Scalable and Reliable Systems nell'era dell'Artificial Intelligence

Architetture, operations e resilienza

White paper

Luglio 2026

Indice

1. Executive Summary

Questo white paper esamina il rapporto, divenuto bidirezionale, tra intelligenza artificiale e ingegneria dei sistemi distribuiti: da un lato le architetture scalabili e affidabili che rendono possibile l'AI su larga scala, dall'altro l'impiego dell'AI per gestire la scalabilità e l'affidabilità dei sistemi stessi. Le due direttrici, analizzate rispettivamente nella Parte I e nella Parte II, vengono ricondotte a un piano comune, la resilienza, in cui reliability e security sono trattate come proprietà congiunte. L'analisi si fonda esclusivamente su letteratura peer-reviewed recente, rapporti di enti di standardizzazione e telemetria di produzione pubblicamente documentata.

Sul primo versante, l'evidenza empirica è netta: alla scala attuale il guasto è una condizione operativa permanente, non un'eccezione. Durante i 54 giorni di pre-training di Llama 3 405B su 16.384 GPU, Meta ha registrato un'interruzione imprevista ogni tre ore circa, per oltre tre quarti riconducibile all'hardware, mantenendo comunque un tempo utile di addestramento superiore al 90%. La frontiera si sposta rapidamente: il checkpointing in memoria consente oggi salvataggi a ogni iterazione senza penalità di throughput, e i sistemi di gestione più recenti, dispiegati su piattaforme con oltre 200.000 GPU, raggiungono il 97% di *Effective Training Time Ratio* su job trimestrali. Sul fronte dell'esercizio, tre svolte sistemistiche (continuous batching, gestione paginata della memoria, disaggregazione delle fasi di prefill e decode) hanno moltiplicato l'efficienza dell'*inference serving*, e il *goodput* (le richieste servite nel rispetto degli SLO di latenza) si è affermato come metrica di riferimento del servizio.

Sul secondo versante, i large language model stanno ridefinendo l'AIOps lungo l'intero ciclo di vita del guasto, dalla rilevazione alla mitigazione. I sistemi che hanno raggiunto la produzione condividono una lezione architetturale: l'affidabilità dell'assistente dipende meno dal modello che dal flusso di lavoro entro cui opera, vincolato a evidenze raccolte e verificabili. Quanto all'autonomia piena, i benchmark più recenti impongono sobrietà (gli agenti allo stato dell'arte risolvono circa un incidente di site reliability su dieci) e indicano al tempo stesso la via: le architetture che formalizzano garanzie di sicurezza e reversibilità delle azioni superano nettamente quelle che non lo fanno.

La prospettiva di security attraversa entrambe le direttrici. La tassonomia NIST dell'*adversarial machine learning* mostra un threat model che investe dati, modelli e contesto; il poisoning dei dataset web-scale è dimostrato praticabile; la *prompt injection* indiretta trasforma ogni sorgente di contesto in un potenziale canale di comando, inclusa la telemetria letta dagli agenti che amministrano l'infrastruttura. Il documento sostiene che fault tolerance e adversarial robustness siano le facce accidentale e intenzionale di un medesimo spazio dei guasti, con una disciplina di risposta condivisa e modelli di minaccia distinti, e che osservabilità, integrità dei log e *forensic readiness* siano requisiti di progetto, oltre che, per i sistemi ad alto rischio, un obbligo normativo ai sensi dell'AI Act europeo.

Restano aperte tre questioni che definiscono l'agenda dei prossimi anni: il denominatore energetico della scala, con i consumi dei data center destinati a più che raddoppiare entro il 2030 secondo l'Agenzia Internazionale dell'Energia; la valutazione di sistemi non deterministici, oggi il principale fattore limitante della fiducia; e il paradosso di un'autonomia che deve essere a sua volta garantita. La conclusione operativa del documento: prima di chiedersi quando l'infrastruttura si amministrerà da sé, conviene stabilire quali classi di decisioni delegare per prime, con quali garanzie verificabili e sotto quale supervisione umana.

2. Introduzione

2.1 La doppia convergenza tra AI e systems engineering

Nel corso dei 54 giorni di pre-training di Llama 3 405B, condotto su un cluster di 16.384 GPU NVIDIA H100, Meta ha registrato 466 interruzioni del job di addestramento, 419 delle quali impreviste: in media, una ogni tre ore. Circa il 78% degli eventi imprevisti è stato ricondotto a guasti hardware, confermati o sospetti, e le sole componenti GPU sono risultate responsabili del 58,7% dei casi. Il team è riuscito ciononostante a mantenere un *effective training time* superiore al 90% (Llama Team, AI @ Meta, 2024). Questo dato, tratto dalla documentazione tecnica di uno dei più grandi addestramenti mai resi pubblici, sintetizza la prima tesi del presente documento: alla scala dell'intelligenza artificiale contemporanea il guasto non è un evento eccezionale da prevenire, ma una condizione operativa permanente da governare.

L'ingegneria dei sistemi distribuiti ha maturato in oltre due decenni un repertorio consolidato di tecniche per convivere con il guasto: ridondanza, replicazione, *stateless design*, *graceful degradation*. Quel repertorio poggia però su un'assunzione che i workload di AI mettono in discussione, ossia che il fallimento di un componente possa essere isolato e assorbito senza compromettere il servizio nel suo complesso. Il training distribuito di un *large language model* è, al contrario, un calcolo strettamente sincrono: a ogni step di ottimizzazione i gradienti prodotti da migliaia di acceleratori devono essere aggregati prima di poter procedere: il guasto, o anche il solo rallentamento, di una singola GPU può arrestare o degradare l'intero cluster. La probabilità di interruzione cresce dunque con la dimensione del sistema, proprio mentre il costo di ogni ora di inattività raggiunge ordini di grandezza in precedenza sconosciuti.

La letteratura sistemistica più recente tratta di conseguenza la stabilità come un obiettivo di progetto al pari del throughput. MegaScale, il sistema di produzione descritto da Jiang et al. (2024) per l'addestramento oltre la soglia delle 10.000 GPU, co-progetta componenti algoritmiche e infrastrutturali (dalla sovrapposizione di calcolo e comunicazione alla diagnostica automatica di guasti e *straggler*) e raggiunge una *Model FLOPs Utilization* del 55,2% nell'addestramento di un modello da 175 miliardi di parametri su 12.288 GPU. Le analisi longitudinali condotte sui cluster di ricerca di Meta (Kokolis et al., 2024) confermano che l'affidabilità dell'hardware è oggi tra i principali fattori limitanti della scala raggiungibile. Considerazioni analoghe valgono per la fase di esercizio dei modelli: sul versante dell'*inference serving*, sistemi come vLLM hanno dimostrato che la gestione della memoria e lo scheduling delle richieste determinano, a parità di hardware, differenze di throughput comprese tra due e quattro volte (Kwon et al., 2023). La Parte I del documento è dedicata a questa prima direttrice.

Il movimento inverso è altrettanto rilevante. La complessità operativa delle infrastrutture moderne (architetture a microservizi con migliaia di componenti, telemetria nell'ordine dei terabyte al giorno, catene di dipendenze che nessun singolo operatore può padroneggiare) ha da tempo superato la capacità di analisi manuale. La disciplina nota come *AIOps* (Artificial

Intelligence for IT Operations) applica tecniche di machine learning ad *anomaly detection*, *root cause analysis* e *incident management*; la diffusione dei large language model ne ha accelerato l'evoluzione in misura tale da ridefinirne i confini. La survey pubblicata da Zhang et al. (2025) su ACM Computing Surveys, che analizza 183 lavori apparsi tra il 2020 e il 2024, documenta lo spostamento del baricentro della ricerca verso approcci LLM-based lungo l'intero ciclo di vita del guasto, dalla rilevazione alla remediation. A questa seconda direttrice è dedicata la Parte II.

Considerate congiuntamente, le due direttrici delineano la circolarità che costituisce il filo conduttore del documento e che la Figura 1 rappresenta in forma schematica: l'AI in produzione dipende da infrastrutture la cui gestione è, a sua volta, sempre più delegata all'AI. Si tratta di una circolarità potenzialmente virtuosa, poiché ciascun dominio compensa i limiti dell'altro; essa introduce tuttavia modalità di fallimento inedite e amplia la superficie d'attacco complessiva. Un agente autonomo dotato di privilegi operativi sull'infrastruttura è, dal punto di vista di un avversario, un bersaglio di elevato valore, e la tassonomia NIST sull'*adversarial machine learning* (Vassilev et al., 2025) mostra quanto il threat model dei sistemi AI si discosti da quello del software tradizionale. Il capitolo 6 sviluppa questa prospettiva, muovendo dalla convinzione che reliability e security siano proprietà inseparabili di uno stesso obiettivo di progetto: la resilienza.

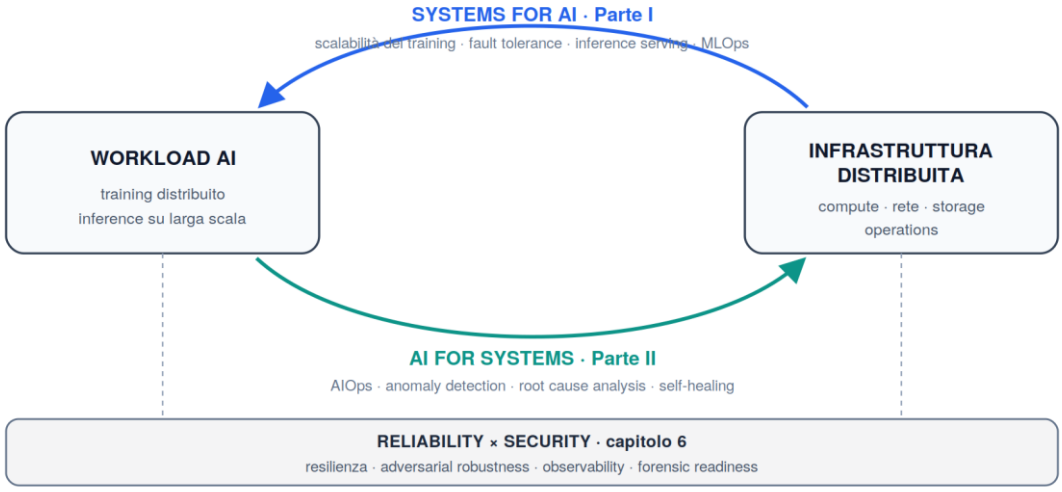

Figura 1 — *La doppia convergenza tra artificial intelligence e systems engineering. La direttrice superiore (Parte I) riguarda le infrastrutture scalabili e affidabili che rendono possibili training e inference su larga scala; quella inferiore (Parte II) riguarda l'impiego dell'AI per la gestione autonoma dell'infrastruttura stessa. Il piano trasversale della resilienza (capitolo 6) unifica le due prospettive sotto il profilo di reliability e security.*

2.2 Obiettivi, ambito e struttura del documento

Il presente white paper si rivolge a un pubblico misto e persegue, di conseguenza, un obiettivo duplice: offrire ai decision maker gli elementi necessari per valutare maturità tecnologica, rischi e costi delle soluzioni disponibili, e fornire alle figure tecniche una sintesi ragionata dello stato

dell'arte, ancorata alle fonti primarie e utilizzabile come punto di ingresso nella letteratura. Le due esigenze convivono in ogni capitolo: le sezioni introduttive privilegiano il quadro concettuale e le implicazioni operative, mentre gli approfondimenti tecnici (architetture, meccanismi, risultati sperimentali) restano circoscritti e sempre riconducibili alle pubblicazioni originali.

Il percorso argomentativo si articola come segue. Il capitolo 3 stabilisce il lessico comune, richiamando le metriche classiche di scalability e reliability e mostrando perché i workload di AI ne impongano una rilettura. Il capitolo 4 (Parte I) analizza i sistemi al servizio dell'AI: il training distribuito su larga scala, la fault tolerance che lo rende sostenibile, l'inference serving e le pratiche MLOps che ne costituiscono la cerniera organizzativa (Kreuzberger et al., 2023). Il capitolo 5 (Parte II) esamina il percorso inverso, dall'AIOps di prima generazione agli agenti basati su LLM per la gestione autonoma dell'infrastruttura. Il capitolo 6 riconduce le due prospettive a un piano comune, quello della resilienza, in cui reliability e security vengono trattate come proprietà congiunte; il capitolo 7 discute infine le sfide aperte e le direzioni di ricerca, prima delle conclusioni del capitolo 8.

Quanto all'ambito, il documento si concentra sui sistemi distribuiti in esercizio, con particolare attenzione ai workload basati su large language model, che rappresentano oggi il banco di prova più severo per scalabilità e affidabilità; i principi discussi mantengono tuttavia validità generale. Restano fuori dal perimetro la progettazione degli acceleratori hardware e gli aspetti puramente algoritmici dell'apprendimento, richiamati soltanto dove indispensabili alla comprensione dei vincoli sistemistici.

2.3 Metodologia e criteri di selezione delle fonti

Il corpus bibliografico privilegia la letteratura peer-reviewed pubblicata tra il 2022 e il 2025 nelle sedi primarie della ricerca sistemistica (USENIX NSDI e OSDI, ACM SOSP, ISCA, MLSys) e nelle riviste di riferimento del settore, tra cui IEEE Access e ACM Computing Surveys, integrata dai rapporti degli enti di standardizzazione, in primo luogo il NIST, e da documentazione tecnica di provenienza industriale.

Tre criteri hanno guidato la selezione. In primo luogo la verificabilità: ogni affermazione quantitativa del documento è riconducibile a una fonte pubblicamente accessibile, elencata nel capitolo 9. In secondo luogo la preferenza per lavori corredati da dati di produzione o da valutazioni sperimentali riproducibili, e per survey a metodologia esplicita. In terzo luogo un uso disciplinato delle fonti industriali: i rapporti tecnici delle grandi piattaforme vengono trattati come fonti primarie di dati telemetrici, spesso inaccessibili alla ricerca accademica indipendente, e non come fonti interpretative, per le quali si rimanda alla letteratura sottoposta a revisione paritaria.

Sul piano delle convenzioni, la terminologia tecnica inglese consolidata (*fault tolerance*, *checkpointing*, *serving*, *observability*) non viene tradotta ed è resa in corsivo alla prima

occorrenza; le citazioni seguono il formato autore-anno, con bibliografia completa in chiusura di documento.

3. Fondamenti: scalability e reliability nei sistemi distribuiti moderni

3.1 Definizioni e metriche

Il lessico dell'affidabilità informatica ha il proprio punto di riferimento nella tassonomia di Avizienis, Laprie, Randell e Landwehr (2004), che organizza sotto il concetto ombrello di *dependability* un insieme di attributi distinti: la *reliability*, intesa come continuità del servizio corretto; la *availability*, come prontezza del servizio quando richiesto; e con esse *safety*, *integrity* e *maintainability*. La tassonomia introduce inoltre la catena causale che collega *fault*, *error* e *failure* (il difetto latente, il suo effetto sullo stato interno, la deviazione osservabile del servizio) e classifica i mezzi per contrastarla in quattro famiglie: prevenzione, tolleranza, rimozione e previsione dei guasti. Questa distinzione, apparentemente accademica, ha conseguenze pratiche immediate: un sistema si progetta in modo diverso a seconda che l'obiettivo sia impedire il fault, contenere l'error o mascherare la failure.

La quantificazione di questi attributi passa per un piccolo insieme di grandezze consolidate. La availability di un sistema riparabile si esprime come rapporto tra il tempo medio al guasto e il tempo totale di un ciclo guasto-ripristino, A = MTTF / (MTTF + MTTR), dove il *Mean Time To Failure* misura quanto a lungo il sistema opera correttamente e il *Mean Time To Repair* quanto rapidamente torna in servizio; la somma delle due grandezze costituisce l'MTBF, *Mean Time Between Failures*. La formulazione rende esplicito un principio spesso trascurato: a parità di frequenza dei guasti, dimezzare il tempo di ripristino produce lo stesso beneficio che raddoppiare il tempo tra un guasto e l'altro. La pratica contemporanea, prendendo atto che nei sistemi su larga scala il guasto non è eliminabile, ha progressivamente spostato l'investimento ingegneristico dalla prevenzione alla rapidità di recovery. Le soglie di availability si esprimono per convenzione in «nove»: un servizio al 99,9% ammette circa otto ore e quarantacinque minuti di indisponibilità l'anno, uno al 99,999% poco più di cinque minuti.

Sul piano operativo, la disciplina del *Site Reliability Engineering* ha tradotto questi concetti in un apparato di gestione oggi largamente adottato (Beyer et al., 2016). Un *Service Level Indicator* (SLI) è una grandezza misurata (la frazione di richieste servite entro una soglia di latenza, il tasso di errori), mentre il *Service Level Objective* (SLO) è il valore obiettivo che l'organizzazione si impegna internamente a rispettare, e il *Service Level Agreement* (SLA) la sua eventuale controparte contrattuale. Dalla distanza tra SLI e SLO discende la nozione di *error budget*: la quota di inaffidabilità che il servizio può ancora «spendere» nel periodo, utilizzabile come criterio decisionale per bilanciare velocità di rilascio e stabilità. L'affidabilità cessa così di essere un vincolo assoluto e diventa una risorsa finita, allocabile in modo deliberato.

Le metriche di latenza meritano un discorso a parte, perché nei sistemi distribuiti il valore medio è quasi privo di significato ingegneristico. Ciò che governa l'esperienza dell'utente e la stabilità dell'architettura è la coda della distribuzione, descritta dai percentili alti (p99, p99.9). Dean e Barroso (2013) hanno mostrato come la scala amplifichi le code per un effetto puramente

combinatorio: se un servizio deve attendere la risposta di cento backend e ciascuno ha una probabilità dell'1% di rispondere lentamente, circa il 63% delle richieste complessive subirà almeno una risposta lenta. La loro proposta (tecniche *tail-tolerant* come *hedged request* e repliche selettive, in analogia con la fault tolerance) muove dalla constatazione che eliminare ogni fonte di variabilità è impraticabile e che occorre invece costruire un insieme prevedibile a partire da parti che non lo sono.

Quanto alla scalability, la teoria classica ne fissa i limiti e le condizioni. La legge di Amdahl (1967) mostra che la frazione seriale di un calcolo pone un tetto invalicabile allo *speedup* ottenibile aggiungendo processori; la rilettura di Gustafson (1988) osserva che, nella pratica, all'aumentare delle risorse si affrontano problemi più grandi, e la scalabilità va dunque misurata a carico crescente (*weak scaling*) oltre che a carico fisso (*strong scaling*). Nella prospettiva ingegneristica odierna la scalabilità di un sistema si valuta come capacità di aumentare il throughput in proporzione alle risorse impiegate mantenendo pressoché costante il costo marginale per unità di lavoro — una definizione che, come si vedrà, i workload di AI mettono sotto tensione da entrambi i lati del rapporto.

3.2 Perché i workload AI cambiano le regole

Le grandezze appena richiamate restano valide per i sistemi di AI su larga scala, ma il modo in cui si compongono cambia al punto da richiedere una rilettura complessiva. Tre discontinuità strutturali lo determinano. La prima riguarda il *failure domain*: nelle architetture a servizi il perimetro di impatto di un guasto coincide, per progetto, con una porzione piccola e sostituibile del sistema, mentre nel training sincrono l'intero cluster costituisce un unico dominio di guasto, perché ogni step di ottimizzazione richiede il contributo di tutti gli acceleratori. La seconda è statistica: a parità di affidabilità del singolo componente, la frequenza dei guasti di un job cresce all'incirca linearmente con il numero di componenti coinvolti, e i job di addestramento occupano decine di migliaia di acceleratori per settimane o mesi consecutivi. La terza è economica: ogni ora di stallo di un cluster di training immobilizza un capitale (hardware, energia, tempo di calendario) che non ha precedenti nella storia dei sistemi informativi, e sposta il costo del downtime dal mancato servizio al mancato progresso.

L'evidenza empirica disponibile è ampia e convergente. L'analisi condotta da Kokolis et al. (2024) su due cluster ML multi-tenant di Meta (undici mesi di telemetria, circa quattro milioni di job e 150 milioni di GPU-hour) mostra che i job di grandi dimensioni sono di gran lunga i più esposti ai guasti, propone un modello predittivo dell'MTTF in funzione della scala e introduce la metrica ETTR (*Effective Training Time Ratio*), che rapporta il tempo di addestramento effettivamente produttivo al tempo totale di occupazione delle risorse. Il caso Llama 3 discusso nel capitolo 2 fornisce il riscontro di produzione: un'interruzione imprevista circa ogni tre ore su 16.384 GPU. Se il tasso di guasto per componente resta costante, la stessa aritmetica implica che un cluster dieci volte più grande debba attendersi un MTTF dell'ordine delle decine di minuti; è la ragione

per cui la letteratura sistemistica recente tratta la fault tolerance non come un requisito accessorio ma come la condizione di possibilità del training a questa scala (Jiang et al., 2024).

Ne discende una reinterpretazione delle metriche. La availability, pensata per servizi che o rispondono o non rispondono, lascia il posto a misure di avanzamento utile: l'effective training time e l'ETTR scontano dal tempo totale non solo gli stalli ma anche il lavoro perduto e ricalcolato dopo ogni ripristino, e la *Model FLOPs Utilization* misura quanta parte della capacità teorica di calcolo si traduce in addestramento effettivo. Il 55,2% riportato da MegaScale su 12.288 GPU rappresenta, in questo senso, un risultato di eccellenza ingegneristica più che un dato scontato (Jiang et al., 2024). Anche l'MTTR va decomposto: il ripristino di un job comprende la rilevazione del guasto, la diagnosi del componente responsabile, la riallocazione delle risorse e il ricaricamento dell'ultimo *checkpoint* valido, e ciascuna fase è oggetto di ottimizzazioni dedicate che il capitolo 4 esamina in dettaglio.

Considerazioni speculari valgono per l'inference. La generazione autoregressiva ha una proprietà che i modelli di costo tradizionali non contemplano: il costo di una richiesta non è noto a priori, perché dipende dalla lunghezza dell'output che il modello stesso produrrà. Il servizio si articola inoltre in due fasi dal profilo opposto (il *prefill*, limitato dal calcolo, e il *decode*, limitato dalla banda di memoria); le metriche di latenza si sdoppiano di conseguenza: il *Time To First Token* (TTFT) misura la reattività percepita, il *Time Per Output Token* (TPOT) la fluidità della generazione, e la letteratura definisce *goodput* il numero di richieste al secondo servite nel rispetto di entrambi gli SLO (Zhong et al., 2024). La coesistenza di richieste eterogenee nello stesso batch genera infine interferenze che gonfiano la coda della distribuzione di latenza, un compromesso tra throughput e tail latency che sistemi come Sarathi-Serve affrontano con tecniche di scheduling dedicate (Agrawal et al., 2024) e che la gestione della memoria del KV cache condiziona alla radice (Kwon et al., 2023).

La Tabella 1 riepiloga il quadro, affiancando a ciascuna metrica classica la sua declinazione nei workload di AI.

Tabella 1 — *Metriche classiche di reliability e scalability e loro rilettura per i workload AI.*

Metrica classica	Definizione sintetica	Rilettura per i workload AI	Fonti principali
Availability	Frazione di tempo in cui il servizio è operativo; A = MTTF/(MTTF+MTTR)	*Effective training time* ed ETTR per il training; *goodput* sotto SLO per il serving	Avizienis et al. 2004; Kokolis et al. 2024; Zhong et al. 2024
MTTF / MTBF	Tempo medio di funzionamento corretto tra due guasti	Decresce in modo circa proporzionale al numero di acceleratori; ~3 h su 16.384 GPU	Llama Team 2024; Kokolis et al. 2024
MTTR	Tempo medio di ripristino del servizio	Somma di detection, diagnosi, rescheduling e restore da checkpoint; obiettivo nell'ordine dei minuti	Jiang et al. 2024

Metrica classica	Definizione sintetica	Rilettura per i workload AI	Fonti principali
Latency (percentili)	Distribuzione dei tempi di risposta; rilevanza dei percentili alti (p99)	TTFT e TPOT per la generazione autoregressiva; tail sensibile a batching e lunghezza variabile dell'output	Dean & Barroso 2013; Zhong et al. 2024; Agrawal et al. 2024
Throughput	Unità di lavoro completate per unità di tempo	Token/s per il serving; *Model FLOPs Utilization* per il training	Kwon et al. 2023; Jiang et al. 2024
Scalability (speedup, efficienza)	Guadagno di prestazioni al crescere delle risorse (Amdahl, Gustafson)	*Scaling efficiency* del training distribuito; costo marginale per token addestrato o generato	Amdahl 1967; Gustafson 1988; Jiang et al. 2024

Il capitolo successivo entra nel merito della prima direttrice, esaminando come le architetture di training e di serving traducano questi principi in meccanismi concreti.

4. Parte I — Sistemi scalabili e affidabili per l'AI

Questo capitolo esamina la prima direttrice della convergenza delineata nell'introduzione: le architetture che rendono possibile l'esercizio dell'AI su larga scala. La trattazione segue il ciclo di vita del modello (addestramento, tolleranza ai guasti che lo sostiene, messa in esercizio) per chiudersi sulle pratiche MLOps che ne governano la dimensione organizzativa. Il filo conduttore resta quello fissato nel capitolo 3: a questa scala scalabilità e affidabilità non sono proprietà separabili, perché ogni tecnica di parallelizzazione ridisegna la superficie di guasto e ogni meccanismo di recovery incide sul throughput.

4.1 Training distribuito su larga scala

La distribuzione del training risponde a una necessità dimensionale prima ancora che a un obiettivo di efficienza: i parametri di un modello di frontiera, con i relativi stati dell'ottimizzatore e le attivazioni, eccedono di ordini di grandezza la memoria di un singolo acceleratore, e il volume di calcolo richiesto renderebbe comunque proibitivi i tempi di un'esecuzione non parallela. La letteratura ha consolidato tre assi di parallelizzazione, componibili tra loro, ciascuno con un profilo distinto di comunicazione e quindi di vincoli topologici.

Il *data parallelism* replica l'intero modello su gruppi di acceleratori e ripartisce tra le repliche i dati di ciascun batch; al termine di ogni step i gradienti vengono aggregati con un'operazione collettiva di *all-reduce*, che costituisce il punto di sincronizzazione globale del sistema. La replica integrale dello stato comporta però una ridondanza di memoria che cresce con il numero di repliche: la famiglia di tecniche ZeRO la elimina progressivamente, partizionando tra i ranghi stati dell'ottimizzatore, gradienti e parametri, e ha reso praticabile l'addestramento di modelli nell'ordine del trilione di parametri (Rajbhandari et al., 2020).

Il *tensor parallelism* opera a grana più fine, suddividendo le singole operazioni di un layer (le grandi moltiplicazioni matriciali dei blocchi Transformer) tra più acceleratori. Il prezzo è una comunicazione intensa e frequente all'interno di ogni forward e backward pass, che ne confina l'impiego entro il perimetro del singolo nodo, dove interconnessioni dedicate ad altissima banda come NVLink rendono sostenibile il traffico (Narayanan et al., 2021). Il *pipeline parallelism*, infine, distribuisce i layer del modello in stadi consecutivi assegnati a gruppi diversi di acceleratori: il batch viene frazionato in micro-batch che attraversano la pipeline in sequenza, e la schedulazione dei micro-batch mira a ridurre la «bolla» di inattività che si forma alle estremità di ogni step.

Nei sistemi di produzione i tre assi vengono composti in un parallelismo tridimensionale la cui mappatura sulla topologia fisica segue una regola divenuta canonica: tensor parallelism entro il nodo, pipeline parallelism tra nodi vicini, data parallelism alla scala più ampia. Su questa base, l'ingegneria della scala si gioca nella sovrapposizione di calcolo e comunicazione e nella rimozione sistematica dei colli di bottiglia: MegaScale documenta un approccio *full-stack* che co-progetta blocchi del modello, ottimizzatore, kernel, pipeline dei dati e tuning della rete,

raggiungendo una Model FLOPs Utilization del 55,2% su 12.288 GPU, con un incremento del 34% rispetto alla baseline Megatron-LM sullo stesso carico (Jiang et al., 2024).

La conseguenza architetturale rilevante ai fini di questo documento è visibile nella Figura 2: la griglia di parallelizzazione che rende il training possibile è, al tempo stesso, la sua superficie di guasto. Ogni acceleratore siede sul cammino critico di ogni step, attraverso il proprio gruppo tensor, il proprio stadio di pipeline e la sincronizzazione globale dei gradienti: non esiste dunque, per costruzione, alcuna porzione del sistema il cui fallimento sia localmente assorbibile. È la premessa da cui muove la sezione successiva.

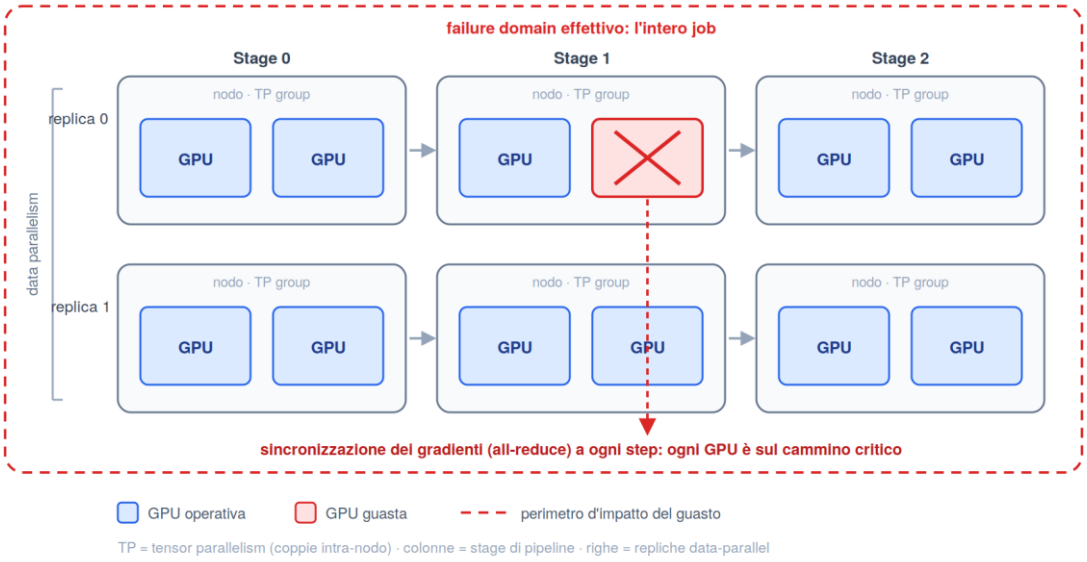

Figura 2 — *Composizione degli assi di parallelismo (tensor, pipeline, data) e inversione del failure domain: il guasto di un singolo acceleratore si propaga, attraverso la sincronizzazione dei gradienti, all'intero job di training.*

4.2 Fault tolerance del training: checkpointing, detection, isolamento

Il problema era noto ben prima della scala attuale: già l'addestramento di OPT-175B, condotto su 992 GPU A100, aveva registrato circa 110 failure in due mesi (dato richiamato in Wang et al., 2023). Ciò che è cambiato è l'ordine di grandezza (dei componenti coinvolti, della durata dei job, del capitale immobilizzato) e con esso la natura della risposta ingegneristica, che si articola oggi su tre fronti complementari: preservare lo stato, rilevare il guasto, isolarlo e ripartire.

La preservazione dello stato poggia sul *checkpointing*, il salvataggio periodico di parametri e stati dell'ottimizzatore da cui riprendere l'esecuzione dopo un'interruzione. La sua calibrazione è un problema di ottimizzazione a tre variabili: la frequenza dei salvataggi, il loro costo in termini di throughput sottratto al training, e la quantità di lavoro perduto, e da ricalcolare, tra l'ultimo checkpoint valido e il guasto. Con stati del modello che raggiungono centinaia di gigabyte o alcuni terabyte, l'approccio tradizionale su storage remoto persistente trova nel collo di bottiglia della

banda il proprio limite: frequenze di salvataggio basse, finestre di lavoro perduto ampie. La risposta più incisiva è venuta dallo spostamento del primo livello di persistenza nella memoria CPU aggregata degli host di training: Gemini dimostra che una strategia di piazzamento delle copie prossima all'ottimo, unita a una schedulazione che interfoglia il traffico di checkpoint con quello di addestramento, consente salvataggi a ogni iterazione senza penalità di throughput e un recovery oltre tredici volte più rapido delle soluzioni su storage remoto (Wang et al., 2023). Sistemi unificati come ByteCheckpoint (Wan et al., 2024) estendono l'approccio all'intero ciclo di sviluppo dei foundation model, dove i checkpoint devono essere letti e riscritti anche al variare della configurazione di parallelismo.

La rilevazione è il fronte più insidioso, perché i guasti più costosi non sono quelli che si annunciano. Un'ampia classe di malfunzionamenti (errori CUDA, valori NaN, cali di prestazione di singoli componenti) si manifesta come *hang* silenzioso o come degrado, e nei framework distribuiti standard l'unico segnale di default è un timeout delle operazioni collettive dell'ordine dei dieci minuti (Wan et al., 2025): a 16.000 GPU, dieci minuti di stallo collettivo sono una quantità industriale di calcolo perduto. Da qui l'investimento in strumentazione dedicata documentato dai sistemi di produzione: MegaScale descrive tool diagnostici e meccanismi di registrazione degli eventi collettivi per individuare rapidamente guasti e straggler (Jiang et al., 2024), mentre l'esperienza Llama 3 mostra il traguardo raggiunto dall'automazione, con soltanto tre interruzioni su 419 che hanno richiesto un intervento manuale significativo (Llama Team, AI @ Meta, 2024). Gli straggler meritano una menzione specifica: un componente lento ma vivo sfugge ai rilevatori di guasto binari e, in un calcolo sincrono, impone il proprio passo all'intero cluster. La loro individuazione richiede baseline prestazionali per componente e un'analisi comparativa continua.

Il terzo fronte è l'isolamento e la ripartenza, e su questo terreno la filosofia operativa più recente segna uno scarto. ByteRobust, il sistema di gestione dell'infrastruttura di training di ByteDance, dichiara esplicitamente la priorità della rapidità di isolamento sulla precisione della localizzazione: individuare il colpevole esatto può richiedere diagnostiche lunghe che lasciano inattive migliaia di GPU, mentre espellere rapidamente la macchina sospetta (combinando rilevazione leggera in tempo reale, diagnostica gerarchica a job fermo e clustering data-driven degli stack trace) massimizza il tempo utile di addestramento. L'insieme dei meccanismi di failover (aggiornamenti a caldo aggregati, macchine di riserva già inizializzate, checkpointing consapevole dei guasti) è dispiegato su una piattaforma di produzione con oltre 200.000 GPU e ha consentito un ETTR del 97% su un job di tre mesi a 9.600 GPU (Wan et al., 2025). Il confronto con il 90% di effective training time riportato per Llama 3 appena un anno prima dà la misura della velocità con cui questa frontiera si sta spostando, e conferma la lettura proposta nel capitolo 3: nel training su larga scala l'affidabilità non è una proprietà accessoria: è il vincolo attorno al quale l'intero sistema viene progettato.

4.3 Inference serving: dal batching continuo alla disaggregazione

Il passaggio all'esercizio rovescia il regime operativo. Dove il training è un singolo calcolo lungo e sincrono, il serving è un flusso continuo di richieste brevi, eterogenee e vincolate da SLO di latenza; la scalabilità non si misura più in FLOPs utilizzati ma in richieste servite entro soglia per unità di hardware, e l'affidabilità torna ad assumere la forma classica della disponibilità del servizio, con alcune complicazioni peculiari che la generazione autoregressiva introduce e che il capitolo 3 ha anticipato: costo della richiesta ignoto a priori e doppio regime prefill/decode.

La prima svolta sistemistica è stata la schedulazione a livello di iterazione. I sistemi di serving tradizionali componevano batch a livello di richiesta, costringendo le richieste brevi ad attendere il completamento delle più lunghe; Orca ha introdotto il *continuous batching*, in cui a ogni passo di generazione il batch può cedere le sequenze completate e accogliere nuove richieste, eliminando le attese strutturali e moltiplicando il throughput (Yu et al., 2022). La seconda ha riguardato la memoria. Nel decode il collo di bottiglia è il *KV cache*, lo stato d'attenzione che cresce con la lunghezza della sequenza e la cui allocazione contigua produce frammentazione e sovra-riserva: PagedAttention lo gestisce con la logica della memoria virtuale, in blocchi non contigui allocati a domanda e condivisibili tra richieste, portando lo spreco quasi a zero e guadagni di throughput tra due e quattro volte a parità di hardware (Kwon et al., 2023).

Il terzo passaggio nasce dal conflitto tra le due fasi della generazione. Prefill e decode competono per gli stessi acceleratori con profili opposti — il primo satura il calcolo, il secondo la banda di memoria — e la loro coabitazione nello stesso batch gonfia la coda della distribuzione di latenza. Le risposte si collocano lungo un continuum: Sarathi-Serve spezza il prefill in porzioni schedulate insieme al decode (*chunked prefill*), attenuando il compromesso tra throughput e tail latency sulla stessa flotta (Agrawal et al., 2024); Splitwise separa fisicamente le fasi su pool di macchine distinti, dimensionabili e persino equipaggiabili in modo indipendente (Patel et al., 2024); DistServe formalizza la disaggregazione in chiave di *goodput*, ottimizzando congiuntamente parallelismo e piazzamento di prefill e decode per massimizzare le richieste servite entro gli SLO di TTFT e TPOT (Zhong et al., 2024). Sul piano operativo, la variabilità della domanda e i tempi di caricamento dei modelli (i «cold start» del serving LLM si misurano in minuti, non in secondi) spingono verso politiche di autoscaling previsionale, oggetto di lavori recenti come SageServe (Jaiswal et al., 2025).

La Tabella 2 riepiloga i sistemi discussi e il contributo di ciascuno alle proprietà in esame.

Tabella 2 — *Sistemi di riferimento per l'inference serving di LLM: idea chiave e contributo a scalabilità e affidabilità del servizio.*

Sistema	Sede, anno	Idea chiave	Contributo principale
Orca (Yu et al.)	OSDI 2022	*Continuous batching* a livello di iterazione	Elimina le attese strutturali del batching a richiesta; throughput ↑
vLLM (Kwon et al.)	SOSP 2023	*PagedAttention*: KV cache paginata e condivisibile	Frammentazione di memoria quasi azzerata; throughput 2–4×

Sistema	Sede, anno	Idea chiave	Contributo principale
Sarathi-Serve (Agrawal et al.)	OSDI 2024	*Chunked prefill* e batching senza stalli	Attenua il trade-off throughput / tail latency
Splitwise (Patel et al.)	ISCA 2024	Separazione fisica delle fasi prefill e decode	Pool dimensionati (ed equipaggiati) per fase; efficienza dei costi
DistServe (Zhong et al.)	OSDI 2024	Disaggregazione orientata al *goodput*	Massimizza le richieste entro SLO di TTFT e TPOT

4.4 MLOps: il ciclo di vita del modello come problema di reliability engineering

Le sezioni precedenti hanno trattato il modello come un carico computazionale; la prospettiva MLOps lo tratta come un artefatto deperibile inserito in un processo. L'osservazione fondativa risale a Sculley et al. (2015): nei sistemi di machine learning il codice è la parte minore, e il debito tecnico si annida nelle dipendenze dai dati, nei confini erosi tra componenti e nei circuiti di retroazione tra predizioni e mondo, forme di fragilità che i controlli del software engineering tradizionale non intercettano. Un modello accurato al momento del rilascio degrada silenziosamente quando la distribuzione dei dati d'ingresso si allontana da quella di addestramento; il guasto, in questo dominio, raramente ha la cortesia di manifestarsi come un'eccezione.

La sistematizzazione più citata del paradigma è quella di Kreuzberger, Kühl e Hirschl (2023), che da revisione della letteratura, analisi degli strumenti e interviste a professionisti deriva principi, componenti e ruoli di un'architettura MLOps di riferimento: versionamento congiunto di codice, dati e modelli; pipeline di *continuous integration* e *continuous delivery* estese al *continuous training*; feature store, model registry e monitoraggio in esercizio come componenti di prima classe. Riletta con le categorie del capitolo 3, questa architettura è a tutti gli effetti ingegneria dell'affidabilità applicata a un nuovo tipo di failure mode: il monitoraggio del drift definisce gli SLI del modello, il retraining automatico ne è la remediation, il registry con versioni e rollback svolge per il comportamento del modello la funzione che il checkpoint svolge per il suo stato, riportando il sistema a un punto di ripristino noto quando la qualità delle predizioni viola l'obiettivo.

Questa saldatura tra ciclo di vita del modello e pratica operativa è anche il punto in cui le due direttrici del documento si toccano: il monitoraggio continuo che l'MLOps istituisce produce esattamente la telemetria (metriche, log, tracce, segnali di qualità) su cui operano i sistemi di gestione intelligente dell'infrastruttura. Il capitolo 5 attraversa il confine e li esamina.

5. Parte II — L'AI per la scalabilità e l'affidabilità dei sistemi

Il capitolo precedente ha percorso la direttrice che va dai sistemi all'AI; questo percorre quella inversa. Il termine *AIOps* — Artificial Intelligence for IT Operations — nasce in ambito industriale a metà del decennio scorso e designa oggi un campo di ricerca maturo: l'applicazione di tecniche di apprendimento automatico alla gestione operativa dei sistemi, lungo un ciclo di vita del guasto che la letteratura articola in quattro fasi: rilevazione, *triage*, analisi delle cause e mitigazione (Ahmed et al., 2023; Zhang et al., 2025). La Figura 3 rappresenta questa pipeline e le due generazioni tecnologiche che vi si sono succedute: i modelli specializzati della prima stagione e i large language model che ne stanno ridefinendo i confini, fino alle architetture agentiche che chiudono il ciclo con l'azione autonoma. Le tre sezioni che seguono ripercorrono questa traiettoria.

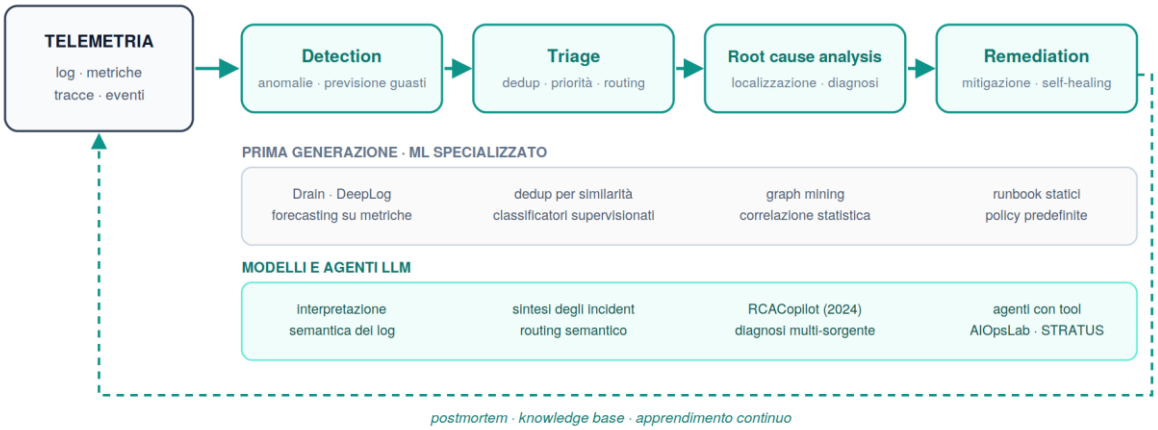

Figura 3 — *La pipeline del failure management (telemetria → detection → triage → root cause analysis → remediation) e le due generazioni di tecniche che la popolano: ML specializzato e modelli/agenti LLM. Il circuito di retroazione — postmortem e knowledge base — alimenta l'apprendimento continuo.*

5.1 AIOps: dall'anomaly detection statistica ai Large Language Model

La materia prima dell'AIOps è la telemetria: log semi-strutturati, serie temporali di metriche, tracce distribuite delle richieste, eventi di piattaforma. In un'infrastruttura di media complessità questi segnali si producono a ritmi che escludono ogni ispezione manuale sistematica, e la loro eterogeneità (testo libero accanto a numeri, formati che mutano a ogni rilascio, semantiche note solo a chi ha scritto il servizio) costituisce da sempre la difficoltà distintiva del dominio rispetto ad altri campi applicativi dell'apprendimento automatico.

La prima generazione di tecniche ha affrontato il problema scomponendolo in compiti specializzati. Il *log parsing* riconduce le righe di log ai loro template generatori, trasformando testo semi-strutturato in sequenze di eventi trattabili: l'algoritmo Drain, con il suo albero a

profondità fissa aggiornabile online, è divenuto il riferimento della categoria (He et al., 2017). Sulle sequenze così ottenute operano i rilevatori di anomalie: DeepLog ha mostrato che una rete ricorrente addestrata sul comportamento normale del sistema può segnalare deviazioni sia nell'ordine degli eventi sia nei valori dei parametri, inaugurando la stagione del deep learning applicato ai log (Du et al., 2017). Accanto a questi, rilevatori statistici e neurali per le serie temporali, classificatori per il triage degli alert, tecniche di mining su grafi di dipendenza per la localizzazione dei guasti. I risultati non sono mancati, ma nemmeno i limiti strutturali: ogni compito richiede il proprio modello, addestrato su dati etichettati costosi da produrre; i modelli sono fragili rispetto all'evoluzione dei formati di log; ciascun segnale viene analizzato nel proprio silo, mentre la diagnosi reale richiede di correlare log, metriche, tracce e contesto; e la proliferazione di rilevatori disgiunti alimenta quella *alert fatigue* che affligge i team di esercizio.

È su questi limiti che i large language model hanno inciso più in profondità. La survey di Zhang et al. (2025) su ACM Computing Surveys, che sistematizza 183 lavori pubblicati tra il gennaio 2020 e il dicembre 2024, documenta uno spostamento netto del baricentro della ricerca verso approcci LLM-based lungo tutte le fasi del ciclo di vita del guasto. Le ragioni tecniche sono riconducibili a poche proprietà: la comprensione semantica di testo semi-strutturato senza addestramento dedicato, che rende trattabili log e descrizioni di incident così come sono; la capacità di correlare sorgenti eterogenee (una riga di log, una soglia violata, un paragrafo di runbook) entro un unico spazio di rappresentazione; l'adattamento *few-shot* a servizi e formati nuovi, dove la prima generazione richiedeva ri-addestramento; e un'interfaccia in linguaggio naturale che abbatte il costo cognitivo di interrogare la telemetria. A queste proprietà corrispondono cautele altrettanto note, che la stessa letteratura registra: la tendenza dei modelli a produrre risposte plausibili ma infondate impone di ancorarne l'uso a dati recuperati e verificabili (knowledge base, incidenti storici, guide di troubleshooting), e i costi computazionali e di latenza dell'inferenza ne condizionano l'impiego sui flussi di telemetria ad alta frequenza. Come si vedrà nella sezione seguente, i sistemi che hanno raggiunto la produzione sono precisamente quelli che hanno preso sul serio entrambe le avvertenze.

5.2 Failure management assistito da LLM: dalla ricerca alla produzione

L'incident management dei grandi servizi cloud è il terreno di verifica naturale di queste tecniche, per ragioni insieme tecniche e organizzative: il volume degli incidenti eccede la capacità dei team on-call, il costo cognitivo della diagnosi è dominato dalla raccolta e dalla sintesi del contesto, e ogni minuto sottratto all'MTTR ha un valore economico misurabile. Non sorprende che i primi studi su larga scala provengano da chi quel problema lo vive quotidianamente.

Il lavoro di Ahmed et al. (2023), presentato a ICSE, costituisce il punto di partenza riconosciuto: il primo studio su larga scala sull'efficacia dei large language model nel raccomandare cause radice e passi di mitigazione per incidenti di produzione, condotto sul patrimonio di incident di Microsoft con modelli della famiglia GPT-3 sottoposti a fine-tuning. I risultati delineano il profilo d'uso che la letteratura successiva avrebbe confermato: i modelli non sostituiscono la diagnosi

dell'ingegnere, ma ne comprimono la fase iniziale, offrendo ipotesi di partenza e sintesi del contesto che riducono il tempo di orientamento.

RCACopilot, presentato a EuroSys l'anno successivo, mostra come questo profilo si traduca in un sistema di esercizio (Chen et al., 2024). L'architettura è istruttiva proprio perché non affida al modello un compito aperto: l'incidente in ingresso viene instradato verso l'handler corrispondente al proprio tipo di alert; una componente dedicata raccoglie e aggrega le informazioni diagnostiche di runtime dalle sorgenti pertinenti; solo a valle di questa preparazione il modello predice la categoria di causa radice e produce una narrazione esplicativa. Valutato su un anno di incidenti reali Microsoft, il sistema raggiunge un'accuratezza di analisi fino a 0,766, e la componente di raccolta diagnostica risulta in uso in produzione da oltre quattro anni. La lezione architetturale è generalizzabile: l'affidabilità dell'assistente LLM non discende dal modello in sé, ma dal grado in cui il flusso di lavoro lo vincola a operare su evidenze raccolte e verificabili: il *grounding* come proprietà strutturale della pipeline, prima ancora che come correttivo a posteriori.

Attorno a questi capisaldi si è consolidato un ecosistema di applicazioni che copre l'intero ciclo: interpretazione e sintesi dei log in linguaggio naturale, deduplicazione semantica e routing degli incident, generazione di ipotesi diagnostiche ancorate a knowledge base e troubleshooting guide, redazione assistita dei postmortem che a loro volta arricchiscono la base di conoscenza. È il circuito di retroazione della Figura 3. Il valore dimostrato a oggi si concentra dunque nelle fasi cognitive del failure management, dove l'errore del modello è recuperabile perché un essere umano resta nel ciclo decisionale. Il passo ulteriore, affidare all'AI anche l'azione correttiva, cambia la natura del problema ed è l'oggetto della sezione che segue.

5.3 Verso il cloud autonomo: agenti, self-healing e benchmark di valutazione

La chiusura del ciclo, dalla diagnosi alla *remediation* eseguita autonomamente, trasforma un problema di qualità delle risposte in un problema di sicurezza delle azioni. Un agente che riavvia un servizio, modifica una configurazione o scala una risorsa opera con un raggio d'impatto reale su sistemi in esercizio; l'errore non è più un suggerimento fuorviante ma un'azione da annullare, ammesso che sia annullabile. La ricerca recente ha risposto su due fronti complementari: costruire infrastrutture di valutazione realistiche e progettare architetture agentiche con garanzie di sicurezza formulate esplicitamente.

Sul primo fronte, AIOpsLab propone un framework olistico che orchestra l'intera interazione agente-cloud: ambienti a microservizi riproducibili, iniezione di guasti, generazione di carico e interfacce standardizzate attraverso cui l'agente osserva la telemetria e agisce, con metriche interpretabili sui compiti di rilevazione, localizzazione, diagnosi e mitigazione (Chen et al., 2025). ITBench, presentato a ICML, estende la valutazione oltre la site reliability: 102 scenari ispirati a casi reali nelle aree SRE, CISO e FinOps, eseguiti in ambienti Kubernetes operativi. I risultati iniziali sono istruttivi per la loro sobrietà: gli agenti basati sui migliori modelli disponibili

risolvono soltanto l'11,4% degli scenari SRE, il 25,2% di quelli CISO e il 25,8% di quelli FinOps (Jha et al., 2025). La distanza tra la fluidità dell'assistenza conversazionale e l'affidabilità richiesta dall'azione autonoma su sistemi reali è, allo stato, ampia e misurabile.

Sul secondo fronte, STRATUS, presentato a NeurIPS, organizza agenti specializzati per rilevazione, diagnosi e mitigazione entro una macchina a stati che rende possibile il ragionamento di sicurezza a livello di sistema, e formalizza una specifica denominata *Transactional No-Regression*: l'esplorazione delle azioni correttive è ammessa solo in forma transazionale, con la garanzia che lo stato del sistema non peggiori rispetto al punto di partenza. L'esito sperimentale è significativo anche in chiave metodologica: la disciplina di sicurezza non penalizza l'efficacia ma la abilita, con tassi di successo nella mitigazione dei guasti superiori di almeno 1,5 volte rispetto agli agenti allo stato dell'arte su AIOpsLab e ITBench (Chen et al., 2025b). È la traduzione agentica di un principio antico dell'ingegneria dell'affidabilità: la reversibilità delle azioni come precondizione dell'automazione.

Il quadro che emerge suggerisce una traiettoria di adozione graduale (autonomia crescente per classi di azioni a raggio d'impatto crescente, privilegi minimi, tracciabilità completa delle decisioni) più che un salto verso il cloud che si amministra da sé. E consegna al capitolo successivo la sua domanda di apertura: un sistema di gestione autonoma dotato di privilegi operativi è anche, inevitabilmente, una superficie d'attacco, e la telemetria stessa che esso legge (log, ticket, annotazioni) è un canale attraverso cui un avversario può tentare di manipolarne il comportamento.

6. Reliability incontra security: la prospettiva della resilienza

I capitoli precedenti hanno trattato il guasto come evento accidentale: componenti che si rompono, bit che si corrompono, distribuzioni di dati che derivano. Questo capitolo rimuove quell'assunzione e considera il guasto provocato deliberatamente. Non si tratta di un'aggiunta estranea al quadro costruito finora: la tassonomia fondativa della dependability classifica i fault anche in base all'intento, includendo esplicitamente i guasti di origine umana e dolosa, e riconosce che dependability e security condividono gli attributi di availability e integrity, cui la seconda aggiunge la confidentiality (Avizienis et al., 2004). Ciò che l'era dell'AI introduce di nuovo è duplice: una classe inedita di superfici d'attacco (il modello stesso, i suoi dati, il suo contesto) e una categoria altrettanto nuova di bersagli di alto valore, gli operatori AI dotati di privilegi sull'infrastruttura descritti nel capitolo 5. La sezione 6.1 organizza il threat model sulla tassonomia NIST di riferimento; la 6.2 argomenta la tesi centrale del capitolo, la continuità profonda tra fault tolerance e adversarial robustness; la 6.3 ne trae le conseguenze su osservabilità, audit e prontezza forense.

6.1 Il threat model dei sistemi AI: la tassonomia NIST

Il riferimento consolidato per il threat modeling dei sistemi di machine learning è il rapporto NIST AI 100-2, la cui edizione 2025 (Vassilev et al., 2025) organizza l'*adversarial machine learning* in una gerarchia a cinque dimensioni: il tipo di sistema colpito, distinguendo tra AI predittiva (PredAI) e generativa (GenAI); la fase del ciclo di vita in cui l'attacco avviene, dall'addestramento al deployment; gli obiettivi dell'attaccante, espressi come proprietà violate (availability, integrity, privacy, cui la sezione GenAI aggiunge l'abilitazione al *misuse*); le sue capacità di accesso; la sua conoscenza del sistema. Rispetto all'edizione 2023, incentrata sulla triade classica evasion-poisoning-privacy per i sistemi predittivi, l'aggiornamento estende sistematicamente la copertura alla AI generativa: attacchi alla supply chain, *prompt injection* diretta e indiretta, rischi specifici dei sistemi agentici.

Nella fase di addestramento la minaccia principale è il *data poisoning*: la manipolazione di una frazione dei dati per degradare il modello, orientarne il comportamento o installarvi una *backdoor* attivabile da un trigger, anche nella variante *clean-label* che non altera le etichette e risulta perciò più difficile da individuare. A lungo considerato un rischio teorico per i modelli addestrati su corpora web-scale, il poisoning è stato dimostrato immediatamente praticabile da Carlini et al. (2024): l'attacco *split-view* sfrutta la natura mutevole dei contenuti web (inclusa la possibilità di acquisire domini scaduti che ospitano frammenti dei dataset distribuiti) per far sì che ciò che il curatore ha validato differisca da ciò che i client scaricheranno; l'attacco *frontrunning* colpisce invece gli snapshot periodici di contenuti crowd-sourced come Wikipedia, iniettando modifiche calibrate sull'istante di cattura. Gli autori mostrano che dieci dataset di uso comune erano avvelenabili con queste tecniche: la conseguenza pratica è che ogni corpus raccolto dal web va assunto come potenzialmente contaminato, e che provenance e validazione dei dati,

componenti della pipeline MLOps del capitolo 4, sono controlli di sicurezza prima ancora che di qualità.

Considerazioni analoghe valgono per la supply chain del modello: il riuso di foundation model pre-addestrati, checkpoint e dipendenze di terze parti trasferisce la fiducia dall'organizzazione a una filiera opaca, in cui un *model poisoning* a monte si propaga a ogni sistema derivato. Nella fase di esercizio, i sistemi predittivi restano esposti agli attacchi di *evasion* — gli adversarial example che inducono classificazioni errate con perturbazioni impercettibili — e agli attacchi alla privacy, dalla *membership inference*, che stabilisce se un dato ha fatto parte dell'addestramento, alla *model extraction* e alla ricostruzione dei dati di training attraverso le API di inferenza.

Per i sistemi generativi la tassonomia registra il vettore oggi più discusso: la prompt injection. Nella forma diretta è l'utente stesso a formulare input che scavalcano le istruzioni di sistema; nella forma indiretta, dimostrata da Greshake et al. (2023) sulle applicazioni LLM-integrate del mondo reale, le istruzioni ostili sono collocate nei contenuti che il modello recupera ed elabora (una pagina web, un documento, un'email), trasformando ogni sorgente di contesto in un potenziale canale di comando. È qui che il threat model incontra la Parte II di questo documento: un agente AIOps legge log, ticket, alert e annotazioni che, per costruzione, includono contenuto influenzabile da terzi; righe di log confezionate ad arte o descrizioni di incident manipolate sono, per un operatore LLM privilegiato, l'equivalente funzionale di un input di comando non fidato. La Tabella 3 riepiloga le classi di attacco e le proietta sulle due direttrici del documento.

Tabella 3 — *Classi di attacco della tassonomia NIST AI 100-2 E2025, fase del ciclo di vita interessata, proprietà violate e superficie rilevante per le due direttrici del documento.*

Classe di attacco (NIST AI 100-2)	Fase del ciclo di vita	Proprietà violata	Superficie nelle due direttrici
Data poisoning (incl. backdoor e clean-label)	Raccolta dati e training	Integrity (availability, se mirato alla degradazione)	Parte I: pipeline dati, continuous training MLOps
Model poisoning e supply chain	Sviluppo e distribuzione	Integrity	Parte I: riuso di foundation model, model registry, dipendenze
Evasion (adversarial examples)	Inference	Integrity	Parte I: serving di sistemi predittivi in produzione
Attacchi alla privacy (membership inference, model extraction, ricostruzione)	Inference	Privacy / confidentiality	Parte I: serving multi-tenant, API di inferenza pubbliche
Direct prompt injection	Inference (input dell'utente)	Integrity, misuse	Applicazioni GenAI esposte all'utente
Indirect prompt injection	Inference (contenuti recuperati: web, documenti, telemetria)	Integrity, misuse	Parte II: agenti AIOps che elaborano log, ticket e alert

Classe di attacco (NIST AI 100-2)	Fase del ciclo di vita	Proprietà violata	Superficie nelle due direttrici
Availability breakdown (poisoning degradante, input a costo massimizzato)	Training e inference	Availability	Parte I: SLO e goodput del serving, costi di esercizio

6.2 Fault tolerance e adversarial robustness come proprietà congiunte

La tesi di questo capitolo è che reliability e security dei sistemi AI siano le due facce, accidentale e intenzionale, di un medesimo spazio dei guasti, e che trattarle come discipline separate produca architetture ridondanti dove non serve e scoperte dove serve. Il fondamento concettuale, come anticipato, è già nella tassonomia della dependability: ciò che distingue un fault doloso da uno accidentale non è l'effetto sul sistema (una violazione di integrity resta tale) ma l'origine e, soprattutto, il modello statistico del suo verificarsi (Avizienis et al., 2004).

Il training distribuito offre l'illustrazione più nitida di questa unità. Dal punto di vista dell'algoritmo di aggregazione dei gradienti, un worker che trasmette un contributo corrotto è un unico oggetto matematico, quale che ne sia la causa: un guasto di memoria, una corruzione silente del calcolo o un partecipante ostile che tenta di avvelenare il modello. La letteratura sulla *Byzantine fault tolerance* applicata all'apprendimento distribuito ha formalizzato esattamente questa equivalenza: regole di aggregazione come Krum garantiscono la convergenza in presenza di una frazione limitata di worker bizantini, senza bisogno di distinguere se il comportamento arbitrario sia frutto di malfunzionamento o di malizia (Blanchard et al., 2017). Sul versante hardware, la scoperta dei *mercurial core* (processori che producono silenziosamente risultati errati, documentati su larga scala tanto da Google quanto da Meta) ha mostrato che violazioni di integrity computazionale prive di qualunque avversario sono una realtà statistica delle flotte moderne (Hochschild et al., 2021; Dixit et al., 2021); le contromisure, dalla verifica ridondante del calcolo allo screening periodico dei componenti, appartengono allo stesso repertorio della verifica di integrità che la security impiega contro la manomissione deliberata.

L'unità dello spazio dei guasti non implica però l'identità dei modelli di minaccia, e la progettazione deve esercitare la distinzione proprio su questo piano. I fault accidentali sono, in prima approssimazione, indipendenti e casuali: contro di essi la ridondanza statistica (repliche, maggioranze, checksum) è efficace e quantificabile. Il fault avversario è invece adattivo e si colloca deliberatamente nel caso peggiore: conosce le difese, ne cerca i bordi, colpisce dove la ridondanza è cieca. Una robustezza dimostrata contro il rumore non dice nulla sulla robustezza contro una perturbazione ottimizzata; i limiti teorici della tolleranza bizantina valgono solo entro la frazione di partecipanti compromessi per cui sono stati derivati. La conseguenza progettuale è una condivisione di strumenti accompagnata da una divaricazione di valutazioni: gli stessi meccanismi di rilevazione, isolamento e ripristino servono entrambi i domini, ma la loro verifica richiede due discipline di fault injection distinte. Il *chaos engineering* inietta guasti secondo il

modello accidentale; il *red teaming* li inietta secondo quello avversario. L'infrastruttura di valutazione degli agenti autonomi vista nel capitolo 5 evolve del resto nella stessa direzione, affiancando agli scenari di guasto quelli di compliance e sicurezza. La Figura 4 sintetizza la matrice.

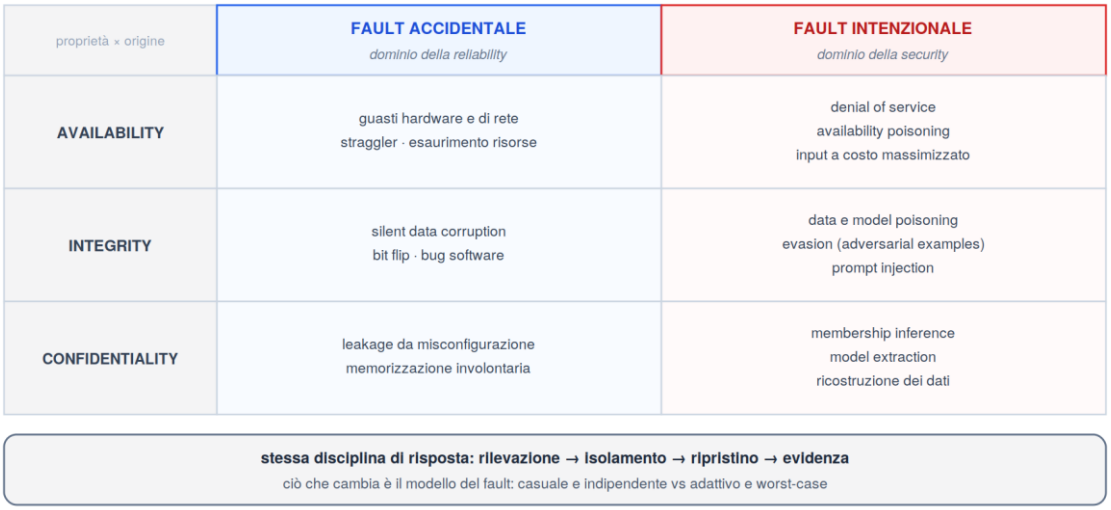

Figura 4 — *La matrice reliability × security: le tre proprietà fondamentali (availability, integrity, confidentiality) incrociate con l'origine del fault, accidentale o intenzionale. La disciplina di risposta — rilevazione, isolamento, ripristino, evidenza — è condivisa; ciò che cambia è il modello del fault: casuale e indipendente da un lato, adattivo e worst-case dall'altro.*

6.3 Osservabilità, audit trail e forensic readiness dei sistemi AI

Se la disciplina di risposta è condivisa, il suo ultimo anello, la capacità di ricostruire l'accaduto e produrne evidenza, merita una trattazione propria, perché è quello che i sistemi AI mettono più duramente alla prova. Un sistema la cui uscita dipende da un modello non deterministico, da un contesto recuperato dinamicamente e da una versione tra le molte in circolazione non è ricostruibile a posteriori se l'osservabilità non è stata progettata per questo scopo: la registrazione deve catturare, oltre agli eventi di piattaforma, gli input, il contesto effettivamente fornito al modello, l'identità e la versione del modello stesso e i parametri di generazione. È il minimo indispensabile perché un comportamento anomalo possa essere spiegato, riprodotto e attribuito.

Per i sistemi agentici della Parte II l'esigenza si fa stringente: ogni azione sull'infrastruttura deve essere riconducibile alla catena di osservazioni e decisioni che l'ha prodotta. Architetture come la macchina a stati di STRATUS, incontrata nel capitolo 5, mostrano che l'auditabilità può essere costruita nel progetto anziché aggiunta come ripensamento: stati e transizioni espliciti sono, di fatto, un registro delle decisioni. Il threat model della sezione 6.1 aggiunge però un'avvertenza che la pratica forense conosce bene: se la telemetria è un canale d'attacco, i log sono al tempo

stesso evidenza e bersaglio, e la loro integrità (conservazione non riscrivibile, firma, catena di custodia) è precondizione del loro valore probatorio.

Il quadro concettuale adeguato è quello della *forensic readiness*, la predisposizione dell'organizzazione a raccogliere e conservare evidenza digitale utilizzabile prima che l'incidente si verifichi, massimizzando il valore probatorio e minimizzando il costo della raccolta (Rowlingson, 2004). Declinata sui sistemi AI, essa investe l'intero ciclo di vita costruito nei capitoli precedenti: la provenance dei dati di addestramento e la firma dei checkpoint trasformano la pipeline MLOps in una catena di custodia degli artefatti; il model registry con versioni e rollback fornisce lo stato di riferimento rispetto a cui misurare la deviazione; la conservazione di prompt, contesti e decisioni degli agenti rende investigabile ciò che altrimenti resterebbe una scatola nera operativa. La direzione è peraltro convergente con quella normativa: il regolamento europeo sull'intelligenza artificiale impone ai sistemi ad alto rischio la registrazione automatica degli eventi lungo il ciclo di vita (Regolamento (UE) 2024/1689, art. 12), saldando l'obbligo giuridico alla buona pratica di ingegneria che questo capitolo ha delineato.

La resilienza, in questa prospettiva, è la proprietà che unifica il documento: un sistema resiliente regge il guasto, accidentale od ostile, si ripristina rapidamente e conserva dell'accaduto un'evidenza verificabile da cui apprendere. Il capitolo conclusivo dell'analisi esamina ciò che ancora manca perché questa proprietà sia raggiungibile alla scala e all'autonomia verso cui i sistemi si stanno muovendo.

7. Sfide aperte e direzioni di ricerca

I capitoli precedenti hanno descritto una frontiera in rapido movimento: recovery misurati in minuti dove si misuravano in ore, agenti che chiudono il ciclo del failure management, un threat model che si consolida in tassonomie di riferimento. Tre tensioni, tuttavia, restano strutturalmente aperte, e definiscono l'agenda di ricerca dei prossimi anni: il denominatore energetico della scala, l'osservabilità e la valutazione di sistemi che non si comportano due volte allo stesso modo, e il paradosso di un'autonomia che deve essere a sua volta garantita.

7.1 Costi energetici e sostenibilità della scala

La scala ha un denominatore fisico che nessuna ottimizzazione algoritmica può eludere. Secondo l'Agenzia Internazionale dell'Energia, i data center hanno consumato nel 2024 circa 415 TWh di elettricità, l'1,5% del totale mondiale, e la proiezione di base al 2030 supera i 945 TWh (più dell'intero consumo elettrico attuale del Giappone), con l'AI come principale motore della crescita e una domanda dei data center AI-oriented destinata a più che quadruplicare (IEA, 2025). Lo stesso rapporto segnala un tratto rilevante per l'analisi di questo documento: i data center dedicati all'AI assorbono potenze paragonabili a quelle di impianti industriali energivori come le fonderie di alluminio, ma con una concentrazione geografica molto maggiore, che li rende attori tanto critici quanto vulnerabili dei sistemi elettrici locali.

Il dibattito scientifico sull'impronta dell'AI si è mosso tra due poli. Da un lato la denuncia del costo, aperta dallo studio di Strubell, Ganesh e McCallum (2019) sull'impronta energetica dell'addestramento dei modelli NLP; dall'altro la risposta ingegneristica di Patterson et al. (2021), che mostra come scelte congiunte di architettura del modello, hardware specializzato ed efficienza e localizzazione dei data center possano ridurre l'impronta di addestramenti equivalenti di ordini di grandezza. L'analisi di Wu et al. (2022), condotta dal punto di vista di chi opera flotte di produzione, allarga opportunamente il perimetro: la contabilità centrata sul solo training trascura tanto il peso dell'inference, che si esercita in continuo su tutta la vita del modello, quanto il carbonio incorporato nella fabbricazione dell'hardware.

Ai fini di questo documento conta soprattutto la convergenza tra sostenibilità e affidabilità. Ogni ora di lavoro perduta per un guasto è energia consumata senza produrre avanzamento: l'ETTR del capitolo 4 ha una lettura energetica immediata, poiché la differenza tra il 90% e il 97% di tempo utile su un job trimestrale a migliaia di GPU è misurabile in gigawattora. Analogamente, la Model FLOPs Utilization è al tempo stesso una metrica di efficienza economica ed energetica, e il goodput per joule si candida a indicatore sintetico dell'esercizio dell'inference. In direzione opposta, l'energia entra nel perimetro dell'affidabilità come dipendenza critica: alimentazione e raffreddamento sono failure domain a pieno titolo, e la concentrazione geografica segnalata dalla IEA trasforma la resilienza della rete elettrica locale in una componente della resilienza del sistema AI. La ricerca sistemistica sta appena iniziando a trattare il vincolo energetico come un

vincolo primario della progettazione: nella schedulazione, nel dimensionamento, nella stessa definizione degli SLO.

7.2 Observability ed evaluation di sistemi non deterministici

Il capitolo 6 ha trattato l'osservabilità come requisito forense; qui la si considera come requisito ingegneristico quotidiano, e il problema si rivela se possibile più arduo. L'ingegneria dell'affidabilità presuppone la capacità di rispondere alla domanda «il sistema sta funzionando?» con una misura; per un sistema la cui uscita è testo aperto, dipendente dal contesto e dal campionamento, la definizione stessa di funzionamento corretto è sfuggente. Gli SLI classici (latenza, tasso d'errore, disponibilità) restano necessari ma cessano di essere sufficienti: la qualità della risposta diventa una dimensione di servizio da misurare, e le pratiche disponibili (valutazione a campione, batterie di regression test comportamentali, impiego di modelli come giudici con i bias che ne conseguono) sono ancora lontane dalla maturità metrologica delle controparti infrastrutturali.

Il bersaglio, inoltre, si muove. Chen, Zaharia e Zou (2024) hanno documentato variazioni sostanziali nel comportamento di servizi LLM commerciali a distanza di pochi mesi, sugli stessi compiti e a parità di versione dichiarata: per chi costruisce sistemi sopra API di terze parti, ogni valutazione ha una data di scadenza implicita, e il regression testing comportamentale diventa una necessità operativa più che una buona pratica. Anche entro il perimetro controllato dalla singola organizzazione, la riproducibilità esatta dell'inferenza è ostacolata da fattori di sistema (la composizione dei batch, l'ordine non associativo delle riduzioni in virgola mobile su hardware parallelo) che rendono il determinismo un obiettivo di progetto, non una proprietà gratuita.

A complicare il quadro c'è la natura intrecciata dei sistemi ML, che Sculley et al. (2015) sintetizzarono nel principio *changing anything changes everything*: la validazione per componenti, pilastro dell'ingegneria del software, perde efficacia dove il comportamento emerge dall'interazione tra modello, dati, contesto e strumenti. È la ragione per cui l'infrastruttura di valutazione end-to-end — gli ambienti riproducibili con iniezione di guasti e scenari realistici incontrati nel capitolo 5 — va considerata il collo di bottiglia della fiducia, non un accessorio accademico: la velocità con cui sapremo costruire valutazioni rappresentative determinerà la velocità con cui potremo delegare responsabilità.

7.3 Il paradosso dell'autonomia: chi garantisce l'AI che gestisce i sistemi?

L'ultima tensione è la più profonda, e la letteratura sui fattori umani l'aveva formulata ben prima dell'AI generativa. Le «ironie dell'automazione» descritte da Bainbridge (1983) calzano all'AIOps autonomo con quarant'anni di anticipo: automatizzare i compiti trattabili lascia all'operatore umano proprio i casi che l'automazione non sa gestire, i più rari e i più gravi, mentre la pratica quotidiana che manteneva vive le sue competenze si assottiglia. Un team che delega all'agente il novanta per cento degli incidenti di routine interverrà sul restante dieci con meno esercizio, meno

contesto e maggiore propensione ad accettare acriticamente le conclusioni del sistema: il fenomeno dell'*automation bias* documentato da Parasuraman e Riley (1997). Il disegno delle interfacce uomo-agente, perché preservino competenza e giudizio anziché ridurli a ratifica passiva, è una direzione di ricerca tanto necessaria quanto trascurata.

La seconda faccia del paradosso è ricorsiva: l'AI che gestisce i sistemi è essa stessa un sistema da gestire. Il suo monitoraggio, la sua valutazione, la sua sicurezza riaprono a un livello superiore esattamente i problemi che essa dovrebbe risolvere, e la ricorsione deve terminare in garanzie verificabili da esseri umani. I materiali per costruirle stanno emergendo dai capitoli precedenti: specifiche di sicurezza formali e verificabili come la Transactional No-Regression di STRATUS, che vincola l'esplorazione delle azioni alla reversibilità; autonomia graduata per classi di azioni a raggio d'impatto crescente; auditabilità strutturale delle decisioni; robustezza alla manipolazione della telemetria delineata nel capitolo 6. I numeri di ITBench (poco più di un incidente SRE su dieci risolto dagli agenti allo stato dell'arte; Jha et al., 2025) ricordano quanto sia ampio il divario tra questa architettura di garanzie e la sua realizzazione, e suggeriscono che la domanda giusta non sia quando l'infrastruttura si amministrerà da sé, ma quali classi di decisioni, con quali garanzie e quale supervisione, convenga delegare per prime. Su questa domanda, che è insieme tecnica, organizzativa e normativa, si chiude l'analisi e si aprono le conclusioni.

8. Conclusioni

Il percorso di questo documento si chiude dove era cominciato: sulla circolarità rappresentata nella Figura 1. L'AI in produzione dipende da infrastrutture la cui gestione è, a sua volta, sempre più delegata all'AI; la traversata delle due direttrici ha mostrato che questa circolarità è un fatto ingegneristico, non una figura retorica, con conseguenze misurabili su come i sistemi vengono progettati, esercitati, attaccati e difesi.

Cinque tesi emergono consolidate. La prima è che, alla scala dell'AI contemporanea, il guasto è una condizione operativa permanente: la sincronia del training rovescia il failure domain e rende l'affidabilità il presupposto stesso della scala. La seconda riguarda la frontiera competitiva, spostatasi dalla prevenzione alla velocità di ripristino: la distanza tra il 90% e il 97% di tempo utile di addestramento, percorsa in poco più di un anno di letteratura, vale gigawattora e mesi di calendario. La terza viene dall'AI applicata alle operations: i sistemi che funzionano in produzione sono quelli in cui il flusso di lavoro incorpora il *grounding* su evidenze verificabili fin dal progetto. La quarta: reliability e security condividono un unico spazio dei guasti, accidentale l'una e intenzionale l'altra, con strumenti comuni e modelli di minaccia distinti: chaos engineering e red teaming come due discipline della stessa fault injection. L'ultima è che l'autonomia operativa va costruita per gradi, vincolata a garanzie verificabili (reversibilità delle azioni, privilegi minimi, auditabilità delle decisioni) e valutata su benchmark realistici prima che le venga delegata responsabilità.

Per il decisore, queste tesi si traducono in domande da porre alla propria organizzazione: se il tempo utile dei workload AI e il goodput del serving siano misurati e presidiati come lo sono disponibilità e latenza dei servizi tradizionali; se la pipeline dei dati e dei modelli sia trattata come una supply chain di sicurezza, con provenance e firma degli artefatti; se le azioni degli strumenti automatici sull'infrastruttura siano ricostruibili a posteriori con valore probatorio; e con quali evidenze di valutazione si stia decidendo cosa delegare all'AI. Per il lettore tecnico, i sistemi discussi (da MegaScale a vLLM, da RCACopilot a STRATUS) costituiscono una mappa ragionata della letteratura primaria da cui partire.

L'era dell'artificial intelligence non ha reso obsoleta l'ingegneria dell'affidabilità: l'ha promossa a disciplina fondativa, estendendone il mandato alla sicurezza e alla produzione di evidenza. Costruire sistemi scalabili e affidabili significa oggi costruire sistemi resilienti: capaci di resistere al guasto accidentale come a quello ostile, di ripristinarsi in tempi commisurati al valore in gioco e di apprendere dall'accaduto lasciandone traccia verificabile. È il criterio con cui misurare tanto le infrastrutture che servono l'AI quanto l'AI che serve le infrastrutture.

9. Riferimenti bibliografici

Agrawal, A., Kedia, N., Panwar, A., et al. (2024). *Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve*. In Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI '24).

Ahmed, T., Ghosh, S., Bansal, C., Zimmermann, T., Zhang, X., Rajmohan, S. (2023). *Recommending Root-Cause and Mitigation Steps for Cloud Incidents using Large Language Models*. In Proceedings of the 45th International Conference on Software Engineering (ICSE '23), pp. 1737–1749. DOI: 10.1109/ICSE48619.2023.00149.

Amdahl, G. M. (1967). *Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities*. In AFIPS Spring Joint Computer Conference, pp. 483–485.

Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C. (2004). *Basic Concepts and Taxonomy of Dependable and Secure Computing*. IEEE Transactions on Dependable and Secure Computing, vol. 1, n. 1, pp. 11–33. DOI: 10.1109/TDSC.2004.2.

Bainbridge, L. (1983). *Ironies of Automation*. Automatica, vol. 19, n. 6, pp. 775–779.

Beyer, B., Jones, C., Petoff, J., Murphy, N. R. (a cura di) (2016). *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media.

Blanchard, P., El Mhamdi, E. M., Guerraoui, R., Stainer, J. (2017). *Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent*. In Advances in Neural Information Processing Systems 30 (NeurIPS 2017).

Carlini, N., Jagielski, M., Choquette-Choo, C. A., Paleka, D., Pearce, W., Anderson, H., Terzis, A., Thomas, K., Tramèr, F. (2024). *Poisoning Web-Scale Training Datasets is Practical*. In 2024 IEEE Symposium on Security and Privacy (SP), pp. 407–425.

Chen, L., Zaharia, M., Zou, J. (2024). *How Is ChatGPT's Behavior Changing over Time?*. Harvard Data Science Review, vol. 6, n. 2.

Chen, Y., Xie, H., Ma, M., et al. (2024). *Automatic Root Cause Analysis via Large Language Models for Cloud Incidents*. In Proceedings of the 19th European Conference on Computer Systems (EuroSys '24), pp. 674–688. DOI: 10.1145/3627703.3629553.

Chen, Y., Shetty, M., Somashekar, G., Ma, M., Simmhan, Y., Mace, J., Bansal, C., Wang, R., Rajmohan, S. (2025). *AIOpsLab: A Holistic Framework to Evaluate AI Agents for Enabling Autonomous Clouds*. In Proceedings of Machine Learning and Systems (MLSys 2025).

Chen, Y., Pan, J., Clark, J., Su, Y., Zheutlin, N., Bhavya, B., Arora, R., Deng, Y., Jha, S., Xu, T. (2025b). *STRATUS: A Multi-agent System for Autonomous Reliability Engineering of Modern Clouds*. In Advances in Neural Information Processing Systems (NeurIPS 2025). arXiv:2506.02009.

Dean, J., Barroso, L. A. (2013). *The Tail at Scale*. Communications of the ACM, vol. 56, n. 2, pp. 74–80. DOI: 10.1145/2408776.2408794.

Dixit, H. D., Pendharkar, S., Beadon, M., et al. (2021). *Silent Data Corruptions at Scale*. arXiv:2102.11245.

Du, M., Li, F., Zheng, G., Srikumar, V. (2017). *DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning*. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS '17).

Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., Fritz, M. (2023). *Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection*. In Proceedings of the ACM Workshop on Artificial Intelligence and Security (AISec '23).

Gustafson, J. L. (1988). *Reevaluating Amdahl's Law*. Communications of the ACM, vol. 31, n. 5, pp. 532–533.

He, P., Zhu, J., Zheng, Z., Lyu, M. R. (2017). *Drain: An Online Log Parsing Approach with Fixed Depth Tree*. In Proceedings of the IEEE International Conference on Web Services (ICWS '17).

Hochschild, P. H., Turner, P., Mogul, J. C., et al. (2021). *Cores That Don't Count*. In Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS '21).

IEA (2025). *Energy and AI*. International Energy Agency, Parigi.

Jaiswal, S., Jain, K., et al. (2025). *SageServe: Optimizing LLM Serving on Cloud Data Centers with Forecast Aware Auto-Scaling*. Proceedings of the ACM on Measurement and Analysis of Computing Systems. arXiv:2502.14617.

Jha, S., et al. (2025). *ITBench: Evaluating AI Agents across Diverse Real-World IT Automation Tasks*. In Proceedings of the 42nd International Conference on Machine Learning (ICML '25), PMLR 267, pp. 27134–27197.

Jiang, Z., Lin, H., Zhong, Y., et al. (2024). *MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs*. In Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI '24), pp. 745–760.

Kokolis, A., Kuchnik, M., Hoffman, J., et al. (2024). *Revisiting Reliability in Large-Scale Machine Learning Research Clusters*. arXiv:2410.21680.

Kreuzberger, D., Kühl, N., Hirschl, S. (2023). *Machine Learning Operations (MLOps): Overview, Definition, and Architecture*. IEEE Access, vol. 11, pp. 31866–31879. DOI: 10.1109/ACCESS.2023.3262138.

Kwon, W., Li, Z., Zhuang, S., et al. (2023). *Efficient Memory Management for Large Language Model Serving with PagedAttention*. In Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP '23), pp. 611–626. DOI: 10.1145/3600006.3613165.

Llama Team, AI @ Meta (2024). *The Llama 3 Herd of Models*. arXiv:2407.21783.

Narayanan, D., Shoeybi, M., Casper, J., et al. (2021). *Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM*. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21).

Parasuraman, R., Riley, V. (1997). *Humans and Automation: Use, Misuse, Disuse, Abuse*. Human Factors, vol. 39, n. 2, pp. 230–253.

Patel, P., Choukse, E., Zhang, C., et al. (2024). *Splitwise: Efficient Generative LLM Inference Using Phase Splitting*. In Proceedings of the ACM/IEEE International Symposium on Computer Architecture (ISCA '24).

Patterson, D., Gonzalez, J., Le, Q., et al. (2021). *Carbon Emissions and Large Neural Network Training*. arXiv:2104.10350.

Rajbhandari, S., Rasley, J., Ruwase, O., He, Y. (2020). *ZeRO: Memory Optimizations Toward Training Trillion Parameter Models*. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '20).

Regolamento (UE) 2024/1689 del Parlamento europeo e del Consiglio, del 13 giugno 2024, che stabilisce regole armonizzate sull'intelligenza artificiale (regolamento sull'intelligenza artificiale).

Rowlingson, R. (2004). *A Ten Step Process for Forensic Readiness*. International Journal of Digital Evidence, vol. 2, n. 3.

Sculley, D., Holt, G., Golovin, D., et al. (2015). *Hidden Technical Debt in Machine Learning Systems*. In Advances in Neural Information Processing Systems 28 (NeurIPS 2015).

Strubell, E., Ganesh, A., McCallum, A. (2019). *Energy and Policy Considerations for Deep Learning in NLP*. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL 2019).

Vassilev, A., Oprea, A., Fordyce, A., Anderson, H., et al. (2025). *Adversarial Machine Learning: A Taxonomy and Terminology of Attacks and Mitigations*. NIST AI 100-2 E2025, National Institute of Standards and Technology. DOI: 10.6028/NIST.AI.100-2e2025.

Wan, B., Han, M., Sheng, Y., et al. (2024). *ByteCheckpoint: A Unified Checkpointing System for Large Foundation Model Development*. arXiv:2407.20143.

Wan, B., Liu, G., Song, Z., et al. (2025). *Robust LLM Training Infrastructure at ByteDance*. In Proceedings of the ACM Symposium on Operating Systems Principles (SOSP '25). DOI: 10.1145/3731569.3764838.

Wang, Z., Jia, Z., Zheng, S., et al. (2023). *Gemini: Fast Failure Recovery in Distributed Training with In-Memory Checkpoints*. In Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP '23), pp. 364–381.

Wu, C.-J., Raghavendra, R., Gupta, U., et al. (2022). *Sustainable AI: Environmental Implications, Challenges and Opportunities*. In Proceedings of Machine Learning and Systems (MLSys 2022).

Yu, G.-I., Jeong, J. S., Kim, G.-W., Kim, S., Chun, B.-G. (2022). *Orca: A Distributed Serving System for Transformer-Based Generative Models*. In Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22), pp. 521–538.

Zhang, L., Jia, T., Jia, M., et al. (2025). *A Survey of AIOps in the Era of Large Language Models*. ACM Computing Surveys. DOI: 10.1145/3746635.

Zhong, Y., Liu, S., Chen, J., et al. (2024). *DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving*. In Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI '24).