutare oggettivamente le reali capacità della piattaforma e per analizzare in modo corretto i risultati ottenuti.

6.1 Sintesi dei Risultati Ottenuti

La prototipazione dell'infrastruttura su Amazon Web Services ha confermato la validità del paradigma *event-driven* per l'Internet of Things ad alte prestazioni. Il disaccoppiamento tra il livello di acquisizione telemetrica (gestito tramite AWS IoT Core e MQTT) e il livello di elaborazione multimediale asincrona (Amazon S3 e SQS) ha permesso di eliminare i colli di bottiglia tipici delle architetture client-server sincrone.

L'integrazione di una *pipeline* di inferenza a cascata ha prodotto risultati eccellenti. L'utilizzo di YOLO11n per il rilevamento spaziale a bassa latenza ha garantito un filtraggio rigoroso del rumore visivo, permettendo al modello multimodale Qwen2-VL-2B di concentrare le proprie risorse semantiche esclusivamente sui fotogrammi critici. Le metriche estratte in fase di collaudo hanno validato l'efficienza della gestione della memoria VRAM sull'acceleratore NVIDIA T4, dimostrando l'elevata robustezza del sistema a scenari di saturazione computazionale.

Inoltre, il progetto ha fornito una risposta concreta alla problematica del *Cloud Waste*. L'ingegnerizzazione della tecnica di *Long Polling* associata alle logiche di *de-provisioning* automatico di Amazon CloudWatch ha trasformato un'infrastruttura Cloud potenzialmente onerosa in un ecosistema a consumo reale, in cui l'hardware GPU viene alimentato e fatturato esclusivamente in presenza di un effettivo fabbisogno logico.

6.2 Limitazioni dell'Architettura Attuale

Sebbene il prototipo sia funzionale, sussistono alcune limitazioni intrinseche che meritano di essere evidenziate, unitamente alle relative strategie di mitigazione.

La prima criticità riguarda la dipendenza dalla connettività mobile. Il caricamento dei video tramite Pre-signed URL dipende strettamente dalla stabilità della connessione 4G/LTE nell'area di volo. In scenari di *disaster response* con infrastrutture di telecomunicazione compromesse, il drone non possiede attual-

mente capacità di *fallback* per la memorizzazione e l'elaborazione semantica a bordo. Una soluzione infrastrutturale per superare questo ostacolo consisterebbe nell'implementazione di protocolli DTN (*Delay-Tolerant Networking*) [66]. Dotando l'UAV di un *buffer* di archiviazione a stato solido (NVMe), il sistema potrebbe salvare i *payload* localmente e avviare il caricamento asincrono solo al ripristino dell'*handshake* con un nodo di rete valido.

La seconda limitazione strutturale è inerente alla gestione delle allucinazioni semantiche. Come documentato nei test, i modelli VLM mantengono un'intrinseca propensione alla generazione di artefatti semantici, ovvero la produzione di descrizioni strutturali plausibili ma fittizie (*hallucination* [65]). Poiché l'output testuale deriva da un processo di decodifica autoregressiva puramente probabilistica dei *token*, il modello può occasionalmente dedurre la presenza di anomalie inesistenti. Nonostante il *tuning* dei parametri per limitare questa deriva generativa, la validazione del report semantico in contesti operativi critici richiede ancora un sistema di verifica con *human-in-the-loop*. Questa dipendenza dall'operatore umano può essere drasticamente ridotta integrando meccanismi di validazione incrociata o ancorando le risposte del modello a un'architettura documentale dedicata.

6.3 Sviluppi Futuri

L'architettura pone solide basi per future evoluzioni, orientate a incrementare il livello di automazione e resilienza della piattaforma.

Sotto il profilo infrastrutturale, l'implementazione delle reti 5G Standalone (SA) offre l'opportunità teorica di sfruttare il paradigma *Multi-Access Edge Computing* (MEC). Ipotizzando la migrazione di parte dei microservizi FastAPI e dei *broker* MQTT direttamente sui server fisici adiacenti alle stazioni radio base (eNodeB/gNodeB), l'architettura potrebbe beneficiare di una considerevole riduzione della latenza trasmissiva, potenzialmente inferiore alla soglia dei 10 ms. Questa prospettiva avvicinerebbe ulteriormente il potere di calcolo all'UAV senza gravare sull'hardware di bordo [67].

Un'ulteriore estensione architetturale risiede nell'implementazione di logiche di *Swarm Intelligence* e orchestrazione multi-drone. Tramite Kinesis Data Streams e algoritmi di instradamento avanzati, un drone primario con funzioni di ricognitore potrebbe delegare i task visivi ad alta risoluzione a droni secondari ispezionatori. Basandosi sui report in tempo reale restituiti dal sistema AWS, sarebbe possibile orchestrare dinamicamente una flotta totalmente autonoma.

Infine, per mitigare i fenomeni di allucinazione precedentemente citati, si prevede l'integrazione di un'architettura RAG (*Retrieval-Augmented Generation*) all'interno del microservizio Qwen2-VL. Implementando un database vettoriale interrogabile sul Cloud, il modello non genererebbe report basandosi unicamente sui propri pesi addestrati, ma confronterebbe l'anomalia rilevata con i manuali

tecnici di ispezione infrastrutturale forniti come contesto in tempo reale. Questo approccio aumenterebbe drasticamente l'affidabilità e l'oggettività del report ingegneristico finale [68].

Bibliografia

[1] Hazim Shakhatreh, Ahmad H Sawalmeh, Ala Al-Fuqaha, Angela Doufexi, Elias Almaita, Issa Khalil, Yousaf Obeidat, Abdallah Khreishah, and Mohsen Guizani. Unmanned Aerial Vehicles (UAVs): A survey on civil applications and key research challenges. *IEEE Access*, 7:48572–48634, 2019. (visitato il giorno 12/10/2026). URL: `https://ieeexplore.ieee.org/document/8682048`.

[2] Weisong Shi, Jie Cao, Quan Zhang, Yihua Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016. (visitato il giorno 12/10/2026). URL: `https://ieeexplore.ieee.org/document/7488250`.

[3] Kimon P. Valavanis and George J. Vachtsevanos. *Handbook of Unmanned Aerial Vehicles*. Springer Dordrecht, 2015. (visitato il giorno 12/10/2026). URL: `https://link.springer.com/referencework/10.1007/978-90-481-9707-1`.

[4] Jiasi Chen and Xukan Ran. Deep Learning with Edge Computing: A Review. *Proceedings of the IEEE*, 107(8):1655–1674, 2019. (visitato il giorno 12/10/2026). URL: `https://ieeexplore.ieee.org/document/8763885`.

[5] Hazim Shakhatreh, Ahmad H Sawalmeh, Ala Al-Fuqaha, Zhengrui Dou, Enas Almaita, Issa Khalil, Mohamed Tabrez Othman, Abdallah Khreishah, and Mohsen Guizani. Unmanned Aerial Vehicles (UAVs): A survey on civil applications and key research challenges. *IEEE Access*, 7:48572–48634, 2019. (visitato il giorno 12/10/2026). URL: `https://ieeexplore.ieee.org/document/8682048`.

[6] Noah Lee. Drone-Based Bridge Health Monitoring and Inspection. *Aaltodoc*, 2025. (visitato il giorno 12/10/2026). URL: `https://aaltodoc.aalto.fi/server/api/core/bitstreams/1b900a88-1d3a-4c8f-a866-12d8a8d2ef4e/content`.

[7] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the*

IEEE conference on computer vision and pattern recognition, pages 779–788, 2016. (visitato il giorno 12/10/2026). URL: https://ieeexplore.ieee.org/document/7780460.

[8] Sattar Dorafshan, Robert J Thomas, and Marc Maguire. Comparison of deep learning convolutional neural networks and edge detectors for image-based crack detection in concrete. *Construction and Building Materials*, 186:1031–1045, 2018. (visitato il giorno 12/10/2026). URL: https://www.sciencedirect.com/science/article/abs/pii/S0950061818319500?via%3Dihub.

[9] B. F. Spencer Jr, V. Hoskere, and Y. Narazaki. Advances in computer vision-based civil infrastructure inspection and monitoring. *Engineering*, 5(2):199–222, 2019. (visitato il giorno 12/10/2026). URL: https://www.sciencedirect.com/science/article/pii/S2095809918308130?via%3Dihub.

[10] Jian Seo, Luis Duque, and James Wacker. Drone-enabled bridge inspection methodology and application. *Automation in Construction*, 94:112–126, 2018. (visitato il giorno 12/10/2026). URL: https://www.sciencedirect.com/science/article/abs/pii/S0926580517309755?via%3Dihub.

[11] Sandeep Sony, Samuel Laventure, and Ayan Sadhu. A literature review of next-generation smart sensing technology in structural health monitoring. *Structural Control and Health Monitoring*, 26(3):e2321, 2019. (visitato il giorno 12/10/2026). URL: https://onlinelibrary.wiley.com/doi/10.1002/stc.2321.

[12] European Union Aviation Safety Agency (EASA). Easy Access Rules for Unmanned Aircraft Systems (Regulation (EU) 2019/947 and Regulation (EU) 2019/945). Technical report, EASA, 2026. (visitato il giorno 12/10/2026). URL: https://www.easa.europa.eu/en/document-library/easy-access-rules/easy-access-rules-unmanned-aircraft-systems.

[13] Jessie Y. C. Chen, Michael J. Barnes, and Michelle Harper-Sciarini. Supervisory control of multiple robots: Human-performance issues and user-interface design. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(4):435–454, 2011. (visitato il giorno 12/10/2026). URL: https://ieeexplore.ieee.org/document/5530413.

[14] Natalia Miloslavskaya and Alexander Tolstoy. Big Data, Fast Data and Data Lake concepts. *Procedia Computer Science*, 88:300–305, 2016.

(visitato il giorno 12/10/2026). URL: https://www.sciencedirect.com/science/article/pii/S1877050916316957?via%3Dihub.

[15] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qiumin Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar Nejad, et al. Cloud programming simplified: A Berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383*, 2019. (visitato il giorno 12/10/2026). URL: https://arxiv.org/abs/1902.03383.

[16] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009. (visitato il giorno 12/10/2026). URL: https://www.sciencedirect.com/science/article/abs/pii/S0167739X08001957?via%3Dihub.

[17] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *The International Journal of High Performance Computing Applications*, 15(3):200–222, 2001. (visitato il giorno 12/10/2026). URL: https://journals.sagepub.com/doi/10.1177/109434200101500302.

[18] Kevin Dorling, Jordan Heinrichs, Geoffrey G Messier, and Sebastian Magierowski. Vehicle routing problems for drone delivery. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(1):70–85, 2017. (visitato il giorno 12/10/2026). URL: https://ieeexplore.ieee.org/document/7513397.

[19] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B Letaief. A survey on mobile edge computing: The communication perspective. *IEEE communications surveys & tutorials*, 19(4):2322–2358, 2017. (visitato il giorno 12/10/2026). URL: https://ieeexplore.ieee.org/document/8016573.

[20] Yazhou Yuan, Shicong Gao, Ziteng Zhang, Wenye Wang, Zhezhuang Xu, and Zhixin Liu. Edge-Cloud Collaborative UAV Object Detection. *IEEE Computer Society*, 2024. (visitato il giorno 12/10/2026). URL: https://www.computer.org/csdl/journal/cc/2024/01/10420469/1Uik7XzxCso.

[21] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015. (visitato il giorno 12/10/2026). URL: https://arxiv.org/abs/1511.08458.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in*

neural information processing systems, volume 25, 2012. (visitato il giorno 12/10/2026). URL: https://papers.nips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

[23] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015. (visitato il giorno 12/10/2026). URL: https://www.nature.com/articles/nature14539.

[24] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015. (visitato il giorno 12/10/2026). URL: https://arxiv.org/abs/1506.01497.

[25] Pengfei Zhu, Longyin Wen, Dawei Du, Xiao Bian, Heng Fan, Qinghua Hu, and Haibin Ling. Vision meets drones: A challenge. *arXiv preprint arXiv:2104.13524*, 2018. (visitato il giorno 12/10/2026). URL: https://arxiv.org/abs/1804.07437.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, volume 30, 2017. (visitato il giorno 12/10/2026). URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[27] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. (visitato il giorno 12/10/2026). URL: https://arxiv.org/abs/2103.00020.

[28] Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. A survey on multimodal large language models. *arXiv preprint arXiv:2306.13549*, 2023. (visitato il giorno 12/10/2026). URL: https://arxiv.org/abs/2306.13549.

[29] Qwen Team. Qwen2-VL: Enhancing Vision-Language Models' Spatial and Temporal Comprehension, 2024. (visitato il giorno 12/10/2026). URL: https://qwen.ai/blog?id=qwen2-vl.

[30] Sebastián Ramírez. FastAPI Features: Performance, 2024. (visitato il giorno 12/10/2026). URL: https://fastapi.tiangolo.com/features/.

[31] R. Nambiar B. Wukkadada, K. Wankhede and A. Nair. Comparison with HTTP and MQTT In Internet of Things (IoT). *IEEE Access*, 2018.

(visitato il giorno 12/10/2026). URL: https://ieeexplore.ieee.org/document/8597401.

[32] Muneer Bani Yassein, Mohammed Shatnawi, Shadi Aljwarneh, and Rawan Al-Hatmi. Internet of Things: Survey and open issues of MQTT protocol. In *2017 International Conference on Engineering & MIS (ICEMIS)*, pages 1–6. IEEE, 2017. (visitato il giorno 12/10/2026). URL: https://ieeexplore.ieee.org/document/8273112.

[33] Carl Boettiger. An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, 2015. (visitato il giorno 12/10/2026). URL: https://dl.acm.org/doi/10.1145/2723872.2723882.

[34] Amazon Web Services. Sharing objects using presigned URLs, 2024. (visitato il giorno 12/10/2026). URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/ShareObjectPreSignedURL.html.

[35] Tim Dierks and Eric Rescorla. The transport layer security (TLS) protocol version 1.2. Technical report, RFC 5246, August, 2008. (visitato il giorno 12/10/2026). URL: https://datatracker.ietf.org/doc/html/rfc5246.

[36] National Institute of Standards and Technology. Advanced Encryption Standard (AES). Technical report, FIPS PUB 197, 2001. (visitato il giorno 12/10/2026). URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[37] J. R. Storlie and J. R. Fuller. *Cloud FinOps: Collaborative, Real-Time Cloud Financial Management*. O'Reilly Media, 2019. (visitato il giorno 12/10/2026). URL: https://www.oreilly.com/library/view/cloud-finops-2nd/9781492098348/.

[38] NVIDIA. Reducing Cold Start Latency for LLM Inference with NVIDIA Run:ai, 2025. (visitato il giorno 12/10/2026). URL: https://developer.nvidia.com/blog/reducing-cold-start-latency-for-llm-inference-with-nvidia-runai-model-streamer.

[39] Simon Josefsson. The Base16, Base32, and Base64 Data Encodings. Technical report, RFC 4648, October, 2006. (visitato il giorno 12/10/2026). URL: https://datatracker.ietf.org/doc/html/rfc4648.

[40] Amazon Web Services. Authenticating Requests (AWS Signature Version 4), 2024. (visitato il giorno 12/10/2026). URL: https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_sigv.html.

[41] Richard Light et al. MQTT Version 3.1.1. Technical report, OASIS Standard, 2014. (visitato il giorno 12/10/2026). URL: `https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html`.

[42] Amazon Web Services. Amazon sqs visibility timeout, 2024. (visitato il giorno 12/10/2026). URL: `https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-visibility-timeout.html`.

[43] Amazon Web Services. Amazon SQS message quotas and payload limits, 2024. (visitato il giorno 12/10/2026). URL: `https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/quotas-messages.html`.

[44] Object Management Group (OMG). Unified Modeling Language (UML) Specification, Version 2.5.1. Technical report, Object Management Group (OMG), 2017. (visitato il giorno 12/10/2026). URL: `https://www.omg.org/spec/UML/2.5.1/PDF`.

[45] Amazon Web Services. Using Amazon CloudWatch alarms to stop EC2 instances, 2024. (visitato il giorno 12/10/2026). URL: `https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/UsingAlarmActions.html`.

[46] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010. (visitato il giorno 12/10/2026). URL: `https://dl.acm.org/doi/10.1145/1721654.1721672`.

[47] Amazon Web Services. Evaluating an alarm: state transitions and evaluation periods, 2024. (visitato il giorno 12/10/2026). URL: `https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/AlarmThatSendsEmail.html#alarm-evaluation`.

[48] FinOps Foundation. What is FinOps?, 2024. (visitato il giorno 12/10/2026). URL: `https://www.finops.org/introduction/what-is-finops/`.

[49] Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. The rise of serverless computing. *Communications of the ACM*, 62(12):44–54, 2019. (visitato il giorno 12/10/2026). URL: `https://dl.acm.org/doi/10.1145/3368454`.

[50] Mark Richards. *Software Architecture Patterns: Understanding Common Architecture Patterns and When to Use Them*. O'Reilly Media, 2015.

(visitato il giorno 12/10/2026). URL: https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/.

[51] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014. (visitato il giorno 12/10/2026). URL: https://link.springer.com/chapter/10.1007/978-3-319-10602-1_48.

[52] Glenn Jocher et al. Ultralytics YOLO11: High-Performance Real-Time Object Detection, 2024. (visitato il giorno 12/10/2026). URL: https://docs.ultralytics.com/models/yolo11/.

[53] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. (visitato il giorno 12/10/2026). URL: https://arxiv.org/abs/1804.02767.

[54] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017. (visitato il giorno 12/10/2026). URL: https://dl.acm.org/doi/10.1145/3079856.3080246.

[55] NVIDIA Corporation. NVIDIA Turing Architecture In-Depth. Technical report, NVIDIA Whitepapers, 2018. (visitato il giorno 12/10/2026). URL: https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf.

[56] Junchen Jiang et al. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 253–266, 2018. (visitato il giorno 12/10/2026). URL: https://dl.acm.org/doi/10.1145/3230543.3230574.

[57] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional, 2004. (visitato il giorno 12/10/2026). URL: https://www.enterpriseintegrationpatterns.com/.

[58] Yuan Liu, Haodong Duan, Yuanhan Zhang, Xin Li, Peize Zhang, Wei Zou, Mingrou Xing, Chongyi Chen, Antonin Zeng, Wenqi Shao, et al. MMBench: Is your multi-modal model an all-around player? *arXiv preprint arXiv:2307.01415*, 2023. (visitato il giorno 12/10/2026). URL: https://arxiv.org/abs/2307.06281.

[59] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang. Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing. *Proceedings of the IEEE*, 107(8):1738–1762, 2019. (visitato il giorno 12/10/2026). URL: `https://ieeexplore.ieee.org/document/8736011`.

[60] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics YOLO, 2023. (visitato il giorno 12/10/2026). URL: `https://github.com/ultralytics/ultralytics`.

[61] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual Instruction Tuning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. (visitato il giorno 12/10/2026). URL: `https://arxiv.org/abs/2304.08485`.

[62] Peng Wang, Shuai Bai, et al. Qwen2-VL: Enhancing Vision-Language Model's Perception of the World at Any Resolution. *arXiv preprint arXiv:2409.12191*, 2024. (visitato il giorno 12/10/2026). URL: `https://arxiv.org/abs/2409.12191`.

[63] Amazon Web Services. Amazon EC2 Pricing: Pay only for what you use, 2024. (visitato il giorno 12/10/2026). URL: `https://aws.amazon.com/ec2/pricing/`.

[64] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg. A Survey of Research on Cloud Robotics and Automation. *IEEE Transactions on Automation Science and Engineering*, 12(2):398–409, 2015. (visitato il giorno 12/10/2026). URL: `https://ieeexplore.ieee.org/document/7006734`.

[65] Yifan Li, Yifan Du, Kun Lin, Zheng Chen, Ming Li, et al. Evaluating Object Hallucination in Large Vision-Language Models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 292–305. Association for Computational Linguistics, 2023. (visitato il giorno 12/10/2026). URL: `https://aclanthology.org/2023.emnlp-main.20`.

[66] Kevin Fall. A delay-tolerant network architecture for challenged internets. *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (ACM SIGCOMM)*, pages 27–34, 2003. (visitato il giorno 12/10/2026). URL: `https://dl.acm.org/doi/10.1145/863955.863960`.

[67] Tarik Taleb, Konstantinos Samdanis, B. Mada, Hannu Flinck, Ashish Dutta, and Dario Sabella. On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681, 2017.

(visitato il giorno 12/10/2026). URL: https://ieeexplore.ieee.org/document/7931566.

[68] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020. (visitato il giorno 12/10/2026). URL: https://arxiv.org/abs/2005.11401.

[69] Naser Hossein Motlagh, Miloud Bagaa, and Tarik Taleb. UAV-based IoT platform: A crowd surveillance use case. *IEEE Communications Magazine*, 55(2):128–134, 2017. (visitato il giorno 12/10/2026). URL: https://ieeexplore.ieee.org/document/7842423.