Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica per il Management

Piattaforma Web per l'Aggregazione e l'Analisi Personalizzata di Dati Agricoli

Relatore:
Chiar.mo Prof.
MARCO DI FELICE

Presentata da:
DAVIDE BATTILANI

Correlatore:
Dott.
IVAN ZYRIANOFF

SESSIONE 3
Anno Accademico 2024/2025

Indice

Abstract **3**

1 Introduzione **4**

2 Stato dell'Arte **6**
2.1 IoT per il Monitoraggio Agricolo 6
2.2 Piattaforme di Visualizzazione e Accesso ai Dati 8
2.3 Large Language Models per l'Analisi di Dati Agricoli 10
2.4 InfluxDB: Architettura e Principi di Funzionamento 14
2.4.1 Modello dei dati e storage engine 14
2.4.2 Flux e gestione del ciclo di vita dei dati 15
2.5 Gap Identificati 16

3 Il Progetto TRACE **17**
3.1 Agricoltura di Precisione e Settore MAPs 17
3.2 Architettura del Progetto TRACE 18
3.3 Implementazione del Progetto TRACE 21

4 Progettazione **24**
4.1 Panoramica Generale 24
4.1.1 Presentation Layer 26
4.1.2 Business Logic Layer 28
4.1.3 Data Layer 30
4.1.4 Integrazione servizi AI 31
4.2 Workflow della Piattaforma 33
4.2.1 Autenticazione 33
4.2.2 Costruzione delle Query 36
4.2.3 Esportazione dei Dati 38
4.2.4 Generazione di Report Analitici 39

5 Implementazione **42**
5.1 Backend: NestJS e Architettura Modulare 42
5.1.1 Struttura modulare e Dependency Injection 43
5.1.2 Autenticazione e autorizzazione 45
5.1.3 Integrazione InfluxDB e generazione query Flux 46
5.1.4 Pipeline di Data Cleaning 48
5.1.5 Sistema di export asincrono 49
5.1.6 Integrazione multi-provider LLM 49
5.2 Frontend: Angular e Material Design 51
5.2.1 Architettura standalone components e routing 51
5.2.2 QueryFormComponent e Reactive Forms 51
5.2.3 Servizi e gestione operazioni asincrone 52
5.3 Containerizzazione e Deployment 54

6 Valutazione **57**
6.1 Metodologia di Valutazione 57
6.2 Scalabilità: Latency vs Time Span (aggregated vs raw) 60
6.3 Performance: Latency vs Aggregation Window (2 vs 3 measurements) . . 62
6.4 Confronto Provider LLM 64
6.4.1 Metriche di performance 64
6.4.2 Qualità semantica 65
6.4.3 Accuratezza numerica 66
6.4.4 Considerazioni comparative 68

7 Conclusioni **69**
7.1 Contributi 69
7.2 Limitazioni 69
7.3 Sviluppi Futuri 70

Abstract

Il diffondersi dei sistemi IoT (Internet of Things) in agricoltura ha reso possibile raccogliere misurazioni ambientali continue su larga scala, aprendo nuove possibilità per la gestione data-driven delle coltivazioni. Trasformare questi dati in informazioni effettivamente accessibili a operatori con competenze tecniche diverse rimane però un problema in larga parte irrisolto. Questo lavoro si inserisce nel contesto del progetto TRACE (Traceability and Resources in Agricultural Cultivation with Electronics), che applica tecnologie di Agricoltura 4.0 alla filiera delle piante officinali, aromatiche e medicinali in Emilia-Romagna. Al momento dell'avvio di questo lavoro, l'infrastruttura IoT era già operativa e i dati ambientali raccolti erano visualizzati attraverso dashboard di monitoraggio. Nonostante la solidità di questa infrastruttura, l'accesso pratico ai dati presentava limiti concreti: mancavano strumenti per l'interrogazione guidata dei dati, meccanismi per l'esportazione flessibile di dataset arbitrari e non esistevano forme di analisi interpretativa automatica. In risposta a queste esigenze è stata progettata e implementata AgroSense, una piattaforma web complementare agli strumenti esistenti. La piattaforma offre tre funzionalità principali: un query builder grafico che consente di interrogare il database senza competenze tecniche specifiche, con supporto per aggregazioni temporali configurabili e combinazione di measurement multipli; un sistema di esportazione asincrono per dataset, anche di grandi dimensioni, in formato CSV, gestito in background tramite un database in-memory; e un modulo di generazione automatica di report analitici in PDF tramite integrazione con Large Language Models, con supporto di tre provider selezionabili dall'utente: Groq, Anthropic Claude e Google Gemini. La valutazione sperimentale si é articolata su due fronti. I benchmark sul motore di query hanno mostrato che le query aggregate scalano in modo sostanzialmente sublineare al crescere dell'arco temporale, mentre le query sui dati grezzi mostrano una dispersione molto più elevata. Il confronto tra i provider LLM ha evidenziato profili distinti: Groq risulta molto più veloce degli altri provider; Gemini offre il costo per chiamata più basso e la migliore accuratezza numerica negli scenari più complessi; Claude, nonostante il costo per chiamata significativamente maggiore, non mostra vantaggi misurabili né sulla qualità semantica né sull'accuratezza, risultando la scelta meno indicata per questo contesto applicativo.

Capitolo 1

Introduzione

La crescente diffusione di sistemi IoT in ambito agricolo ha reso possibile la raccolta continua e automatizzata di dati ambientali direttamente nei campi. Sensori distribuiti su ampie superfici trasmettono misurazioni quali temperatura, umidità e pressione, a frequenza regolare, producendo dataset time-series che possono supportare decisioni agronomiche più informate e documentare in modo oggettivo le condizioni di coltivazione. Trasformare questi flussi di dati grezzi in informazioni utilizzabili da operatori con profili tecnici diversi rimane un problema aperto.

Il presente lavoro si colloca all'interno del progetto TRACE [1] (Traceability and Resources in Agricultural Cultivation with Electronics), che applica tecnologie di Agricoltura moderna alla filiera delle piante officinali, aromatiche e medicinali (MAPs) in Emilia-Romagna. Al momento dell'avvio di questo lavoro, il progetto disponeva già di un'infrastruttura IoT operativa: stazioni meteorologiche e sensori di umidità del suolo distribuiti presso le aziende agricole aderenti, con i dati raccolti persistiti su un backend centralizzato e visualizzati attraverso strumenti di monitoraggio in tempo reale.

Nonostante la solidità di questa infrastruttura, l'accesso pratico ai dati presentava limiti concreti per le diverse categorie di utenti del sistema. Gli strumenti di monitoraggio esistenti sono progettati per la visualizzazione operativa in tempo reale, ma non consentono agli utenti di costruire interrogazioni personalizzate senza competenze tecniche specifiche: selezionare un sottoinsieme di sensori, definire un arco temporale arbitrario o applicare un'aggregazione statistica richiede la conoscenza diretta del linguaggio di query del database sottostante, una barriera che esclude di fatto la maggior parte degli utenti finali. Mancavano inoltre meccanismi per esportare i dati in formati standard utilizzabili al di fuori della piattaforma. Questa limitazione ha conseguenze pratiche rilevanti per i diversi attori coinvolti nel progetto. I ricercatori del settore agronomico necessitano

[1] `https://site.unibo.it/trace/en`

di poter scaricare serie storiche complete per integrarle con strumenti di analisi specifici del proprio dominio: modelli di crescita delle colture, software di gestione aziendale, o pipeline di elaborazione statistica sviluppate nel contesto della ricerca. Gli agricoltori e i tecnici di campo vogliono poter incrociare i dati ambientali raccolti dai sensori con informazioni proprie: quantità prodotte, dati di vendita, registrazioni delle pratiche agronomiche. Senza un meccanismo di esportazione flessibile, questo tipo di analisi integrata non è praticabile. A queste esigenze si aggiunge la richiesta di forme di analisi interpretativa automatica: strumenti che non si limitino a mostrare serie numeriche, ma che traducano i dati in osservazioni comprensibili e contestualizzate.

L'obiettivo principale di questo lavoro é la progettazione e l'implementazione di AgroSense, una piattaforma web complementare agli strumenti esistenti del progetto TRACE. La piattaforma si articola attorno a tre funzionalità principali: un query builder grafico che permette di interrogare il database senza competenze tecniche specifiche, con supporto per aggregazioni temporali configurabili e combinazioni di measurement multipli; un sistema di esportazione asincrono per dataset di grandi dimensioni in formato CSV, gestito in background per non bloccare l'interfaccia durante le operazioni più pesanti; e un modulo di generazione automatica di report analitici tramite integrazione con Large Language Model commerciali, con supporto per tre provider - Groq, Anthropic Claude e Google Gemini - selezionabili dall'utente.

La valutazione sperimentale si é articolata su due fronti distinti. Il primo riguarda la scalabilità del motore di query: sono stati condotti benchmark sistematici per misurare come la latenza varia al crescere dell'arco temporale interrogato e al variare della finestra di aggregazione, confrontando query su dati aggregati e query su dati grezzi. Il secondo fronte riguarda il modulo di generazione automatica dei report: i tre provider LLM integrati sono stati confrontati su metriche di latenza, costo per chiamata, qualità semantica delle analisi prodotte e accuratezza numerica su scenari con valori di riferimento noti.

La tesi si struttura in diversi capitoli: il Capitolo 2 situa il lavoro rispetto alla letteratura esistente su IoT agricolo, piattaforme di visualizzazione time-series e impiego degli LLM nell'analisi di dati ambientali, identificando i gap che motivano lo sviluppo di AgroSense. Il Capitolo 3 descrive il progetto TRACE e i limiti degli strumenti esistenti che hanno reso necessaria la piattaforma. Il Capitolo 4 illustra le scelte architetturali e i workflow principali dell'interfaccia. Il Capitolo 5 entra nei dettagli tecnici, dalla generazione delle query alla pipeline di data cleaning, fino al sistema di export e all'integrazione LLM. Infine, il Capitolo 6 presenta i risultati sperimentali, mentre il Capitolo 7 sintetizza i contributi, discute i limiti del lavoro e traccia le possibili direzioni future.

Capitolo 2

Stato dell'Arte

Il presente capitolo situa il lavoro nel contesto della ricerca esistente, articolandosi lungo tre direzioni principali. Sul fronte dell'IoT agricolo, l'analisi si concentra sulle scelte architetturali ricorrenti nella letteratura e, in particolare, su come queste influenzano le modalità con cui i dati vengono resi accessibili a valle della raccolta. Una seconda parte passa in rassegna le piattaforme software esistenti per la visualizzazione di dati time-series in agricoltura: l'obiettivo non è censirle esaustivamente, ma capire dove si fermano e perché quelle lacune hanno reso necessario sviluppare AgroSense. L'ultima direzione riguarda i Large Language Models applicati all'interpretazione di dati agricoli — un ambito ancora in rapida evoluzione, ma già sufficientemente maturo da orientare alcune delle scelte progettuali del modulo di reportistica. Il capitolo si chiude con la sintesi dei gap identificati e il posizionamento della soluzione proposta rispetto al panorama analizzato.

2.1 IoT per il Monitoraggio Agricolo

Negli ultimi anni, sempre più aziende agricole hanno iniziato a installare sensori in campo per monitorare le condizioni del suolo e del microclima. Quello che fino a poco tempo fa richiedeva sopralluoghi manuali e misurazioni saltuarie oggi può essere registrato in continuo e trasmesso automaticamente a un sistema centrale. Le applicazioni si sono moltiplicate rapidamente: dal controllo dell'irrigazione alla sorveglianza fitosanitaria, dal monitoraggio microclimatico alla tracciabilità lungo la filiera. Quello che accomuna questi scenari è però anche il loro limite comune: raccogliere i dati è diventato relativamente semplice, ma renderli effettivamente utilizzabili da chi lavora in campo rimane un problema che la letteratura riconosce ma che le soluzioni disponibili affrontano solo in parte.

Ayaz et al. [1] offrono una rassegna sistematica che ricostruisce la traiettoria evolutiva di questi sistemi nell'ultimo decennio. Il quadro che emerge è quello di architetture che integrano reti di sensori wireless con protocolli a bassa potenza e backend cloud per rendere disponibili misurazioni ambientali con latenze contenute. Le variabili acquisite coprono uno spettro piuttosto ampio: accanto ai classici parametri del suolo come umidità, temperatura e conducibilità elettrica, molti deployment includono anche dati atmosferici - precipitazioni, irraggiamento, vento - e informazioni sullo stato operativo dei dispositivi stessi.

La rassegna metodologica di Chicaiza et al. [2] analizza le scelte tecnologiche adottate in un'ampia raccolta di lavori sperimentali nel dominio dello smart farming. Le soluzioni esaminate si organizzano tipicamente su tre livelli funzionali: dispositivi edge per la raccolta dei dati sul campo, middleware cloud per l'elaborazione e lo storage, e interfacce utente per la visualizzazione e il controllo. Sul piano della connettività, Wi-Fi risulta la tecnologia di comunicazione più diffusa nei deployment analizzati, mentre LoRaWAN è preferita negli scenari che richiedono lungo raggio e basso consumo energetico, tipici dei contesti rurali privi di infrastruttura di rete consolidata [2]. Sul piano dello storage e dell'accesso ai dati, le interfacce web e mobile rappresentano la categoria dominante nell'ambito dei casi analizzati, con soluzioni cloud generiche e piattaforme IoT specializzate che coprono quote minoritarie del panorama esaminato.

Un punto di riferimento architetturale utile per contestualizzare il presente lavoro è il progetto SWAMP [3], che ha sviluppato una piattaforma IoT per la gestione intelligente delle risorse idriche in ambito agricolo. La piattaforma è strutturata su cinque livelli funzionali e ha alimentato quattro scenari pilota tra Italia, Spagna e Brasile,una distribuzione geografica che ha messo alla prova la soluzione su condizioni ambientali e contesti operativi piuttosto eterogenei. L'aspetto forse più istruttivo dell'esperienza SWAMP, dal punto di vista del presente lavoro, non riguarda tanto l'architettura in sé quanto il fatto che sia stato necessario riconfigurare componenti della piattaforma FIWARE per adattarli ai vincoli specifici del dominio: una conferma concreta di quanto le soluzioni enterprise generiche richiedano un lavoro non banale di specializzazione prima di poter essere impiegate in contesti reali. La stessa infrastruttura SWAMP ha ispirato studi successivi sulla scalabilità e sui trade-off della rete LoRaWAN sottostante [4], a conferma di come deployment reali in ambito agricolo sollevino questioni tecniche che vanno ben oltre la semplice installazione dei sensori.

Nonostante i progressi documentati, la rassegna di Sharma e Shivandu [5] identifica nella letteratura due gap ricorrenti che rimangono largamente irrisolti: la creazione di interfacce utente accessibili ai non tecnici e la scalabilità delle soluzioni verso aziende agricole di piccole e medie dimensioni. Questi due elementi: accessibilità e scalabilità, costituiscono il filo conduttore che connette la ricerca sull'infrastruttura IoT alle esigenze concrete degli operatori del settore, e rappresentano il punto di partenza per la proposta

sviluppata in questa tesi.

Il Progetto TRACE [6], che costituisce il contesto applicativo del presente lavoro, si inserisce pienamente in questo scenario. Come verrà descritto nel Capitolo 3 questa infrastruttura, pur essendo solida dal punto di vista tecnico, presenta limitazioni nell'accesso ai dati per utenti non specializzati che motivano direttamente lo sviluppo di AgroSense.

2.2 Piattaforme di Visualizzazione e Accesso ai Dati

Il panorama degli strumenti software per la visualizzazione e l'accesso ai dati agricoli time-series è ampio e variegato. Si trovano da un lato piattaforme di monitoraggio operativo generaliste come Grafana e Kibana, progettate per offrire dashboard configurabili su qualsiasi sorgente dati. A questi si affiancano i database time-series, che costituiscono lo strato di persistenza su cui queste piattaforme si appoggiano e le cui caratteristiche architetturali influenzano direttamente le prestazioni delle query. L'analisi che segue non intende essere un puro censimento, ma identificare con precisione i limiti che ciascuna categoria presenta rispetto ai requisiti del presente lavoro, motivando le scelte progettuali di AgroSense.

Grafana rappresenta uno degli standard per la creazione di dashboard di monitoraggio in contesti IoT [7]. La piattaforma consente di definire pannelli altamente configurabili che aggregano dati da sorgenti eterogenee, tra cui database time-series dedicati. La sua diffusione capillare in contesti IoT è motivata dalla maturità del prodotto, dalla disponibilità di connettori nativi per i database time-series e dalla possibilità di configurare visualizzazioni adatte al monitoraggio operativo continuo dei sensori.

Tuttavia, Grafana presenta limiti concreti quando le esigenze degli utenti vanno oltre il monitoraggio operativo standard. Gli utenti finali possono visualizzare i grafici esistenti, ma non possono combinare misurazioni diverse al di fuori di quanto previsto in fase di configurazione. Questa rigidità è particolarmente problematica per i ricercatori del settore agronomico, che spesso hanno necessità di estrarre serie storiche specifiche per integrarle nei propri strumenti di analisi di dominio, o di costruire interrogazioni personalizzate che la dashboard preconfigurata non contempla. A queste limitazioni si aggiunge l'assenza di qualsiasi funzionalità di analisi interpretativa automatica: i grafici visualizzano i dati, ma non generano sintesi in linguaggio naturale né suggeriscono interpretazioni agronomiche.

Kibana, sviluppato nell'ecosistema Elasticsearch [8] condivide con Grafana la vocazione al monitoraggio operativo e la medesima assunzione di partenza sull'utente tecnico. Il suo punto di forza è l'integrazione con le capacità di ricerca full-text di Elasticsearch, che lo rendono adatto alla correlazione di log di sistema con dati numerici. Per un contesto di

monitoraggio di sensori agricoli con predominanza di serie temporali numeriche, questo vantaggio è marginale, mentre persistono le stesse limitazioni in termini di accessibilità delle query e flessibilità dell'esportazione.

Accanto agli strumenti di visualizzazione, un ruolo altrettanto determinante è giocato dal sistema di gestione dei dati sottostante. I database relazionali tradizionali presentano limitazioni strutturali significative quando applicati alla gestione di serie temporali dense. Il problema principale riguarda l'indicizzazione: le query su intervalli temporali estesi richiedono la scansione di indici che non sono ottimizzati per accessi sequenziali su timestamp contigui, generando un overhead crescente al crescere del volume dei dati. A questo si aggiunge l'assenza di primitive native per le operazioni più comuni nell'analisi di serie temporali: il calcolo di aggregazioni su finestre temporali fisse e il campionamento (*downsampling*) richiedono query complesse, spesso basate su sottoquery o funzioni di finestra, che il motore non può ottimizzare in modo specifico per il pattern di accesso temporale. Infine, i database relazionali non applicano alcuna compressione specifica per sequenze di valori numerici correlati nel tempo, mentre i database time-series sfruttano questa correlazione per ottenere rapporti di compressione significativamente più elevati [9, 10].

InfluxDB è uno dei sistemi più maturi e diffusi in questa categoria [9]. Il suo modello dei dati organizza le misurazioni in *measurement* - analoghe alle tabelle SQL - dove ogni punto è caratterizzato da un timestamp, da *tag* indicizzati che rappresentano metadati categorici come l'identificativo del dispositivo, e da *fields* che contengono i valori numerici effettivi. La separazione tra tag indicizzati e fields non indicizzati consente al motore di filtrare efficientemente sui metadati senza scansionare i valori numerici. Il linguaggio di query associato, Flux, è un linguaggio funzionale progettato specificamente per la manipolazione di serie temporali: adotta un modello a pipeline in cui i dati fluiscono attraverso una sequenza di trasformazioni, consentendo di esprimere filtraggio temporale, aggregazioni su finestre e join tra measurement diversi in modo componibile. Un aspetto particolarmente rilevante per AgroSense è il principio di *computation at source*: le aggregazioni vengono eseguite direttamente nel database e non nell'applicazione, così che il volume di dati trasferiti sulla rete dipende dalla finestra di aggregazione scelta e non dal numero di misurazioni grezze presenti nel periodo interrogato. Questo comportamento è alla base dei risultati dei benchmark presentati nel Capitolo 6.

TimescaleDB, costruito come estensione di PostgreSQL [10], adotta un approccio ibrido che combina il modello relazionale con ottimizzazioni specifiche per le serie temporali, tra cui il partizionamento automatico dei dati in hypertable e indici temporali. Questo lo rende particolarmente adatto a scenari in cui i dati time-series devono essere correlati con metadati relazionali complessi attraverso join SQL. Tuttavia, in scenari ad altissima frequenza di scrittura e dati puramente numerici, sistemi purpose-built come InfluxDB possono offrire prestazioni di ingestion più efficienti; per dataset costituiti esclusivamente

da misurazioni numeriche omogenee, i motori specificamente progettati per time-series possono inoltre ottenere rapporti di compressione più elevati.

Per il caso d'uso di AgroSense - interrogazione aggregata su finestre temporali estese su dati prevalentemente scalari, senza necessità di join relazionali complessi - la compatibilità SQL non rappresenta un vantaggio determinante, mentre un sistema purpose-built come InfluxDB offre un'interfaccia di query più espressiva per le operazioni tipiche dell'analisi di serie temporali.

Prometheus [11], pur essendo anch'esso un sistema di time-series, è progettato esplicitamente per il monitoraggio di infrastrutture IT e architetture a microservizi: la sua documentazione ufficiale indica che il sistema eccelle nel registrare metriche numeriche di servizi e applicazioni, adottando un modello pull in cui il server interroga periodicamente gli endpoint esposti dai target monitorati. L'analisi storica su intervalli estesi e l'export flessibile di dataset arbitrari non rappresentano i casi d'uso principali per cui Prometheus è stato progettato, rendendolo inadatto al profilo di accesso richiesto da AgroSense.

Le limitazioni fin qui descritte riguardano principalmente aspetti tecnici, ma esiste un ulteriore livello di criticità che attraversa tutte le categorie analizzate. Guardando al panorama complessivo, emerge un pattern ricorrente: le piattaforme esistenti sono progettate con l'assunzione implicita che l'utente finale abbia competenze tecniche almeno di base. La costruzione di query, la configurazione dei pannelli di visualizzazione, la scelta del formato di esportazione e l'interpretazione dei risultati restano operazioni che richiedono familiarità con i concetti fondamentali di database e analisi dei dati.

Kaloxylos et al. [12] documentano come i panel di valutazione condotti con agricoltori, agronomi ed esperti in contesto di serre greche abbiano evidenziato la necessità di strumenti sofisticati ma economicamente accessibili, e come il coinvolgimento iterativo degli utenti finali nel processo di sviluppo sia essenziale per tradurre tale necessità in scelte architetturali concrete. Questo é anche confermato da Braun et al. [13] che, nell'esperienza di sviluppo del sistema MyJohnDeere Mobile per John Deere, documentano come la progettazione di sistemi software per l'agricoltura richieda scelte architetturali specifiche e un coinvolgimento continuo degli utenti reali .

2.3 Large Language Models per l'Analisi di Dati Agricoli

L'emergere dei LLM come strumenti di uso generale ha aperto nuove prospettive anche nel dominio dell'analisi dei dati agricoli. La capacità di questi modelli di elaborare testo, ragionare su contesti complessi e generare output in linguaggio naturale li rende candidati

naturali per colmare il divario tra i dati tecnici prodotti dai sensori e l'interpretazione che gli operatori agricoli necessitano per supportare le proprie decisioni.

La rassegna condotta da Sapkota et al. [14] offre un quadro completo e aggiornato sullo stato dell'impiego dei modelli multimodali in agricoltura. Il lavoro adotta una metodologia di revisione rigorosa che ha portato a esaminare articoli da cinque banche scientifiche: ScienceDirect, IEEE Xplore, PubMed, Web of Science e Scopus; selezionando infine 84 contributi direttamente rilevati a partire da 460 lavori inizialmente individuati, ridotti poi a 207 dopo una prima fase di scrematura per pertinenza e duplicazione. L'analisi si articola attorno a undici domande di ricerca, quattro di carattere generale sulla natura dei modelli multimodali e sette specificamente orientate alle applicazioni agricole. Le applicazioni vengono quindi catalogate in cinque categorie principali: monitoraggio e gestione delle colture, rilevamento di parassiti e malattie, irrigazione e fertilizzazione di precisione, strumenti informativi e di supporto decisionale, e servizi di estensione agricola. I risultati mostrano come i modelli multimodali abbiano conseguito progressi significativi nell'elaborazione automatizzata di immagini di colture, nel rilevamento precoce di patologie e nella generazione di raccomandazioni agronomiche personalizzate. Le sfide principali identificate dalla rassegna sono tre: la qualità e la disponibilità dei dati, con evidenza di alta variabilità geografica e frammentazione delle fonti; gli elevati costi computazionali richiesti per il training e il deployment, che costituiscono una barriera concreta nei contesti rurali con accesso limitato alle infrastrutture tecnologiche; e l'alfabetizzazione digitale limitata degli utenti finali nelle aree agricole, che ostacola l'adozione su larga scala. La conclusione principale é che, sebbene il potenziale tecnico sia ampiamente dimostrato, l'integrazione di questi modelli in sistemi operativi accessibili agli utenti finali rimane un problema in larga parte aperto.

Tra i contributi esaminati nella rassegna, quello di Kuska [15] affronta in modo diretto la questione del valore pratico degli LLM nell'agricoltura operativa. Partendo dall'osservazione che i modelli linguistici sono fondamentalmente addestrati su testi e non su dati numerici strutturati, gli autori delimitano con precisione i confini entro cui questi strumenti possono portare contributo concreto. Il lavoro identifica quattro categorie di casi d'uso in cui gli LLM possono generare valore sostanziale: la *consulenza e assistenza contestualizzata* alle condizioni della singola azienda, che può includere raccomandazioni su tempi di intervento, scelta degli strumenti e gestione delle pratiche colturali; la *documentazione automatizzata*, intesa come traduzione di dati leggibili dalle macchine, provenienti dal sistema di gestione aziendale e tracker di macchinari, in testi comprensibili per l'operatore; la *spiegazione e la formazione*, con generazione automatica di materiali didattici come manuali, tutorial e guide pratiche adattati al livello tecnico dell'utente; e infine l'*interpretazione dei risultati di modelli predittivi*, ovvero la conversione di output statistici complessi in indicazioni operative concrete per il supporto alle decisioni. Quest'ultima categoria é quella che più direttamente caratterizza la funzionalità di reportistica automatica implementata in AgroSense. Kuska et al. sottolineano anche i limiti

strutturali di questo approccio: l'output degli LLM é probabilistico e la sua affidabilità dipende criticamente dalla qualità e dalla validazione delle basi di conoscenza sottostanti; la trasferibilità delle raccomandazioni tra contesti geografici e legislativi diversi non é garantita; e l'integrazione diretta di dati numerici per analisi predittive rimane fuori dalla portata dei modelli attuali senza un'integrazione specifica che vada oltre la semplice interpretazione testuale.

Oltre all'analisi dei dati storici, la letteratura recente esplora l'integrazione degli LLM direttamente nelle pipeline operative dei sistemi IoT agricoli. Il progetto AgriNex [16], proposto da Hazarika et al., introduce un'architettura a due livelli in cui droni UAV fungono da nodi intermedi tra i sensori a terra e i server cloud, risolvendo il problema della connettività nelle aree rurali prive di infrastrutture di rete consolidata. L'innovazione centrale del contributo è il concetto di *Semantic Criticality Index* (SCI), un indicatore che quantifica dinamicamente l'importanza di ciascun dato acquisito dai sensori in funzione del contesto colturale corrente, delle condizioni ambientali e della fase stagionale. Dal punto di vista implementativo, il sistema utilizza un modello BERT fine-tuned per classificare i dati dei sensori in quattro livelli di criticità semantica -normale, anomalo, critico o urgente- attraverso un processo di embedding contestuale che integra trend storici, correlazioni spaziali tra sensori vicini e conoscenza di dominio sulla fase di crescita della coltura. Il valore dell'SCI viene poi sfruttato da un algoritmo di ottimizzazione basato su reinforcement learning, il Semantic-Guided Deep Q-Network (SG-DQN), per pianificare in modo adattivo la raccolta dei dati dai sensori, bilanciando la necessità di acquisire informazioni critiche con il risparmio energetico di UAV e sensori. I risultati sperimentali su dataset pubblici mostrano un miglioramento del 66% rispetto al DQN standard e del 71% rispetto al Q-Learning nella funzione di reward cumulativa, con risparmi energetici fino al 50% nel sensing e al 60% nella trasmissione rispetto ai metodi tradizionali di scheduling. AgriNex illustra come gli LLM possano andare ben oltre il ruolo di strumenti di post-elaborazione per diventare componenti attive nella gestione delle pipeline IoT, aprendo scenari di integrazione ancora in larga parte inesplorati.

Contributi recenti mostrano come questa integrazione possa operare a livelli diversi della pipeline dei dati. Fang et al. [17] propongono Agri-LLM, un modello che affronta il problema dell'imputation (riempimento dei dati mancanti) e del forecasting (previsione) di serie temporali di emissioni di gas serra raccolte da sensori IoT agricoli in 43 paesi tra il 1990 e il 2020. L'architettura del modello si articola in tre componenti principali: un modulo di *information fusion embedding* che integra i valori mancanti con i corrispondenti pattern di assenza e le irregolarità temporali per generare token multi livello; un modulo di *global temporal similarity informed prompting* che seleziona prompt basati su caratteristiche temporali simili estratte da altri nodi della rete sensoriale; e un LLM pre-addestrato con pesi congelati che allinea direttamente le rappresentazioni numeriche con la conoscenza linguistica senza richiedere fine-tuning del backbone. L'aspetto metodologicamente più rilevante dell'approccio é proprio il ricorso al *prompt learning* anziché

al fine-tuning: mantenendo i pesi del modello congelati, Agri-LLM riesce a operare efficacemente anche in scenari con disponibilità limitata di dati di addestramento. Gli esperimenti dimostrano miglioramenti del 21,7% in MAE (Mean Absolute Error), 31,7% in RMSE (Root Mean Square Error) e 11,61% in MAPE (Mean Absolute Percentage Error) rispetto al secondo miglior modello nell'imputation task, calcolati come media sui due dataset di valutazione utilizzati. Il vantaggio si mantiene anche in scenari few-shot con solo il 5% dei dati di training, condizione in cui il modello continua a superare i competitor addestrati sull'intero dataset. Il contributo dimostra in modo convincente che gli LLM possono elaborare direttamente dati numerici time-series senza passare per rappresentazioni testuali intermedie, ampliando il perimetro applicativo di questi modelli ben oltre la generazione di linguaggio naturale.

Leite et al. [18] affrontano invece il livello interpretativo e decisionale, investigando l'impatto della Retrieval-Augmented Generation (RAG) sulla qualità delle risposte di un LLM applicato al controllo ambientale in allevamenti avicoli. Il sistema analizza misurazioni di temperatura, umidità relativa e concentrazione di gas prodotte da sensori IoT e genera report automatici con raccomandazioni operative. L'architettura RAG integra un meccanismo di recupero del contesto esterno che fornisce al modello informazioni aggiornate e specifiche del sito produttivo, permettendo di superare due limitazioni note dei modelli linguistici: le allucinazioni e la generalizzazione eccessiva delle risposte. Gli esperimenti condotti con GPT-4o mostrano che l'integrazione del layer RAG produce risposte significativamente più precise e contestualmente rilevanti rispetto all'LLM utilizzato senza recupero del contesto. I test si basano sulle dieci pubblicazioni più citate sul controllo ambientale in avicoltura come corpus di riferimento, consentendo una valutazione quantitativa della correttezza delle risposte generate. Le limitazioni principali identificate dagli autori riguardano la dipendenza dalla qualità del sistema e dall'aggiornamento del corpus recuperato, le difficoltà nella gestione di inconsistenze nelle informazioni estratte, e le sfide di scalabilità del framework RAG in contesti operativi reali.

Questi quattro lavori illustrano come l'integrazione LLM-IoT possa operare a livelli distinti e complementari: dall'elaborazione diretta delle serie temporali nel caso di Agri-LLM, alla gestione intelligente delle risorse di rete nel caso di AgriNex, alla classificazione dei casi d'uso nel caso di Kuska, e all'interpretazione e reportistica contestualizzata nel caso di Leite.La prospettiva comune che emerge é la fattibilità tecnica dimostrata in contesti specifici, a fronte di limiti ancora aperti che riguardano la scalabilità, la generalizzazione e l'accessibilità per gli utenti finali.

2.4 InfluxDB: Architettura e Principi di Funzionamento

InfluxDB occupa un ruolo centrale nell'infrastruttura del progetto TRACE e nella piattaforma AgroSense, comprendere come il sistema gestisce internamente la scrittura, la compressione e il recupero dei dati é necessario per contestualizzare la collocazione del sistema nel contesto del progetto e per interpretare correttamente il comportamento osservato nei benchmark presentati nel Capitolo 6.

2.4.1 Modello dei dati e storage engine

InfluxDB organizza i dati attorno a un modello che si discosta in modo significativo dal paradigma relazionale [9, 19]. All'interno di un bucket - l'unità di organizzazione a cui viene associata una *retention policy* che determina per quanto tempo i dati vengono conservati - i punti sono raggruppati per *mesurement*, concetto che per analogia ricorda una tabella SQL ma con semantica diversa: ogni measurement raccoglie tutti i punti relativi a uno stesso tipo di osservazione.

Ogni punto di dato é identificato da quattro elementi: il *timestamp*, con precisione configurabile fino al nanosecondo; i *tag* , coppie chiave-valore che rappresentano metadati categorici come l'identificativo del dispositivo o la località; i *field* , coppie chiave-valore che contengono i valori numerici o stringa effettivi della misurazione; e il nome del measurement. La primary key di un punto é data dalla combinazione di timestamp e tag set. La distinzione tra tag e field non é accessoria: i tag vengono indicizzati dal motore di storage, mentre i field non lo sono. Le query che filtrano per valori di tag possono quindi sfruttare l'indice per accedere direttamente ai dati rilevanti senza scansionare l'intero dataset. I field non vengono indicizzati perché indicizzare valori numerici continui con alta cardinalità sarebbe computazionalmente inefficiente rispetto ai pattern di accesso tipici delle serie temporali. La scelta di quali attributi modellare come tag e quali come field é pertanto una decisione con impatto diretto sulle prestazioni delle query. Il sistema di persistenza di InfluxDB si basa sul *Time-Structured Merge Tree* (TSM), una struttura dati sviluppata internamente da InfluxData e derivata concettualmente dagli LSM tree (Log-Structured Merge Tree), ma con modifiche specifiche per le serie temporali [19].

Prima di adottare il TSM, InfluxDB ha attraversato diverse iterazioni dello storage engine: inizialmente utilizzava LevelDB, che ottimizza il throughput in scrittura e offre compressione integrata, ma non supporta backup a caldo; successivamente ha sperimentato varianti come RocksDB e HyperLevelDB, anch'esse basate su LSM tree. Il passaggio a BoltDB (basato su B+ tree) ha migliorato la stabilità, ma ha introdotto picchi di IOPS al crescere della dimensione del database. Questi problemi hanno portato InfluxData a sviluppare il TSM, che riunisce la velocità in scrittura degli LSM tree con una gestione

efficiente delle cancellazioni massive richieste dalla retention automatica - operazione notoriamente costosa negli LSM tree tradizionali. Rispetto al precedente B+ tree, il TSM ha portato un miglioramento di circa 45x nell'utilizzo di spazio su disco [19]. Come gli LSM tree, il TSM utilizza un *Write Ahead Log* (WAL) per garantire la durabilità dei dati in caso di crash, file indice in sola lettura, e un processo di compaction periodica che aggrega i file più piccoli in file più grandi, eliminando contestualmente i dati scaduti secondo la retention policy.

Sul fronte della compressione, i database time-series adottano algoritmi differenziati per tipo di dato che sfruttano le proprietà statistiche di ciascun campo. I timestamp, tipicamente equispaziati in serie a frequenza fissa, si prestano a una codifica delta che memorizza la differenza tra valori consecutivi anzichè il valore assoluto. Per i valori numerici in virgola mobile, i principali database time-series adottano l'algoritmo Gorilla per la compressione delle serie [20]. Il risultato complessivo é che un database time-series occupa su disco significativamente meno spazio rispetto a una soluzione relazionale tradizionale contenente gli stessi dati, con benefici proporzionali anche sui tempi di I/O nelle query su grandi intervalli temporali [19].

2.4.2 Flux e gestione del ciclo di vita dei dati

Il linguaggio di query di InfluxDB é Flux, un linguaggio funzionale a pipeline progettato specificamente per la manipolazione di serie temporali [9]. Una query Flux si compone di una sequenza di funzioni che trasformano progressivamente i dati: dalla sorgente (`from()`), attraverso il filtro temporale (`range()`) e il filtro sui metadati (`filter()`), fino alle trasformazioni come `aggregateWindow()` e `mean()`. Il principio fondamentale che governa le prestazioni delle query aggregate è il *computation at source*: le aggregazioni vengono eseguite internamente al database durante la lettura, quindi il volume di dati trasferiti dipende dalla finestra di aggregazione scelta e non dal numero di misurazioni grezze nel periodo interrogato.

La gestione del ciclo di vita dei dati avviene attraverso due meccanismi complementari. La *retention policy* associata a ciascun bucket specifica per quanto tempo i dati vengono conservati prima di essere eliminati automaticamente durante le operazioni di compaction [19]: un aspetto particolarmente rilevante per i sistemi IoT in produzione, dove dati ad alta risoluzione temporale accumulati per anni occuperebbero volumi di storage non sostenibili. Il *downsampling* complementa questo meccanismo aggregando periodicamente i dati ad alta frequenza in serie a risoluzione più bassa, persistite in un secondo bucket con retention più lunga, così da mantenere, ad esempio, i dati al minuto per i tre mesi più recenti e i dati orari per i cinque anni precedenti, con un costo di storage complessivo contenuto. InfluxDB supporta questo pattern attraverso le *tasks*, processi pianificati che eseguono periodicamente query Flux di aggregazione e scrivono i risultati nei bucket di destinazione.

Rispetto ai database relazionali tradizionali, InfluxDB offre prestazioni superiori per i pattern di accesso tipici delle serie IoT grazie all'architettura TSM, alla compressione specifica per tipo di dato e alle primitive native di Flux per finestre temporali e downsampling. I benchmark riportati in [19] su un dataset di oltre 30 milioni di record mostrano che InfluxDB occupa circa 27 volte meno spazio su disco rispetto a SQL Server e risulta mediamente 8 volte più veloce nelle query, con picchi di 20x per le query aggregate su intervalli temporali ampi. Rispetto a TimescaleDB [10], costruito come estensione di PostgreSQL con compatibilità SQL completa, InfluxDB offre prestazioni di ingestion più elevate su workload con alta frequenza di scrittura di valori scalari, al prezzo di un modello di dati meno flessibile che non supporta nativamente join relazionali con tabelle di metadati arbitrarie. Rispetto a Prometheus [11], ottimizzato per il monitoraggio di infrastrutture IT con modello pull e retention breve, InfluxDB é progettato per retention lunghe e analisi storica su intervalli estesi.

2.5 Gap Identificati

L'analisi della letteratura presentata nelle sezioni precedenti permette di identificare tre gap distinti che motivano direttamente la proposta di AgroSense.

Il **primo gap** riguarda l'accessibilità degli strumenti di interrogazione dei dati IoT. Le piattaforme di visualizzazione più diffuse offrono capacità analitiche potenti, ma richiedono all'utente la conoscenza di linguaggi di query specifici o la configurazione manuale. La letteratura evidenzia in modo ricorrente la necessità di strumenti accessibili agli utenti finali [5, 12, 13] ma non affronta in modo esplicito il problema dell'integrazione.

Il **secondo gap** riguarda l'esportazione flessibile e asincrona dei dati. Le esigenze dell'utente che ha bisogno di analizzare i dati fuori dalla piattaforma - in fogli di calcolo, strumenti statistici, o applicazioni di visualizzazione dedicate - non sono adeguatamente coperte dalle soluzioni esistenti, che tipicamente offrono export in formati fissi o richiedono accesso diretto alle API.

Il **terzo gap** è la limitata disponibilità di piattaforme integrate che combinino interrogazione dei dati, esportazione e generazione automatica di report basati su LLM. I contributi recenti su questa tecnologia [14, 17, 18] mostrano la fattibilità dell'approccio ma non presentano sistemi completi.

AgroSense si posiziona come risposta diretta a questi tre gap: una piattaforma web che offre un query builder grafico per la costruzione di interrogazioni su dati time-series InfluxDB, un sistema di esportazione asincrona in formato CSV, e un modulo di generazione automatica di report analitici tramite integrazione con modelli LLM commerciali.

Capitolo 3

Il Progetto TRACE

3.1 Agricoltura di Precisione e Settore MAPs

L'agricoltura contemporanea affronta pressioni crescenti: aumentare la produzione con risorse limitate, ridurre l'impatto ambientale, adattarsi a condizioni climatiche sempre più variabili. L'agricoltura di precisione risponde a queste sfide introducendo, al posto delle pratiche tradizionali uniformi, interventi differenziati basati su dati raccolti direttamente in campo.

L'IoT è al centro di questa trasformazione. Attraverso reti di sensori wireless distribuite nei campi, è possibile monitorare continuamente le condizioni ambientali e del suolo. Temperatura, umidità dell'aria e del terreno, precipitazioni, velocità del vento: tutti questi parametri vengono registrati automaticamente, generando dataset time-series con alta risoluzione temporale. Questi dati permettono di prendere decisioni agronomiche più informate, dall'irrigazione mirata alla prevenzione di stress idrici o termici.

Implementare sistemi IoT in agricoltura significa però confrontarsi con vincoli operativi molto rigidi. I dispositivi devono essere efficienti dal punto di vista energetico, dato che vengono installati in luoghi senza alimentazione stabile e devono funzionare per periodi lunghi (intere stagioni colturali). La comunicazione deve essere affidabile anche con maltempo, interferenze radio o guasti locali. La robustezza nel tempo è essenziale per ottenere dataset completi e continuativi.

I parametri atmosferici tipicamente monitorati includono temperatura e umidità dell'aria, pressione atmosferica, precipitazioni, velocità e direzione del vento, mentre per quanto riguarda il suolo vengono acquisite misure di temperatura, umidità volumetrica e conducibilità elettrica. Tutte queste misurazioni vengono trasmesse tramite protocolli

wireless a lungo raggio verso sistemi centralizzati, dove i dati vengono memorizzati e resi disponibili per l'analisi.

Il monitoraggio IoT è particolarmente rilevante per le piante officinali, aromatiche e medicinali (MAPs - Medicinal and Aromatic Plants). Queste specie vegetali trovano impiego in ambito farmaceutico, cosmetico e alimentare, e sono caratterizzate da elevata sensibilità alle condizioni di coltivazione. A differenza delle colture commodity, dove la qualità si misura principalmente in termini di resa quantitativa, nelle MAPs il valore del prodotto dipende dalla concentrazione del principio attivo.

Ed è proprio la qualità e la concentrazione di tali principi attivi che sono fortemente influenzate da fattori ambientali come temperatura, umidità, radiazione solare e disponibilità idrica. Questa sensibilità rende il monitoraggio IoT non un semplice supporto operativo, ma uno strumento essenziale per ottimizzare la qualità del prodotto, ridurre la variabilità inter-stagionale e individuare tempestivamente condizioni critiche quali gelate, deficit idrici o eccessi termici.

A questa complessità agronomica si aggiunge la crescente richiesta di tracciabilità completa della filiera produttiva. Il monitoraggio IoT assume quindi un ruolo che va oltre il supporto operativo, configurandosi come elemento abilitante per la documentazione oggettiva delle condizioni di coltivazione. I produttori devono fornire certificazioni biologiche conformi ai regolamenti europei, registrare in modo verificabile le pratiche agronomiche, e dimostrare la conformità agli standard qualitativi richiesti dal settore farmaceutico. I sistemi IoT forniscono evidenze oggettive e automatiche di queste condizioni, trasformando la compliance normativa da onere documentale manuale a processo automatizzato.

È in questo contesto che nasce il progetto TRACE (Traceability and Resources in Agricultural Cultivation with Electronics), con l'obiettivo di introdurre tecnologie Agricoltura 4.0 nella filiera delle MAPs in Emilia-Romagna. Il progetto integra monitoraggio IoT, strumenti di analisi dati avanzata e sistemi di tracciabilità digitale per fornire a produttori, consorzi e ricercatori gli strumenti necessari per una gestione data-driven delle coltivazioni di piante medicinali. Al momento dell'avvio di questo lavoro, l'infrastruttura IoT del progetto TRACE era già operativa e raccoglieva continuamente dati ambientali e pedologici da sensori distribuiti nei campi delle aziende agricole aderenti. Il presente lavoro si focalizza sull'implementazione di una piattaforma per l'accesso avanzato e l'analisi intelligente dei dati raccolti dal sistema.

3.2 Architettura del Progetto TRACE

TRACE nasce per portare innovazione nel settore delle MAPs mediante l'integrazione di tecnologie digitali avanzate. Tre sono gli obiettivi che il progetto si pone di realizzare: il

primo è assicurare un'alta qualità delle coltivazioni mediante un monitoraggio in tempo reale con pratiche di agricoltura di precisione; il secondo scopo consiste nel garantire la certificazione e la tracciabilità dei prodotti con sistemi di registrazione sicura; infine, il terzo obiettivo è di incrementare la produttività, contestualmente con la sostenibilità, basando le decisioni sui dati, in modo da combinare informazioni provenienti da più sensori IoT, dati storici aziendali e tecniche di elaborazione avanzate.

L'architettura implementata per raggiungere questi obiettivi, rappresentata nella Figura 3.1 [6], si articola in tre componenti logiche integrate. Il primo livello, dedicato al monitoraggio IoT, si occupa della raccolta dati ambientali e pedologici sul campo. Il secondo livello gestisce integrazione dati e intelligenza, ed è responsabile dello storage, dell'accesso alle informazioni e dell'elaborazione analitica. Il terzo livello, focalizzato sulla tracciabilità e certificazione, fornisce registrazioni delle operazioni agricole lungo l'intera filiera produttiva.

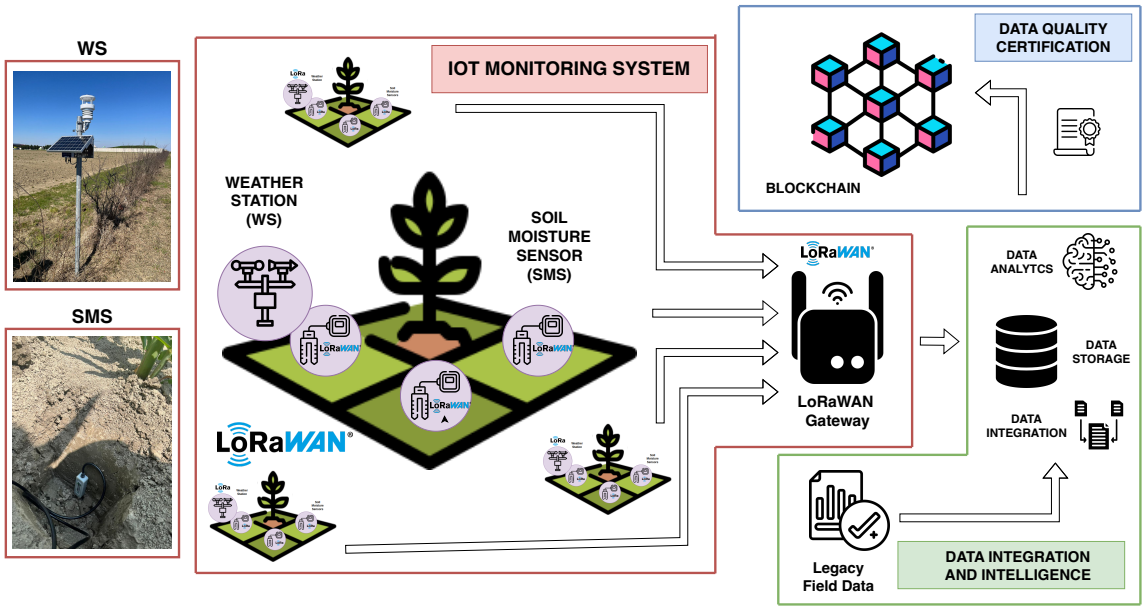


Figura 3.1: Architettura del sistema TRACE [6].

Il sistema di monitoraggio IoT costituisce l'infrastruttura di base per l'acquisizione continuativa di misurazioni nelle zone coltivate. La raccolta si basa su due tipologie di dispositivi complementari: le Weather Station (WS), dedicate ai parametri atmosferici, e le Soil Moisture Stations (SMS), specializzate nel monitoraggio pedologico. Le WS acquisiscono temperatura dell'aria, umidità relativa, pressione atmosferica, quantità di precipitazioni, intensità e direzione del vento. Le SMS, posizionate nel terreno a circa 30 centimetri di profondità, rilevano temperatura del suolo, contenuto idrico volumetrico e conducibilità elettrica.

L'implementazione risponde ai requisiti operativi stringenti che caratterizzano i sistemi IoT per l'agricoltura di precisione [6]: l'autonomia energetica prolungata, necessaria poiché i dispositivi operano in ambienti privi di alimentazione stabile per intere stagioni colturali, é garantita dall'adozione di dispositivi commerciali dotati di pannelli solari integrati; la comunicazione affidabile in condizioni atmosferiche avverse o in presenza di guasti ai nodi é assicurata dalla tecnologia LoRaWAN -standard di comunicazione wireless a lungo raggio e basso consumo energetico progettato per reti IoT distribuite su ampie aree geografiche - con tre gateway posizionati in punti sopraelevati; la durabilità nel tempo, fondamentale per ottenere dataset completi e privi di discontinuità, é conseguita tramite l'impiego di dispositivi progettati specificamente per deployment agricoli in campo aperto.

Nel contesto di TRACE, il deployment sul territorio dell'Emilia-Romagna comprende otto WS distribuite presso le aziende agricole partecipanti, affiancate da venti SMS posizionate per assicurare copertura rappresentativa delle differenti condizioni pedologiche. L'area monitorata presenta un'estensione massima di circa dieci chilometri tra i punti più distanti. La copertura di rete è garantita da tre gateway LoRaWAN installati in posizioni sopraelevate su edifici selezionati, ciascuno collegato a Internet mediante connettività 5G o ethernet.

I dati raccolti dai sensori vengono trasmessi tramite la rete LoRaWAN a un'infrastruttura centralizzata composta da un'applicazione custom che gestisce la ricezione dei messaggi, la decodifica dei payload dei dispositivi, il filtraggio dei campioni anomali e la persistenza dei dati validati [6]. La scelta di InfluxDB come sistema di storage é motivata dalla natura stessa dei dati prodotti: misurazioni numeriche scalari raccolte ad intervalli regolari per periodi estesi, un profilo di accesso per cui i database time-series offrono prestazioni superiori rispetto ai sistemi relazionali tradizionali [9]. L'obiettivo complessivo di questa pipeline é garantire che i dati prodotti dai sensori siano disponibili in modo affidabile, continuo e in una forma ottimizzata per le interrogazioni.

Il livello di integrazione dati e intelligenza combina le informazioni provenienti dai sensori IoT con altre fonti informative aziendali quali statistiche di resa, risultati di analisi del suolo, e documentazione delle pratiche agronomiche. L'obiettivo è fornire supporto decisionale mediante identificazione di pattern nei dati, previsione di situazioni critiche, e formulazione di strategie operative ottimizzate.

Per l'accesso e la visualizzazione dei dati, il livello si avvale di Grafana [7], una piattaforma open source per il monitoraggio e la creazione di dashboard interattive su sorgenti eterogenee, configurata per interrogare direttamente InfluxDB e fornire visualizzazioni in tempo reale dei parametri ambientali e pedologici..

Il terzo livello dell'architettura TRACE gestisce la tracciabilità e certificazione lungo l'intera catena di produzione delle MAPs. Nel settore delle piante officinali, produttori,

trasformatori e distributori devono poter dimostrare in modo verificabile le condizioni di coltivazione e le pratiche agronomiche adottate, sia per rispettare le normative vigenti sia per consolidare la fiducia tra gli attori della filiera A questo scopo vengono impiegati i Digital Field Notebook (DFN) [6], strumenti digitali per la registrazione delle operazioni agricole. Tuttavia la semplice registrazione non garantisce di per sé l'integrità dei dati nel tempo: un registro centralizzato può essere modificato o alterato. L'integrazione con tecnologie blockchain risolve questo problema rendendo i record immutabili e crittograficamente verificabili da qualsiasi attore della supply chain, senza necessità di affidarsi a un'autorità centrale.

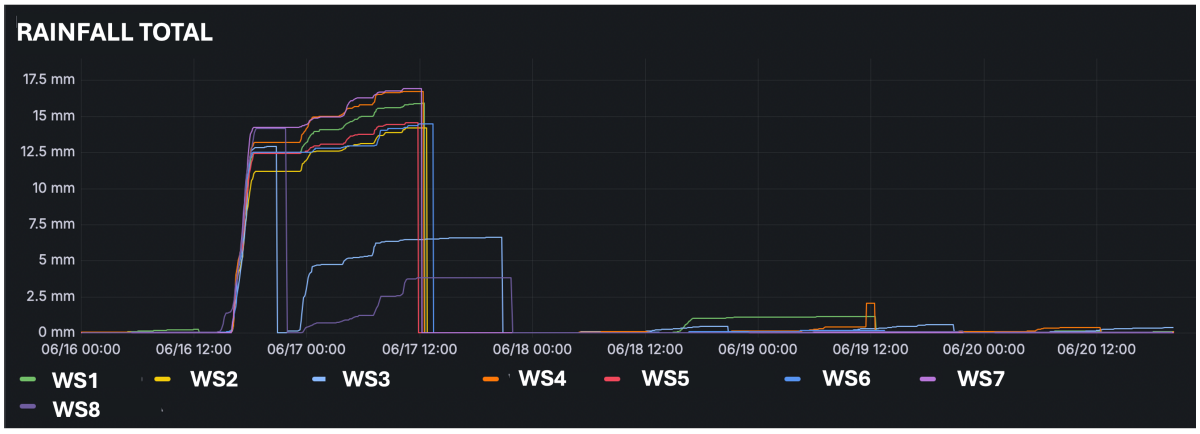
3.3 Implementazione del Progetto TRACE

Elemento centrale del progetto TRACE è l'implementazione concreta del sistema, sulla base dell'architettura precedentemente illustrata. Questa implementazione è stata svolta seguendo i tre requisiti fondamentali dello smart farming: efficienza energetica, affidabilità della comunicazione, e robustezza operativa nel lungo periodo.

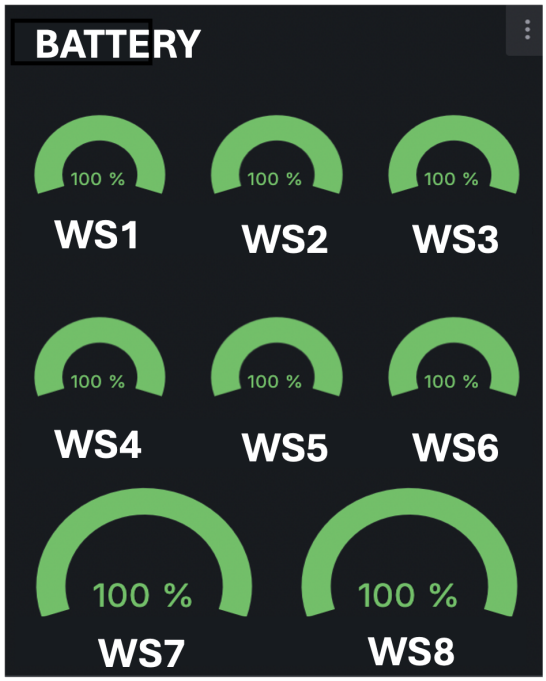
Per soddisfare questi vincoli sono state adottate scelte progettuali specifiche. La comunicazione tra sensori e gateway sfrutta la tecnologia LoRa attraverso l'infrastruttura LoRaWAN, gestita tramite il server open-source ChirpStack [21].

L'utilizzo delle tecnologie LoRa e LoRaWAN è ormai consolidato nell'ambito agricolo, grazie alla capacità di coprire distanze considerevoli con consumi energetici ridotti e di trasmettere dati a più gateway contemporaneamente. Progetti simili come SWAMP (Smart Water Management Platform) [3] hanno implementato la stessa tecnologia per la trasmissione dei dati con ottimi risultati.

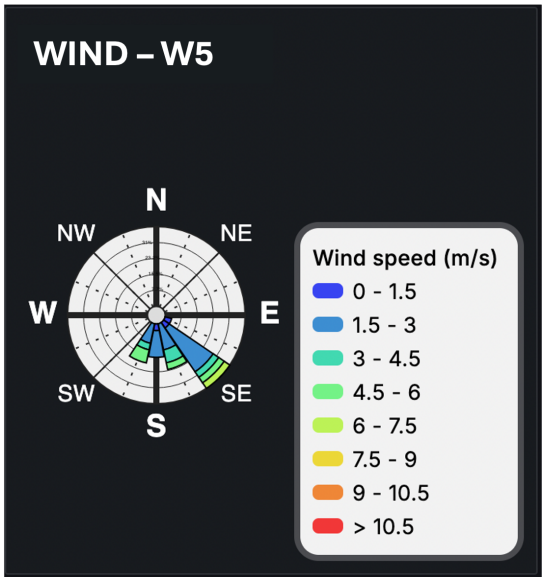
Per consentire la visualizzazione e l'interrogazione di questi dati sono stati implementati diversi strumenti. La dashboard Grafana é configurata per interrogare direttamente InfluxDB con query Flux. I pannelli sono organizzati per categoria di parametro, come mostrato nelle Figure 3.2b 3.2a 3.2c: grafici temporali per i parametri continui come temperatura, umidità e precipitazioni cumulative; pannelli di stato per il monitoraggio operativo dell'infrastruttura, tra cui i livelli di carica delle batterie dei dispositivi; rose dei venti per la rappresentazione della distribuzione direzionale e dell'intensità del vento.



(a) Visualizzazione delle precipitazioni cumulative per ciascuna WS



(b) Monitoraggio del livello di carica delle batterie



(c) Rosa dei venti per la stazione WS5

Figura 3.2: Esempi di visualizzazioni implementate nella dashboard Grafana del progetto TRACE. Originale: [6]

Nonostante l'efficacia di Grafana per il monitoraggio operativo, durante l'utilizzo del sistema sono emerse esigenze che gli strumenti esistenti non riescono a soddisfare pie-

namente. Ricercatori coinvolti nel progetto hanno manifestato la necessità di estrarre dataset con aggregazioni temporali personalizzate, combinare parametri provenienti da sensori differenti con logiche complesse, ed esportare i risultati in formati standard per elaborazioni successive. Agronomi e tecnici delle aziende agricole hanno espresso interesse per strumenti che permettano interrogazioni semplici senza richiedere competenze tecniche avanzate e conoscenza del linguaggio di query Flux. Inoltre, è emersa la richiesta di capacità analitiche che vadano oltre la semplice visualizzazione: identificazione automatica di correlazioni tra variabili, rilevamento di anomalie nei dati, generazione di report interpretativi che contestualizzino i valori numerici in un framework agronomico comprensibile.

Il presente lavoro di tesi risponde a queste esigenze implementando una piattaforma web complementare agli strumenti esistenti. La piattaforma fornisce un'interfaccia user-friendly per la costruzione di query personalizzate, supporto per aggregazioni temporali flessibili e combinazioni complesse di measurements, sistema di export in formati multipli, e integrazione con Large Language Models per l'analisi automatica e la generazione di report. I capitoli successivi descrivono la metodologia di sviluppo adottata, l'architettura tecnica implementata, e i risultati ottenuti nella validazione del sistema.

Capitolo 4

Progettazione

La piattaforma AgroSense è stata sviluppata con lo scopo di estendere le capacità di accesso e analisi dei dati raccolti dal progetto TRACE. Come descritto nel capitolo precedente, l'interfaccia Grafana di cui era dotato il sistema, nonostante l'efficacia per il monitoraggio operativo, presenta dei limiti per gli utenti che necessitano di estrarre dataset personalizzati o di eseguire analisi più approfondite. AgroSense si pone quindi come piattaforma complementare, offrendo funzionalità per query builder intuitivi, un sistema di export asincrono per dataset di grandi dimensioni e l'integrazione di Large Language Models per la generazione automatica di report analitici.

Sebbene AgroSense sia stata sviluppata e validata nel contesto specifico del progetto TRACE, la sua architettura non dipende da questo: la piattaforma si interfaccia con qualsiasi istanza di InfluxDB, indipendentemente dal dominio applicativo o dalla tipologia di sensori che la alimentano. I measurement, i device e i field vengono enumerati dinamicamente dallo schema del database al momento dell'accesso. Questo significa che la piattaforma può essere impiegata per monitorare reti di sensori in contesti differenti, semplicemente puntando ad un'istanza InfluxDB diversa tramite le variabili d'ambiente. La scelta del progetto TRACE come contesto di riferimento ha orientato alcune decisioni di design, in particolare la pipeline di data cleaning e i prompt inviati agli LLM, ma queste componenti sono parametrizzabili e adattabili senza modifiche strutturali al sistema.

4.1 Panoramica Generale

La piattaforma è organizzata con un'architettura a tre livelli, come illustrato in Figura 4.1. Questo consente di mantenere separate le diverse parti del sistema: la presentazione,

la logica applicativa e i dati. Ogni componente può così essere sviluppato e testato indipendentemente, rendendo più leggibile e manutenibile il codice.

Il primo livello è il Presentation Layer, costituito dall'applicazione web realizzata con Angular [22]. Gli utenti la utilizzano per costruire le query, visualizzare i risultati in tabelle, richiedere export e report. Per lo sviluppo dell'interfaccia sono stati sfruttati gli Angular Material [23], componenti grafici responsive e moderni. L'applicazione viene quindi servita tramite Nginx [24], che non svolge solo il ruolo di web server ma anche quello di reverse proxy, inoltrando le richieste API dal frontend al backend.

Il secondo livello è il Business Logic Layer, implementato con NestJS [25]. Il backend coordina tutte le operazioni. Riceve le richieste dal frontend, verifica che l'utente sia autenticato usando un token JWT, per le operazioni sensibili, costruisce le query per InfluxDB, processa i dati e li restituisce. NestJS permette di organizzare il codice in moduli, dove ogni modulo si occupa di una funzionalità specifica. Nella piattaforma sono presenti 8 moduli che sono: il modulo Auth gestisce login e sicurezza, quello Users amministra gli utenti e il workflow di approvazione, il modulo InfluxModule che si occupa della comunicazione con il database, con il modulo Export si gestisce l'esportazione di dataset utilizzando Redis [26], un database in-memory usato come store per la gestione di code di job asincroni, il modulo Report si occupa della generazione di report analitici tramite chiamate a servizi di intelligenza artificiale, il modulo DatabaseModule che si occupa della persistenza di SQLite, il modulo Experimentation che invece si occupa della valutazione e comparazione dei diversi LLM integrati, e infine il modulo ApiKeyModule che gestisce il salvataggio e il recupero delle chiavi API personali degli utenti per i provider LLM. Questa struttura modulare rende il codice più ordinato e manutenibile.

Il terzo livello è il Data Layer, che utilizza tre tecnologie diverse. SQLite [27] viene usato per memorizzare le informazioni sugli utenti (credenziali, nome, stato di approvazione e data di creazione) e le chiavi API associate a ciascun utente. E' stato scelto in quanto è un database relazionale leggero e che non necessita di una configurazione complessa. InfluxDB è il database che già utilizzava il progetto TRACE e contiene tutte le misurazioni temporali di sensori IoT, essendo ottimizzato per le query su serie temporali. Infine Redis serve per gestire le code dei job di export, permettendo di processare richieste di esportazione pesanti in modo asincrono senza bloccare l'interfaccia utente. Le due tecnologie rispondono a requisiti fondamentalmente diversi: SQLite garantisce la persistenza strutturata e durabile dei dati mentre Redis é ottimizzato per l'accesso in memoria ad alta velocità, rendendolo particolarmente adatto alla gestione di job transitori dove la priorità é la velocità di lettura e scrittura, non la persistenza a lungo termine.

L'integrazione con i servizi AI esterni avviene tramite gli SDK dei provider. Il backend può chiamare tre provider diversi di Large Language Models: Groq [28], una piattaforma specializzata nell'esecuzione rapida di modelli linguistici, Anthropic Claude [29], e Google Gemini [30]. Il backend invia i dati estratti da influxDB al provider selezionato insieme

a un prompt strutturato per l'analisi, riceve la risposta e la trasforma in un documento PDF utilizzando librerie server-side per la generazione di grafici.

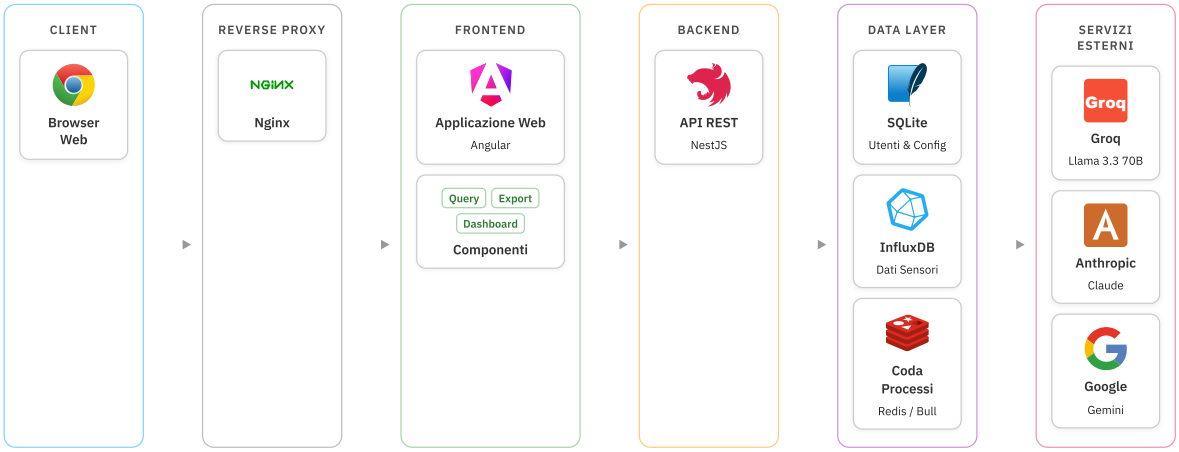


Figura 4.1: Architettura generale di AgroSense

4.1.1 Presentation Layer

Il livello di presentazione é costituito da un'applicazione web sviluppata con Angular 20.3, configurata come Single Page Application. Angular è stato scelto in quanto offre una struttura ben definita per applicazioni di medio-grandi dimensioni, utile per eventuali ulteriori implementazioni nel tempo. Inoltre Angular include già tutti gli strumenti necessari come routing, gestione HTTP e form, senza richiedere configurazioni complesse o librerie esterne.

Angular, nelle versioni più recenti, ha introdotto il concetto di standalone components, che cambia il modo in cui i componenti vengono organizzati rispetto all'approccio tradizionale. In precedenza ogni componente faceva parte di un NgModule: un contenitore che dichiarava un insieme di componenti e stabiliva quali librerie esterne potessero usare. Un componente che aveva bisogno di un campo di testo di Angular Material, per esempio, non poteva importarlo direttamente, doveva farlo il modulo che lo conteneva. Con gli standalone components questo passaggio intermedio sparisce: ogni componente importa direttamente ciò di cui ha bisogno, e leggendo il suo file si capisce subito da cosa dipende. Nella pratica questo ha ridotto il codice di configurazione e reso più semplice capire le relazioni tra le parti dell'applicazione.

Per l'interfaccia grafica è stata adottata la libreria Angular Material. Questa libreria fornisce componenti UI pronti all'uso seguendo le linee guida del Material Design. I componenti utilizzati includono Form Fields per l'input di dati, Dialog per le operazioni

modali, Tabelle per visualizzazione di risultati, Date Picker per selezionare intervalli temporali e Progress Bar per mostrare lo stato di operazioni lunghe.

La struttura dell'applicazione é organizzata in componenti specializzati. Il componente principale é QueryFormComponent, che implementa l'interfaccia per costruire query InfluxDB. Gestisce la selezione dei sensori, la configurazione degli intervalli temporali, la scelta dei metodi di aggregazione, e mostra i risultati in tabelle paginate. ExportDialogComponent è un dialog modale che permette di configurare e monitorare export asincroni di grandi dataset. ReportCustomizationDialogComponent consente di personalizzare la generazione di report analitici, permettendo di scegliere quali sezioni includere e quale LLM utilizzare. Oltre a questi ci sono diverse altre componenti di supporto per funzionalità come la cronologia delle query e la gestione dei preferiti.

L'applicazione segue pattern consolidati di Angular per garantire una gestione efficiente delle operazioni. Il pattern Reactive Forms viene utilizzato per tutti i form dell'applicazione, permettendo validazione real-time degli input e una gestione dichiarativa dello stato dei form stessi. RxJS gestisce la maggior parte delle operazioni asincrone tramite Observable, affiancandosi al pattern async/await. Un Observable é concettualmente un flusso di dati nel tempo: invece di attendere un singolo risultato, rimane attivo e notifica l'applicazione ogni volta che arriva un nuovo valore. Questo é particolarmente utile, per esempio, quando un utente modifica la selezione del measurement nel form: il componente é in ascolto sul cambiamento e aggiorna automaticamente la lista dei device disponibili senza che sia necessario gestire manualmente ogni aggiornamento. Per operazioni più semplici che producono un unico risultato, come una chiamata API, si usa invece async/await, più diretto e leggibile. La Dependency Injection di Angular viene sfruttata per iniettare servizi nei componenti, mantenendo basso l'accoppiamento tra le diverse parti dell'applicazione. La Dependency Injection è un pattern in cui un componente non crea direttamente i servizi di cui ha bisogno, ma li riceve dall'esterno. In Angular esiste un sistema centrale che si occupa di istanziare i servizi e di fornirli ai componenti che li richiedono. Un componente che ha bisogno di fare una chiamata API, per esempio, non crea da solo il servizio: dichiara nel costruttore di averne bisogno, e Angular provvede automaticamente a fornirgli l'istanza corretta.

I servizi condivisi costituiscono il layer di comunicazione tra i componenti e il backend. InfluxApiService gestisce le chiamate API relative alle query su InfluxDB, dall'elaborazione dei parametri utente alla trasformazione delle risposte. ExportService coordina il processo di export, dalla creazione del job fino al download del file completo. ReportService si occupa della generazione di report tramite LLM. AuthService infine implementa la logica di autenticazione, gestendo login, logout e verifica dello stato di autenticazione.

L'applicazione viene servita tramite Nginx configurato come reverse proxy. Nginx fornisce i file statici dell'applicazione e inoltre le richieste API al backend NestJS. Que-

sta configurazione permette al frontend e al backend di apparire sullo stesso dominio, semplificando la gestione della sicurezza e delle comunicazioni.

4.1.2 Business Logic Layer

Il livello di logica applicativa é implementato con NestJS 11, un framework Node.js progressivo e open source. La scelta di NestJS come framework backend è stata guidata da diverse considerazioni architetturali. Innanzitutto, impone una struttura modulare ben definita che facilita l'organizzazione del codice delle applicazioni, anche di grandi dimensioni. Utilizza TypeScript nativamente, mantenendo coerenza tecnica con il frontend Angular e garantendo type safety su tutto lo stack. Il framework include un sistema di Dependency Injection ispirato a quello del frontend, che permette di gestire le dipendenze tra componenti in modo dichiarativo riducendo l'accoppiamento. Inoltre, NestJS fornisce decoratori per definire controller, servizi e middleware in modo espressivo, rendendo il codice più leggibile rispetto a framework minimali come Express, un framework che fornisce solo le funzionalità di base per creare server HTTP, lasciando allo sviluppatore la scelta e la configurazione degli strumenti aggiuntivi, che richiedono una maggiore configurazione.

L'architettura del backend segue il pattern modulare di NestJS. Ogni funzionalità dell'applicazione é incapsulata in un modulo indipendente che raggruppa controller, servizi e configurazioni correlate. Questa organizzazione permette di sviluppare ogni modulo separatamente. I moduli comunicano tra di loro tramite l'esportazione e importazione esplicita di servizi, evitando dipendenze implicite difficili da tracciare.

1. **Il modulo Auth** gestisce l'autenticazione degli utenti. Implementa la registrazione di nuovi account, il processo di login con generazione di token JWT, e fornisce guard per proteggere gli endpoint che richiedono autenticazione. Durante la registrazione le password vengono hashate prima di essere salvate nel database. Il token JWT generato al login ha una validità di sette giorni e contiene l'identificativo dell'utente e l'email. Gli utenti appena registrati non sono automaticamente approvati ma devono attendere l'autorizzazione da parte dell'amministratore, che deve approvare manualmente ogni richiesta.

2. **Il modulo Users** si occupa della gestione degli account utente. Fornisce funzionalità per visualizzare la lista degli utenti registrati e approvare gli utenti in attesa. L'amministratore può accedere a questi endpoint mediante guard specifici che controllano l'effettiva carica di amministratore dell'account. Questo modulo lavora a stretto contatto con quello Auth condividendo l'entità User del database SQLite.

3. **Il modulo InfluxModule** implementa la comunicazione con il database timeseries del progetto TRACE. Fornisce servizi per estrarre le misurazioni disponibili costruire query Flux in base ai parametri ricevuti dal frontend, eseguire le query sul

server InfluxDB e processare i risultati trasformandoli in formato JSON. Questo modulo astrae la complessità delle query Flux esponendo un'interfaccia semplificata agli altri moduli che necessitano dei dati temporali.

4. **Il modulo Export** gestisce l'esportazione asincrona di grandi dataset. Quando un utente richiede l'export di dati, questo modulo crera un job nella coda Redis gestita da Bull. Un worker separato processa i job in background, estraendo i dati InfluxDB in modo incrementale e scrivendoli su file CSV. Il job traccia il proprio progresso permettendo al frontend di mostrare una barra di avanzamento all'utente. Una volta completato l'export, il file viene reso disponibile per il download. Questa architettura asincrona evita che richieste di export pesanti blocchino il server impattando altri utenti.

5. **Il modulo Report** coordina la generazione di report analitici tramite Large Language Models. Riceve dal frontend la configurazione del report desiderato estrae i dati necessari da InfluxDB, li pulisce rimuovendo valori anomali dovuti a errori hardware dei sensori, calcola statistiche descrittive e correlazioni tra variabili, invia i dati processati insieme a un prompt strutturato al provider LLM selezionato, riceve l'analisi testuale generata dal modello, crea grafici delle serie temporali e assembla tutto in un documento PDF. Il modulo supporta tre provider LLM configurabili: Groq, Anthropic Claude e Google Gemini. La generazione dei report prevede un timeout esteso a dieci minuti per permettere al modello di elaborare dataset complessi.

6. **Il modulo DatabaseModule** coordina la connessione al database SQLite locale, gestendo la persistenza dei dati applicativi degli utenti.

7. **Il modulo Experimentation** si occupa della valutazione e comparazione dei diversi provider LLM integrati.

8. **Il modulo ApiKeyModule** adibito al salvataggio e al recupero delle chiavi API degli utenti per i diversi provider LLM.

Il backend implementa pattern consolidati per garantire robustezza e manutenibilità. La Dependency Injection permette di iniettare servizi nei controller e in altri servizi dichiarando semplicemente le dipendenze nel costruttore. Questo facilita il testing permettendo di sostituire le dipendenze con mock. I guard sono middleware che verificano condizioni prima di permettere l'esecuzione di un endpoint. Il sistema utilizza Authguard per verificare che l'utente sia autenticato e AdminGuard per verificare che abbia i privilegi amministrativi. I DTO (Data Transfer Objects) definiscono la struttura dei dati in input e output degli endpoint, specificando i tipi e i campi richiesti per ogni operazione.

Tutti i moduli condividono alcune configurazioni globali. Il modulo Database configura TypeORM per connettersi al database SQLite locale dove vengono salvate gli utenti.

Il modulo principale dell'applicazione configura Bull [31], una libreria Node.js per la gestione di code di job, per connettersi a Redis, permettendo ai moduli che lo necessitano di utilizzare code asincrone quando necessario. La configurazione CORS (Cross-Origin Resource Sharing) permette richieste dal frontend specificando esplicitamente l'origine consentita per ragioni di sicurezza.

4.1.3 Data Layer

Il livello dati utilizza tre tecnologie diverse. Ogni tecnologia è stata scelta per caratteristiche specifiche che la rendono ottimale per il tipo di dati che deve gestire. Questa strategia multi-database separa le responsabilità. Ogni sistema può operare nelle condizioni per cui é ottimizzato. Il primo database é SQLite, utilizzato per memorizzare i dati applicativi degli utenti. È stato scelto perché non richiede un server database separato. Il file database viene creato localmente e non serve una configurazione complessa o amministrazione continua. Per un'applicazione, che almeno inizialmente prevede pochi utenti, SQLite offre prestazioni più che sufficienti evitando l'overhead di database più strutturati come PostgreSQL o MySQL. La libreria better-sqlite3 [32] fornisce poi un binding performante per Node.js.

Lo schema SQLite contiene due tabelle. La prima è la tabella **User**, che memorizza tutte le informazioni relative agli account: il campo id, chiave primaria autoincrementale; il campo email con vincolo di unicità; la password salvata come hash bcrypt; il campo name per il nome completo; il booleano isApproved che indica se l'amministratore ha approvato l'account; e infine createdAt, valorizzato automaticamente al momento della creazione. La seconda tabella è **api_keys**, introdotta per permettere a ciascun utente di gestire in modo indipendente le proprie chiavi di accesso ai provider LLM. Ogni riga è identificata da un id autoincrementale e mette in relazione un userId con il provider corrispondente - groq, claude o gemini - tramite un vincolo di unicità sulla coppia, in modo che ogni utente possa avere al più una chiave per provider. Il valore della chiave non viene mai salvato in chiaro: prima della persistenza viene cifrato tramite AES-256-GCM, un algoritmo di cifratura autenticata che garantisce sia la riservatezza che l'integrità del dato [33]. La tabella registra anche le date di creazione e di aggiornamento di ciascuna riga. TypeORM gestisce in maniera autonoma la creazione di entrambe le tabelle sincronizzandole con le entity definite nel codice, eliminando la necessità di scrivere manualmente SQL per le operazioni sul database.

Il secondo database é InfluxDB. E'un database specializzato per le serie temporali già utilizzato dal progetto TRACE. InfluxDB non memorizza dati in tabelle tradizionali ma organizza le informazioni in measurement. I measurement sono concettualemente simili alle tabelle ma ottimizzati per le time-series. Ogni measurement rappresenta un tipo di misurazione specifica. I dati in InfluxDB sono strutturati in modo particolare: c'è il timestamp che indica il momento esatto della misurazione, il measurement che identifica

il tipo di dato. I tags sono metadati indicizzati, per esempio device_name identifica quale sensore ha prodotto la misurazione. I fields contengono i valori numerici effettivi. Solitamente c'è un field chiamato value con il dato misurato.

I measurement presenti nel database includono:

- `device_frmpayload_data_temperature`, per i dati di temperatura dell'aria;
- `device_frmpayload_data_humidity`, per l'umidità atmosferica;
- `device_frmpayload_data_moisture`, per l'umidità del suolo;
- `device_frmpayload_data_battery`, per il livello di batteria dei sensori;
- `device_frmpayload_data_pressure`, per la pressione atmosferica;
- `device_frmpayload_data_wind_speed e device_frmpayload_data_wind_direction`, rispettivamente per velocità e direzione del vento;
- `device_frmpayload_data_ec`, per la conducibilità elettrica del suolo;
- `device_frmpayload_data_rainfall_total e device_frmpayload_data_rainfall_counter`, per i dati pluviometrici.

Il terzo componente del data layer é Redis. Viene utilizzato esclusivamente per gestire code di job asincroni. Questo avviene grazie alla libreria Bull. Redis gestisce i dati transitori relativi a questi job, funziona di fatto come broker di messaggi per coordinare l'esecuzione di task in background. Quando un utente richiede l'export di un dataset, il backend crea un job in Redis. Un processo worker separato li preleva, li esegue e ne aggiorna lo stato. Questo permette al frontend di tracciare il progresso in tempo praticamente reale. Redis mantiene quindi queste informazioni in memoria per un tempo variabile.

Questa architettura multi-database separa tre tipi di dati con requisiti molto diversi. I dati applicativi in SQLite richiedono persistenza affidabile e struttura relaizonale. Hanno però volumi contenuti. I dati time-series in InfluxDB hanno volumi enormi e necessitano di query temporali complesse. Infine i job temporanei in Redis richiedono accesso molto veloce per eseguire e tracciare le operazioni in tempo reale. Questa separazione permette ad ogni base di dati di operare nelle condizioni che meglio di adattano alle loro caratteristiche.

4.1.4 Integrazione servizi AI

Il sistema integra LLM per generare automaticamente report analitici dai dati IoT. Questa integrazione permette di trasformare serie temporali numeriche in analisi testuali. Gli

utenti non devono interpretare manualmente grafici e statistiche, ma è l'LLM a produrre un report strutturato con osservazioni e raccomandazioni.

La piattaforma supporta tre provider di modelli linguistici. Ogni provider, come vedremo più dettagliatamente nei prossimi capitoli, offre caratteristiche diverse in termini di velocità, costo e qualità dell'output. Il primo provider é Groq che utilizza il modello llama-3.3-70b-versatile [34]. Che é anche la scelta predefinita se l'utente non specifica altrimenti. Il secondo provider é Anthropic Claude con il modello claude-sonnet-4-20250514 [35]. Infine il terzo provider é Google Gemini che usa gemini-2.5-flash [36].

L'architettura del sistema permette di selezionare il provider a runtime senza modificare il codice. Un servizio centrale LLMAnalysisService coordina le chiamate ai vari provider. Il servizio riceve i dati processati e la configurazione ed identifica quale provider utilizzare. Costruisce quindi il prompt appropriato ed esegue la chiamata API. Una volta ottenuta la risposta la processa per farla vedere in seguito all'utente. I client dei vari provider vengono inizializzati lazy, ciò significa che vengono creati solo se effettivamente necessari. Questo riduce il consumo di memoria, migliorando così le prestazioni.

La comunicazione con i provider avviene tramite SDK ufficiali. Per groq si usa la libreria *groq-sdk*. Per Claude si utilizza il package *anthropic-ai/sdk* di Anthropic. Per Gemini c'é *google/generative-ai* di Google. Questi SDK gestiscono autenticazione e parsing delle risposte. Questo semplifica molto l'integrazione rispetto a chiamate HTTP raw.

Ogni provider richiede una chiave API per l'autenticazione. Queste chiavi sono personali: ogni utente le gestisce in autonomia tramite l'interfaccia della piattaforma, dove può salvarle, visualizzarle o eliminarle. Lato backend, le chiavi vengono persistite nel database SQLite cifrate con AES-256-GCM: quando il frontend richiede le chiavi salvate tramite l'endpoint dedicato, il backend le decifra in memoria prima di restituirle. La chiave viaggia poi nel corpo della richiesta di generazione del report e viene usata direttamente dal backend per inizializzare l'SDK del provider selezionato. Le variabili d'ambiente per le chiavi API rimangono disponibili come configurazione alternativa a uso degli sviluppatori, utile durante il testing e lo sviluppo locale.

Il prompt engineering è un aspetto critico del sistema. Il prompt determina la qualità dell'analisi generata. La sua struttura é divisa in due parti. La prima è il system prompt che definisce il ruolo dell'LLM. Il sistema istruisce il modello a comportarsi come un data analyst esperto specializzato in dati IoT da sensori ambientali. Deve fornire analisi professionali, concise a actionable. La seconda parte é lo user prompt che contiene i dati effettivi. Include le statistiche aggregate estratte da InfluxDB, informazioni sul periodo temporale analizzato e le correlazioni tra diverse misurazioni, quando rilevate. Gli utenti possono personalizzare la generazione del report. Possono scegliere quali sezioni includere. Per esempio solo summary, solo analisi dettagliata o solo conclusioni. Possono specificare il livello di dettaglio desiderato. Conciso per report brevi, standard per un

buon bilanciamento o dettagliato per analisi più approfondite. Possono anche fornire istruzioni custom tramite un campo di testo libero. Queste personalizzazioni vengono integrate nel prompt finale inviato all'LLM.

I modelli sono configurati con parametri specifici per ottimizzare le analisi. Il parametro temperature é impostato a 0.3 per tutti i provider. Un valore basso rende le risposte più deterministiche e focalizzate sui dati riducendo la creatività eccessiva che potrebbe portare a interpretazioni fantasiose dei dati. Il parametro maxTokens controlla la lunghezza massima della risposta. Groq usa 4000 token mentre Claude e Gemini 8000. Questa scelta è stata dettata dai vincoli di utilizzo dei vari sistemi LLM.

Il timeout per la generazione é configurato a dieci minuti. Questo é significativamente più lungo dei normali timeout. Ma è stato reso necessario in quanto l'intera pipeline di generazione, schematizzata in Figura 4.2, avviene in una singola richiesta, e con dataset complessi può richiedere tempo. Il modello deve analizzare migliaia di punti dati, questo richiede potenza computativa e quindi tempo.

La risposta dell'LLM viene parsata e strutturata. Il testo generato utilizza formattazione Markdown con sezioni numerate. Il sistema estrae automaticamente executive summary, analisi dettagliata e conclusioni pratiche tramite pattern matching sulle intestazioni. Queste sezioni vengono poi integrate nel documento PDF finale insieme ai grafici generati server-side. il PDF completo viene quindi reso disponibile per il download all'utente.

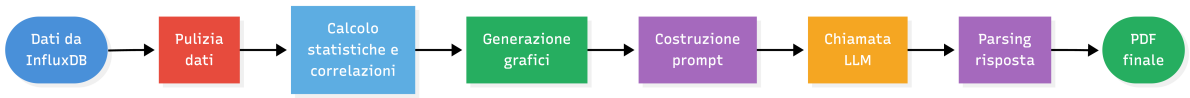


Figura 4.2: Pipeline di generazione dei report analitici.

4.2 Workflow della Piattaforma

L'utilizzo di AgroSense segue un percorso che parte dall'accesso alla piattaforma e porta, attraverso la costruzione delle query e l'esportazione di dati, fino alla generazione automatica di report analitici. Di seguito vengono descritte le fasi principali di questo percorso, con riferimento alle schermate dell'interfaccia.

4.2.1 Autenticazione

Al momento dell'accesso l'utente visualizza la schermata di login. Se l'utente non é mai stato registrato può navigare alla pagina di registrazione, dove vengono richiesti nome completo, indirizzo email e password, come mostrato in Figura 4.3. L'account creato,

tuttavia, non é immediatamente operativo. Il sistema impone che ogni nuovo utente venga approvato dall'amministratore prima di poter accedere alla piattaforma. La necessità di questa operazione viene comunque resa nota all'utente mediante un messaggio a schermo. L'account amministratore non é registrabile nell'interfaccia, ma viene inizializzato al primo avvio del backend tramite credenziali predefinite.

Dalla navbar della dashboard, l'amministratore può aprire il pannello di gestione utenti. Questo dialog espone due tab: la prima filtra i soli utenti in attesa, la seconda l'elenco completo degli account registrati con il relativo stato di approvazione, come illustrato in Figura 4.3. Per ciascuno di questi é presente un pulsante che consente di approvare l'accesso o di revocarla se l'utente é già approvato.

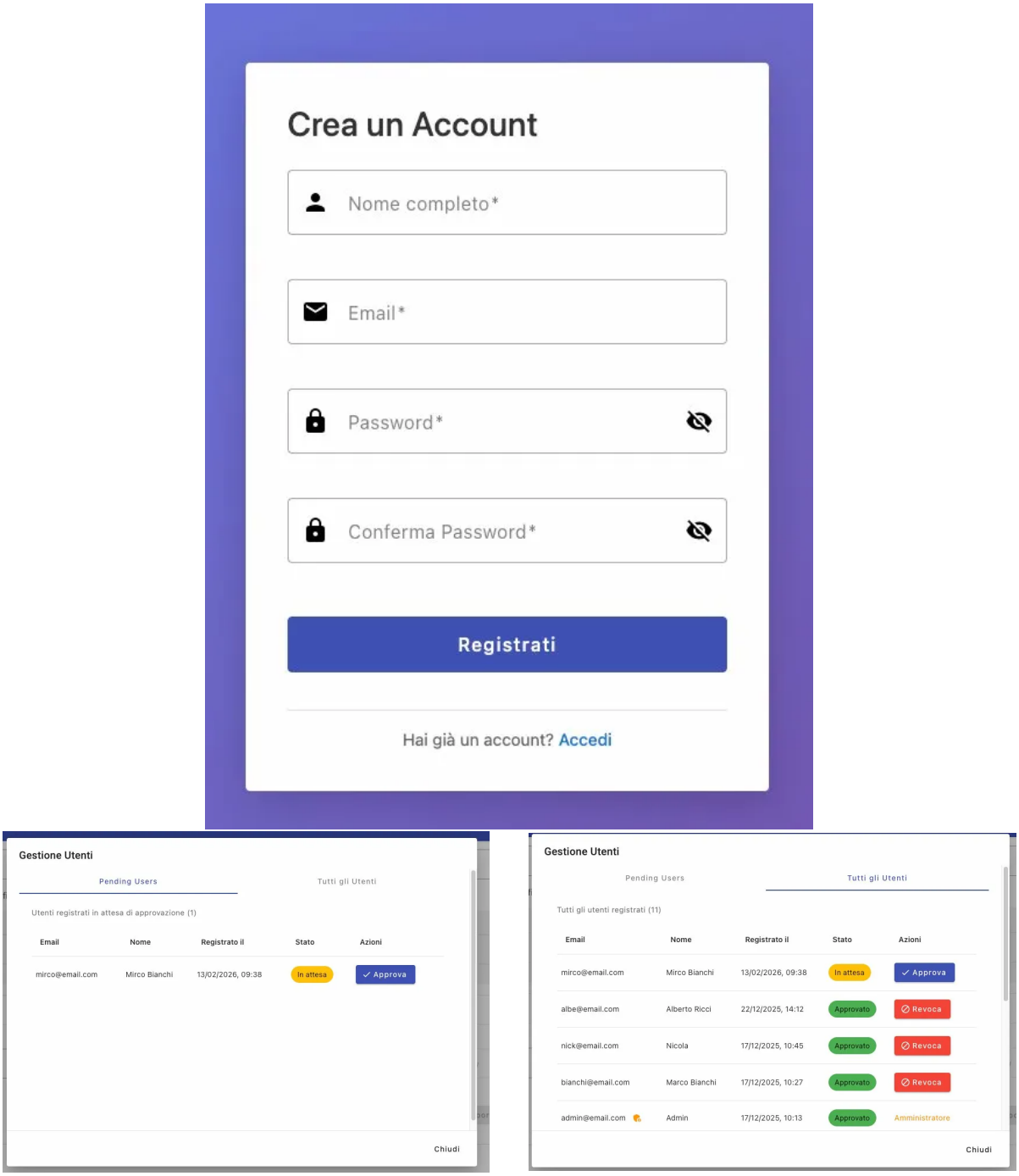


Figura 4.3: In alto: schermata di registrazione. In basso: pannello amministratore con utenti in attesa e elenco completo degli utenti con relativi stati.

4.2.2 Costruzione delle Query

Una volta autenticato, l'utente accede alla schermata principale della piattaforma. L'interfaccia mette a disposizione due modalità di interrogazione del database: una query semplice e una query avanzata.

Nella modalità semplice, il primo step prevede di selezionare il measurement. Il menù a tendina mostra tutti i measurement disponibili con nomi non tecnici come *Temperature*, *Soil Moisture* o *Pressure*. Dopo aver scelto il measurement, la piattaforma carica automaticamente i device disponibili, che possono essere selezionati singolarmente o in combinazione. La configurazione del range temporale avviene secondo due modalità: attraverso dei preset oppure attraverso una scelta personalizzata del range temporale delle date e degli orari. E' infine possibile specificare una funzione di aggregazione temporale, come *mean*, *sum*, *min*, *max*, insieme a una finestra di aggregazione.

A titolo di esempio, come illustrato in Figura 4.4, un utente ha selezionato il measurement *Humidity*, scelto le stazioni WS-1-AGRIBIO, WS-5-TRASVERSALE e WS-6-GAIANA, impostato un range temporale degli ultimi 7 giorni e richiesto la media oraria (*mean*, finestra *1h*), ottenendo una tabella con l'andamento dell'umidità atmosferica nell'ultima settimana.

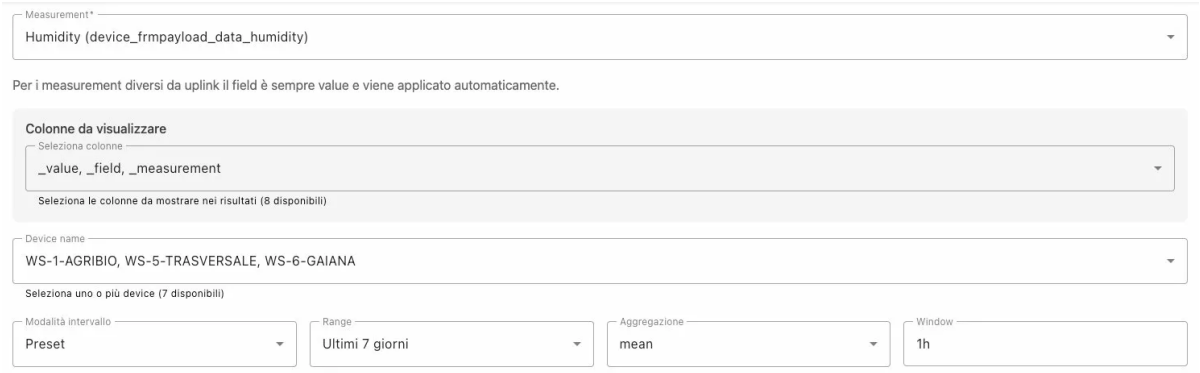


Figura 4.4: Schermata principale con measurement, device e range temporale configurati.

Se si vogliono combinare più misurazioni é disponibile la modalità avanzata delle query. Tramite il click di un pulsante dedicato si apre la finestra di dialog. In tale finestra é possibile aggiungere più measurement in un'unica configurazione assegnando ad ognuno un metodo di aggregazione indipendente. Il range temporale e la finestra di aggregazione, invece, sono globali e si applicano allo stesso modo su tutti i measurement selezionati. Quando la modalità avanzata é attiva, i campi delle query semplici sono disabilitati e un banner ne informa l'utente.

Un esempio é mostrato Figura 4.5: la prima aggregazione interroga *Humidity* con metodo *mean* sulle stazioni WS-7-FIORENTINA e WS-8-VIA LARGA, mentre la seconda interroga *Pressure*, con metodo *count* sulle stazioni WS-5-TRASVERSALE e WS-6-GAIANA, entrambe su una finestra oraria degli ultimi 7 giorni.

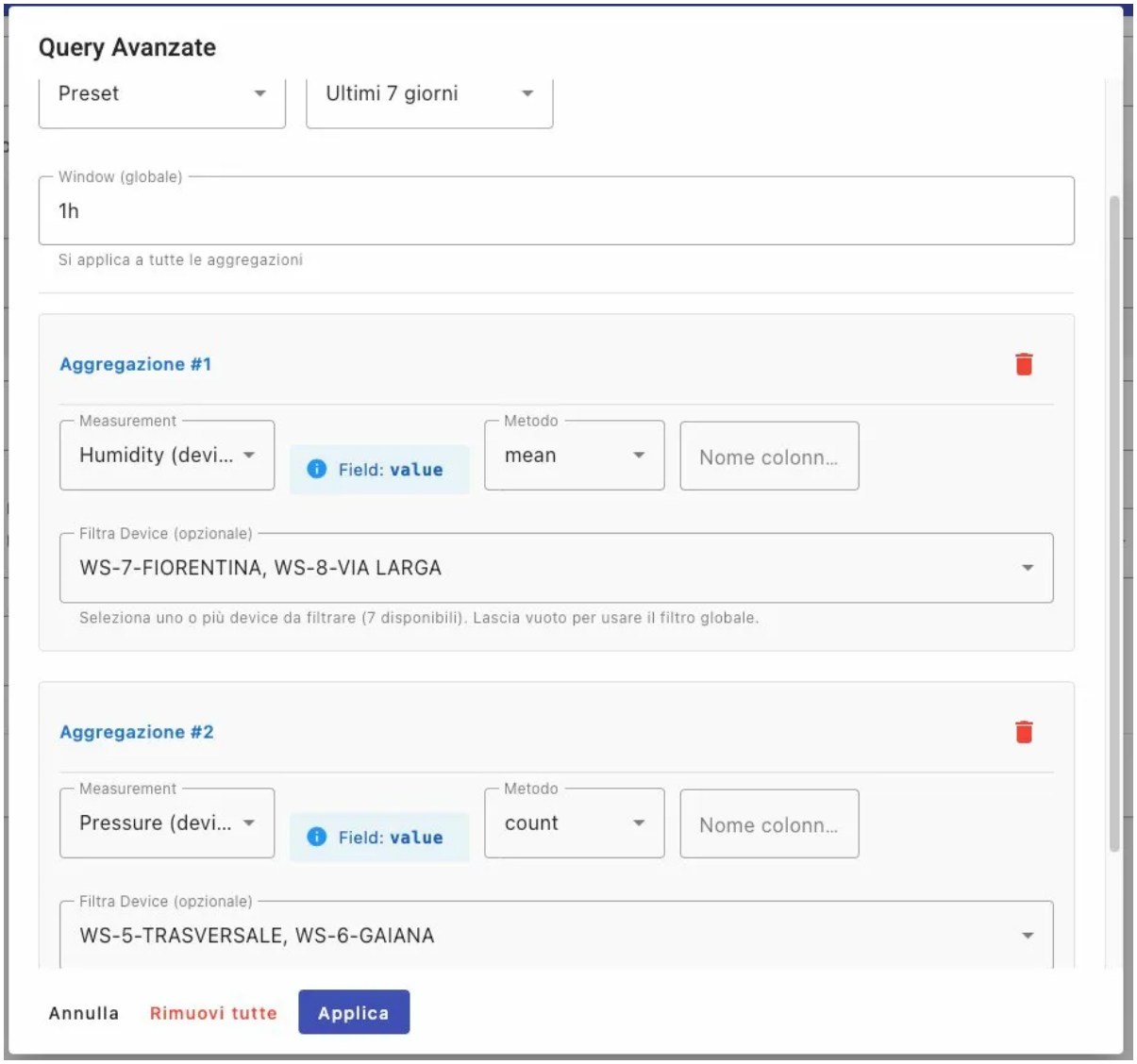


Figura 4.5: Dialog per la configurazione di query avanzate con measurement multipli, ciascuno con la propria aggregazione.

I risultati, infine, vengono visualizzati in una tabella paginata nella parte inferiore della schermata, accompagnata dal tempo di esecuzione e dal numero di righe restituite, come da Figura 4.6.

Risultati trovati: 398 righe in 181 ms

_time	_value	_field	_measurement	device_name	dev_eui	application_name	f_port
2026-02-06T10:00:00Z	97.5	value	device_frmpayload_data_humidity	WS-6-GAIANA	24e124454e353600	trace-application	85
2026-02-06T11:00:00Z	97.5	value	device_frmpayload_data_humidity	WS-6-GAIANA	24e124454e353600	trace-application	85
2026-02-06T12:00:00Z	97.5	value	device_frmpayload_data_humidity	WS-6-GAIANA	24e124454e353600	trace-application	85
2026-02-06T13:00:00Z	96.5	value	device_frmpayload_data_humidity	WS-6-GAIANA	24e124454e353600	trace-application	85
2026-02-06T14:00:00Z	94.5	value	device_frmpayload_data_humidity	WS-6-GAIANA	24e124454e353600	trace-application	85
2026-02-06T15:00:00Z	90.5	value	device_frmpayload_data_humidity	WS-6-GAIANA	24e124454e353600	trace-application	85
2026-02-06T16:00:00Z	87	value	device_frmpayload_data_humidity	WS-6-GAIANA	24e124454e353600	trace-application	85
2026-02-06T17:00:00Z	88	value	device_frmpayload_data_humidity	WS-6-GAIANA	24e124454e353600	trace-application	85
2026-02-06T18:00:00Z	92.5	value	device_frmpayload_data_humidity	WS-6-GAIANA	24e124454e353600	trace-application	85
2026-02-06T19:00:00Z	94.75	value	device_frmpayload_data_humidity	WS-6-GAIANA	24e124454e353600	trace-application	85

Items per page: 10 1 – 10 of 398

Figura 4.6: Tabella di visualizzazione dei risultati di una query.

4.2.3 Esportazione dei Dati

Per eseguire l'esportazione dei dati la piattaforma mette a disposizione la funzione *Export Completo* tramite l'omonimo tasto, al cui click si apre una finestra dialog dove l'utente può personalizzare l'operazione. In primis può decidere di esportare l'intero database oppure solo i measurement deisderati, così come per le colonne. Infine, in via opzionale, é chiamato a scegliere il range temporale dell'esportazione, che se non diversamente specificato, é impostato dalla prima registrazione avvenuta fino all'ultima. Avviato l'export si apre una modale che mostra lo stato di avanzamento dell'operazione, che una volta ultimata permette di scaricare il file risultante.

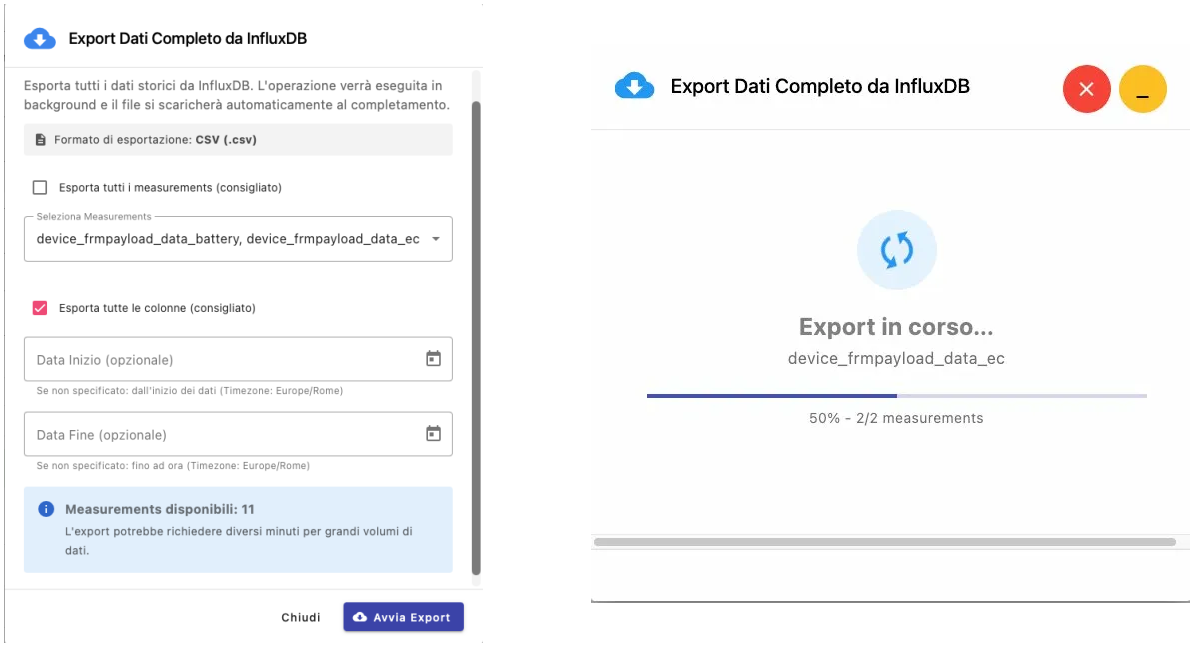


Figura 4.7: Da sinistra: schermata di selezione opzione di export e visualizzazione dell'avanzamento dell'operazione.

4.2.4 Generazione di Report Analitici

Riprendendo l'esempio precedente, una volta ottenuti i risultati della query avanzata su **Umidità** e **Pressione**, come mostrato in Figura 4.8, nella schermata si attiva il pulsante *Genera Report LLM*. Premendolo si apre una finestra di dialog, mostrata in Figura 4.9, dove l'utente configura la generazione prima di avviarla. Può scegliere quali sezioni includere tra sommario esecutivo, analisi dettagliata e conclusioni pratiche, decidere se aggiungere statistiche descrittive e analisi delle correlazioni tra variabili, e selezionare il provider LLM da utilizzare tra Groq, Anthropic Claude e Google Gemini, andando a specificare per ognuno la chiave API da utilizzare, se l'utente ha già salvato una chiave questa gli viene automaticamente proposta. L'utente é quindi in grado di controllare le proprie chiavi nella apposita finestra, dove può interagire mostrando l'intera chiave o eliminandola. È inoltre possibile specificare il livello di dettaglio - conciso, standard o dettagliato - e fornire istruzioni personalizzate in testo libero da passare direttamente al modello. Avviata la generazione, l'utente é in grado di osservarne l'avanzamento grazie a una apposita finestra. Al termine della generazione, il PDF viene scaricato automaticamente, contenendo le sezioni selezionate applicate ai dati della query eseguita.

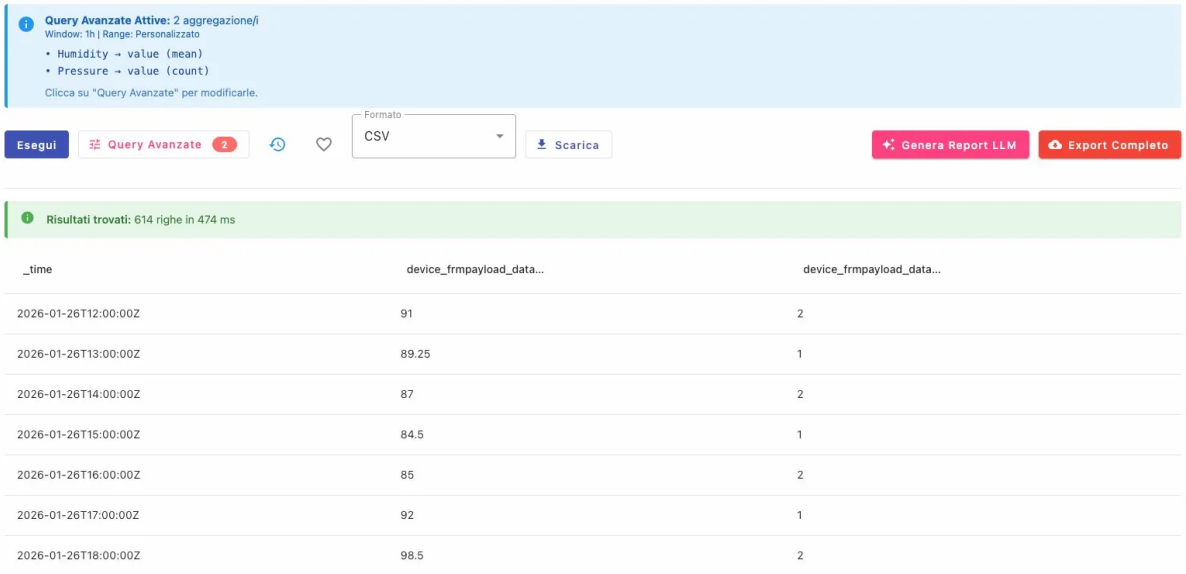


Figura 4.8: Tabella dei risultati popolata dalla query avanzata: Umidità e Pressione. I pulsanti *Genera Report LLM* ed *Export Completo* risultano attivi.

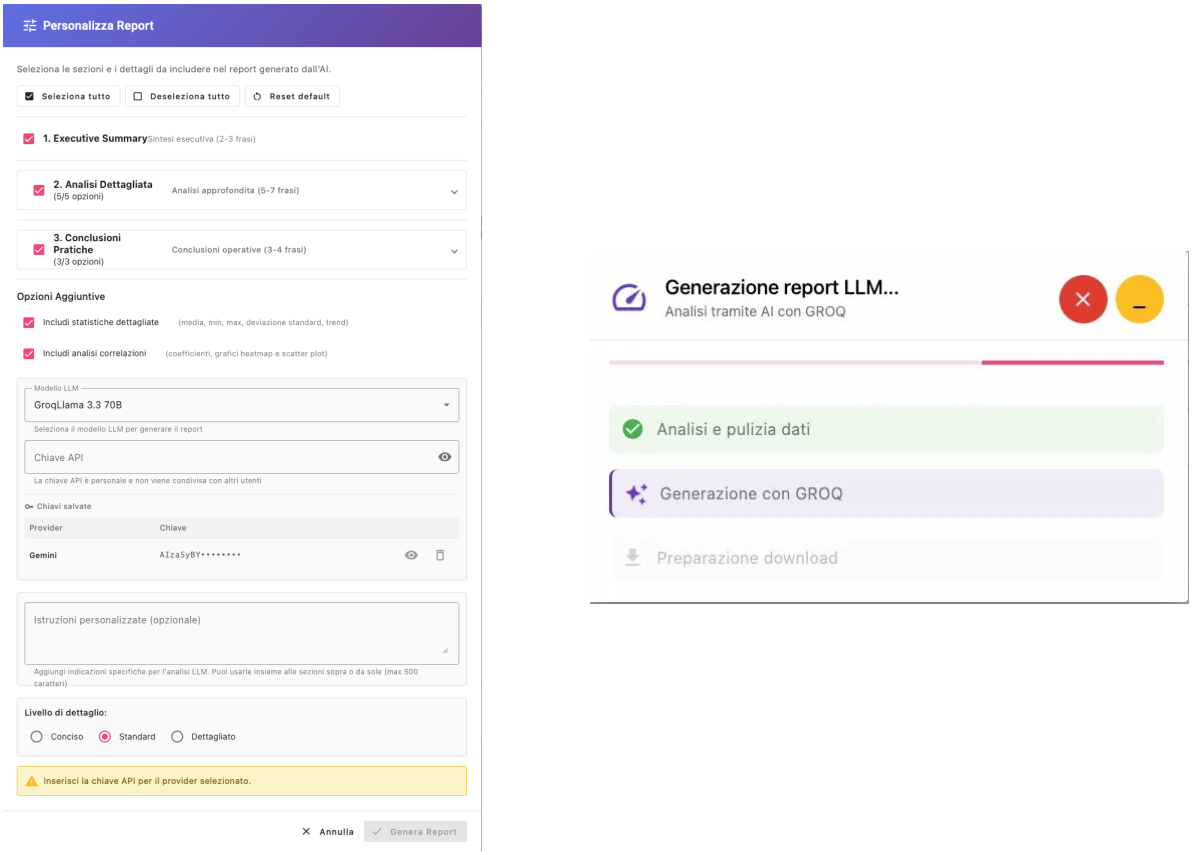


Figura 4.9: Da sinistra: schermata di personalizzazione del report LLM e stato di avanzamento della generazione.

Capitolo 5

Implementazione

Il presente capitolo descrive le scelte implementative adottate nella realizzazione di AgroSense, analizzando in dettaglio ciascuna componente del sistema. Il capitolo procede dal backend verso il frontend, approfondendo i meccanismi che abilitano le funzionalità descritte nel capitolo precedente: la generazione dinamica delle query flux, la pipeline di data cleaning, il processing asincrono degli export e l'integrazione con i provider LLM. Il capitolo si conclude con la containerizzazione Docker che ha permesso il deployment dell'intero sistema.

5.1 Backend: NestJS e Architettura Modulare

Il backend di AgroSense é implementato con NestJS 11 [25], che impone una struttura modulare ben definita fin dalla fase di progettazione. L'intero progetto é organizzato come monorepo gestito da Nx [37] con pnpm. Un monorepo è una configurazione in cui frontend e backend risiedono nello stesso repository -rispettivamente in *apps/InfluxDashboard/* e *api/* - condividendo un unico file di lock delle dipendenze e le configurazioni di build, ma mantenendo artefatti di deployment completamente indipendenti. Nx aggiunge a questa struttura un sistema di build incrementale con cache: solo i progetti effettivamente modificati vengono ricompilati, riducendo i tempi di build durante lo sviluppo. pnpm è il package manager scelto per la gestione delle dipendenze perché, a differenza di npm, utilizza un approccio basato su hard link che evita la duplicazione dei pacchetti condivisi tra i due progetti. La scelta del monorepo é motivata principalmente da ragioni operative: avere frontend e backend nello stesso repository semplifica la gestione del ciclo di sviluppo, consente di eseguire build delle due parti con comandi unificati tramite Nx, e produce un unico artefatto versionabile.

5.1.1 Struttura modulare e Dependency Injection

Come descritto nel capitolo precedente, il backend si articola in otto moduli funzionali. Ciascuno é implementato come una classe TypeScript decorata con *Module()*, che dichiara esplicitamente fino a quattro elementi: *imports* per le dipendenze da altri moduli, *controllers* per gli endpoint HTTP, *providers* per i servizi interni, ed *exports* per i provider resi visibili e disponibili all'esterno. Questa dichiarazione esplicita rende il grafo delle dipendenze verificabile staticamente e impedisce l'emergere di accoppiamenti impliciti.

Il modulo radice *AppModule* aggrega tutti i moduli funzionali e configura i servizi globali, tra cui *BullModule.forRoot()* che inizializza la connessione a Redis leggendo host e porta dalle variabili d'ambiente.

Listing 5.1: app.module.ts: modulo radice dell'applicazione

```
import { Module } from '@nestjs/common';
import { BullModule } from '@nestjs/bull';
import { DatabaseModule } from '../database/database.module';
import { AuthModule } from '../auth/auth.module';
import { UsersModule } from '../users/users.module';
import { InfluxModule } from '../influx/influx.module';
import { ReportModule } from '../report/report.module';
import { ExportModule } from '../export/export.module';
import { ExperimentationModule } from '../llm-experimentation/
    experimentation.module';
import \{ ApiKeyModule \} from '../␣api␣-␣keys␣/␣api␣-␣key␣.␣
    module␣';

@Module({
  imports: [
    BullModule.forRoot({
      redis: {
        host: process.env.REDIS_HOST || 'localhost',
        port: parseInt(process.env.REDIS_PORT || '6379'),
      },
    }),
    DatabaseModule,
    AuthModule,
    UsersModule,
    InfluxModule,
    ReportModule,
    ExportModule,
    ExperimentationModule,
    ApiKeyModule,
  ],
```

```
29    controllers: [AppController],
30    providers: [AppService],
31  })
32  export class AppModule {}
```

La visibilità dei servizi tra moduli é controllata esclusivamente tramite exports e imports: un servizio non é accessibile al di fuori del proprio modulo a meno che non venga esplicitamente esportato. Questo meccanismo impedisce dipendenze trasversali non dichiarate.

Listing 5.2: report.module.ts: imports ed exports espliciti

```
1  @Module({
2    imports: [InfluxModule],
3    controllers: [ReportController],
4    providers: [
5      ReportGeneratorService,
6      LLMAnalysisService,
7      ChartGeneratorService,
8      PdfGeneratorService,
9      DataCleaningService,
10     CorrelationService,
11   ],
12   exports: [ReportGeneratorService, LLMAnalysisService],
13 })
14 export class ReportModule {}
```

Il meccanismo che risolve queste dipendenze a runtime é la Dependency Injection. NestJS mantiene un container IoC (Inversion of Control) che istanzia i provider e li inietta nei costruttori dei servizi che li richiedono. La dichiarazione avviene esclusivamente nel costruttore: il framework si occupa della creazione e del ciclo di vita degli oggetti.

Listing 5.3: Costruttore di ReportGeneratorService: DI in azione

```
1  @Injectable()
2  export class ReportGeneratorService {
3    private readonly logger = new Logger(ReportGeneratorService.
       name);
4
5    constructor(
6      private readonly llmAnalysisService: LLMAnalysisService,
7      private readonly chartGeneratorService: ChartGeneratorService
         ,
8      private readonly pdfGeneratorService: PdfGeneratorService,
9      private readonly dataCleaningService: DataCleaningService,
```

```
10      private readonly correlationService: CorrelationService,
11    ) {}
12  }
```

Questo approccio produce due benefici importanti. Il basso accoppiamento che deriva dal fatto che ogni servizio delega al container la risoluzione delle proprie dipendenze, senza istanziarle direttamente. La testabilità migliora perché qualsiasi dipendenza può essere sostituita con un mock nel modulo di test senza modificare il codice del servizio stesso.

5.1.2 Autenticazione e autorizzazione

Il sistema di autenticazione si basa su JSON Web Token uno standard per l'autenticazione stateless in cui il server non mantiene sessioni: ogni richiesta é autonomamente verificabile tramite la firma del token. Al momento del login, AuthService verifica le credenziali confrontando la password fornita con l'hash bcrypt memorizzato in SQLite tramite *brcypt.compare()*. Se la verifica ha successo e l'account risulta approvato, viene generato un token JWT firmato con la chiave JWT_SECRET letta dalle variabili di ambiente, con scadenza configurata in sette giorni. Il payload del token contiene esclusivamente l'identificativo numerico dell'utente e la sua email, minimizzando le informazioni esposte.

La protezione degli endpoint é implementata tramite due guard distinti. AuthGuard verifica che ogni richiesta contenga un token Bearer valido nell'header Authorization, lo valida crittograficamente e, in caso di successo, inietta il payload decodificato nell'oggetto request rendendolo disponibile ai controller downstream. AdminGuard estende questa logica verificando che l'email contenuta nel payload corrisponda all'account amministratore: gli endpoint di gestione utenti sono così inaccessibili anche a utenti autenticati regolari.

Listing 5.4: AuthGuard: validazione token e iniezione payload nella request

```
1  @Injectable()
2  export class AuthGuard implements CanActivate {
3    constructor(private jwtService: JwtService) {}
4
5    async canActivate(context: ExecutionContext): Promise<boolean>
       {
6      const request = context.switchToHttp().getRequest();
7      const token = this.extractTokenFromHeader(request);
8      if (!token) throw new UnauthorizedException('Token non
         presente');
9      try {
```

```
10        const payload = await this.jwtService.verifyAsync(token, {
11          secret: process.env.JWT_SECRET,
12        });
13        request['user'] = payload;
14      } catch {
15        throw new UnauthorizedException('Token non valido');
16      }
17      return true;
18    }
19
20    private extractTokenFromHeader(request: Request): string |
          undefined {
21      const [type, token] = request.headers.authorization?.split(' '
          ) ?? [];
22      return type === 'Bearer' ? token : undefined;
23    }
24  }
```

Un aspetto rilevante del sistema riguarda il workflow di approvazione. Gli utenti appena registrati hanno il campo *isApproved* impostato a *false* e il login viene bloccato da AuthService prima ancora della generazione del token, restituendo *401 Unauthorized* con messaggio esplicito. Solo dopo l'approvazione manuale da parte dell'amministratore tramite gli endpoint del UsersModule l'account diventa operativo. L'account amministratore è l'unica eccezione: viene inizializzato automaticamente al primo avvio del backend tramite *AdminSeedService* come *isApproved* già impostato a *true*.

5.1.3 Integrazione InfluxDB e generazione query Flux

InfluxService è il componente che astrae la comunicazione con il database time-series del progetto TRACE. All'inizializzazione, il servizio configura il client ufficiale *influxdata/influxdb-client* leggendo URL, token e organizzazione dalle variabili d'ambiente, ottenendo un'istanza di QueryApi utilizzata per tutte le operazioni successive. Prima di eseguire qualsiasi query, il servizio espone metodi di introspezione dello schema che sfruttano il pacchetto *influxdata/influxdb/schema* di Flux per elencare dinamicamente i measurement disponibili, i field key e i tag value presenti nel bucket. E' questo meccanismo che permette al frontend di popolare i menu a tendina in modo autonomo, senza che i nomi dei measurement siano codificati staticamente nell'applicazione. Il cuore del modulo è il metodo queryGeneric(), che riceve dal frontend un oggetto GenericQueryDto e lo traduce in una query Flux eseguibile. La costruzione avviene in modo programmatico concatenando le clausole della pipeline Flux in base ai parametri ricevuti. La gestione del range temporale distingue diversi casi: se sono presenti sia *start* che *stop* vengono usati direttamente nel parametro *time* della query, altrimenti se é selezionato uno dei

preset disponibili, viene usato direttamente quello nella clausola *range*. I filtri sui tag vengono aggiunti come clausole filter() con supporto per gli operatori di uguaglianza e disuguaglianza. L'aggregazione temporale, quando presente, viene applicata tramite aggregateWindow() con la funzione e la finestra specificate dall'utente. Per la modalità avanzata, in cui l'utente combina più measurement, il metodo *queryMultiMeasurement()* costruisce sub-query Flux separate per ciascun measurement, ciascuna con il proprio filtro su device, metodo di aggregazione e nome della colonna personalizzato assegnato tramite *map()*. I risultati delle sub-query vengono poi riorganizzati in memoria per timestamp, producendo un array di righe con colonne separate per measurement, con valori null espliciti dove un timestamp non ha un corrispondente valore per una certa misurazione.

Il seguente esempio mostra una query Flux generata per una richiesta avanzata con due measurement, umidità e temperatura, aggregati come media oraria sugli ultimi sette giorni:

Listing 5.5: Esempio di query Flux generata per modalità avanzata multi-measurement

```
from(bucket: "trace_project")
  |> range(start: -7d)
  |> filter(fn: (r) => r._measurement ==
       "device_frmpayload_data_humidity")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.device_name =~ /^(WS-1-AGRIBIO)$/)
  |> aggregateWindow(every: 1h, fn: mean, createEmpty: false)
  |> map(fn: (r) => ({ r with _value: r._value,
       _field: "Humidity" }))
  |> yield(name: "Humidity")

from(bucket: "trace_project")
  |> range(start: -7d)
  |> filter(fn: (r) => r._measurement ==
       "device_frmpayload_data_temperature")
  |> filter(fn: (r) => r._field == "value")
  |> aggregateWindow(every: 1h, fn: mean, createEmpty: false)
  |> map(fn: (r) => ({ r with _value: r._value,
       _field: "Temperature" }))
  |> yield(name: "Temperature")
```

La risposta di InfluxDB viene raccolta tramite *collectRows()*. Per la modalità avanzata i risultati vengono riorganizzati per produrre righe con colonne affiancate per la visualizzazione sul frontend.

5.1.4 Pipeline di Data Cleaning

I dati provenienti da sensori IoT in campo aperto presentano inevitabilmente anomalie di diversa natura: valori null dovuti a interruzioni di trasmissione, letture fisicamente impossibili possibilmente causate da malfunzionamenti hardware, e outlier statistici generati da condizioni ambientali transitorie o da interferenze. La pipeline di data cleaning di AgroSense affronta queste problematiche in sequenza prima della generazione di ogni report, operando sui dati grezzi estratti da InfluxDB prima che vengano aggregati e analizzati dall'LLM. Il componente responsabile é `DataCleaningService`, che organizza il processo in tre fasi distinte. Nella prima fase i dati vengono raggruppati per il measurement, consentendo di applicare regole di validazione specifiche. Nella seconda fase ogni gruppo viene validato: vengono rimossi i valori null e quelli che cadono fuori range non ammissibili definiti staticamente per ciascun measurement. Nella terza fase, sui valori sopravvissuti alla validazione, viene applicato un algoritmo di rilevamento di outlier. Le regole di validazione per range fisico coprono diversi casi come: la temperatura é vincolata tra -50 e 70°C, l'umidità tra 0 e 100%, la pressione tra 800 e 1100 hPa, la batteria tra lo 0 e il 100%. Per il rilevamento di outlier statistici il sistema supporta tre metodi: Z-score standard, IQR (Interquartile Range) e Modified Z-score. Il metodo predefinito é quest'ultimo che utilizza la mediana e la MAD (Median Absolute Deviation) al posto di media e deviazione standard. Questa scelta é motivata dalla maggiore robustezza del metodo in presenza di distribuzioni asimmetriche, tipiche delle serie temporali agricole dove picchi stagionali o eventi climatici estremi possono distorcere significativamente la media. Il servizio di data cleaning implementa inoltre due meccanismi di sicurezza per evitare una pulizia eccessivamente aggressiva. Il primo è una soglia di retention range globale: se il processo rimuoverebbe più del 50% dei dati totali, il cleaning viene annullato e i dati originali vengono passati all'LLM con un warning esplicito. Il secondo è una soglia per measurement: se un singolo measurement perde più del 40% dei propri valori viene generato un avvertimento e la generazione bloccata, come evidenziato dalla Figura 5.1. Le statistiche di cleaning - valori originali, rimossi per categoria e percentuali di retention - vengono incluse nel report PDF finale nella sezione *Qualità dei Dati e Pulizia*, rendendo così il processo trasparente.

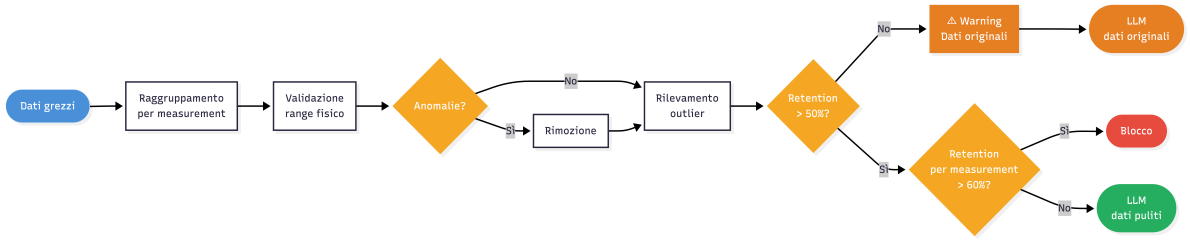


Figura 5.1: Pipeline di data cleaning di AgroSense.

5.1.5 Sistema di export asincrono

L'esportazione di grandi dataset da InfluxDB è un'operazione che può richiedere diversi minuti, durante i quali il server non deve restare bloccato ad attendere il completamento prima di poter gestire altre richieste. Per questo motivo il sistema di export é implementato con un'architettura asincrona basata su Bull, una libreria Node.js per la gestione di code di job persistite su Redis. Il flusso inizia quando un utente avvia un export dal frontend. `ExportService` genera un UUID univoco per il job, determina la lista dei measurement da esportare (se non specificati dall'utente vengono recuperati dinamicamente da InfluxDB), e inserisce il job nella coda Bull con i parametri della richiesta: masurement, range temporale, formato e colonne selezionate.

Il job viene configurato con tre tentativi automatici in caso di errore, con un backoff di 5 secondi tra un tentativo e l'altro. `ExportWorker` rimane in ascolto sulla coda e processa i job in background. Per ciascun measurement nella lista, il worker invoca `InfluxStreamService` che costruisce una query Flux e restituisce i dati come Node.js Readable. I dati vengono trasformati in righe CSV tramite un Trasform stream e scritti direttamente su disco in */tmp*. Ad ogni measurement completato, il worker aggiorna il campo progress del job su Redis e persiste il measurement corrente in *job.data.currentMeasurement*, permettendo al frontend di mostrare una barra di avanzamento precisa.

La cancellazione di un job in corso é gestita tramite un flag cancelled scritto direttamente nel payload del job su redis. Il worker controlla questo flag all'inizio dell'elaborazione di ogni measurement: se trova il flag attivo interrompe il processing, elimina il file parziale e termina il job.

Quando l'export è completato, il file rimane disponibile per il download tramite l'endpoint dedicato.

5.1.6 Integrazione multi-provider LLM

Il modulo di analisi LLM é progettato per supportare tre provider distinti - Groq, Anthropic Claude e Google Gemini - in modo intercambiabile, senza che il codice chiamante debba conoscere quale provider sia attivo. La selezione avviene a runtime tramite il parametro *llmProvider* passato nel corpo della richiesta, che corrisponde alla scelta effettuata dall'utente nella modale di configurazione del report. La chiave API arriva allo stesso modo nel campo *apiKey* del corpo della richiesta: è il frontend che, all'apertura della modale, interroga l'endpoint */api/api-keys* per recuperare le chiavi salvate, che il backend restituisce già decifrate, e pre-popola automaticamente il campo con quella corrispondente al provider selezionato. Il backend riceve quindi la chiave in chiaro nel DTO della richiesta e la passa direttamente a *getLLMConfig()*, che la usa per inizializzare la chiamata all'SDK.

La configurazione di ciascun provider é centralizzata in `llm.config.ts` che definisce modello, numero massimo di token e temperatura per ognuno. La temperatura é impostata a 0.3 per tutti e tre i provider, un valore basso per favorire risposte analitiche e deterministiche piuttosto che creative. Il limite di token diverge: Groq é configurato a 4000 mentre Claude e Gemini dispongono di 8000, riflettendo le capacità dei rispettivi modelli. I modelli utilizzati sono llama-3.3-70b-versatile per Groq, claude-sonnet-4-20250514 per Anthropic e gemini-2.5-flash per Google.

Listing 5.6: llm.config.ts: configurazione centralizzata dei provider

```
export const LLM_PROVIDERS_CONFIG = {
  [LLMProvider.GROQ]: {
    model: 'llama-3.3-70b-versatile',
    maxTokens: 4000,
    temperature: 0.3,
  },
  [LLMProvider.CLAUDE]: {
    model: 'claude-sonnet-4-20250514',
    maxTokens: 8000,
    temperature: 0.3,
  },
  [LLMProvider.GEMINI]: {
    model: 'gemini-2.5-flash',
    maxTokens: 8000,
    temperature: 0.3,
  },
};
```

I client SDK dei tre provider sono inizializzati *lazy* all'interno di `LLMAnalysisService`: nessun client viene istanziato al momento della creazione del servizio, ma solo alla prima invocazione che richiede quel provider specifico. Questa scelta evita di istanziare connessioni non necessarie ed é implementata tramite tre metodi privati *getGroqClient()*, *getClaudeClient()* e *getGeminiClient()* che verificano l'esistenza dell'istanza prima di crearla. Il metodo pubblico analyzeData() riceve i dati processati e delega al metodo privato corrispondente il provider selezionato tramite uno switch. Ogni metodo invoca l'SDK del rispettivo provider con la stessa struttura di prompt: viene definito il ruolo da analista IoT agricolo che l'LLM deve ricoprire e vengono passate le statistiche, le serie temporali e le opzioni di personalizzazione del report. Nonostante le API dei tre provider abbiano interfacce diverse - Groq usa *chat.completions.create()*, Claude usa *messages.create()* con il system prompt come parametro separato, Gemini usa *generateContent()* - tutti e tre restituiscono una stringa di testo che viene elaborata da *parseAnalysis()* per estrarre le sezioni strutturate del report.

5.2 Frontend: Angular e Material Design

5.2.1 Architettura standalone components e routing

Il routing dell'applicazione é definito in `app.routes.ts` con cinque entry. La rotta radice rediriziona a */login*, garantendo che gli utenti non autenticati non accedano mai direttamente alla dashboard. Le rotte */login* e */register* sono pubbliche e non richiedono alcuna guard. La rotta /dashboard é protetta da `authGuard`, una guard funzionale che verifica la presenza del token JWT in `localStorage` prima di consentire l'acceso al componente. Qualsiasi percorso non riconosciuto viene reindirizzato a */login* tramite il pattern wildcard "*".

Listing 5.7: app.routes.ts: definizione delle rotte con protezione tramite authGuard

```
export const appRoutes: Routes = [
  { path: '', redirectTo: '/login', pathMatch: 'full' },
  { path: 'login',    component: LoginComponent },
  { path: 'register', component: RegisterComponent },
  {
    path: 'dashboard',
    component: DashboardComponent,
    canActivate: [authGuard]
  },
  { path: '**', redirectTo: '/login' }
];
```

Il componente radice AppComponent ospita esclusivamente il RouterOutlet e due componenti floating persistenti: `ReportProgressFloatingButton` e `ExportFlotingButton`, posizionati in basso a destra tramite CSS position: *fixed*. Questa scelta consente di monitorare lo stato di report ed export anche navigando tra le schermate, senza interrompere le operazioni in corso.

La guard authguard é implementata come funzione anziché come classe, seguendo il pattern introdotto da Angular. Questo elimina la necessità di decorare la guard con *Injectable()* e dichiararla come provider, riducendo il boilerplate e mantenendo la stessa espressività.

5.2.2 QueryFormComponent e Reactive Forms

`QueryFormComponent` è il componente centrale dell'applicazione e, data la sua complessità, inietta otto dipendenze tramite la funzione *inject()*: `InfluxApiService`, `ExportService`, `ReportService`, `QueryHistoryService`, `ReportStatusService` e `MatDialog` per l'apertura delle dialog Material, a cui si aggiungono due dipendenze Angular core: `DestroyRef`,

usato in combinazione con l'operatore *takeUntilDestroyed*() per gestire il ciclo di vita delle subscription RxJS senza dover implementare manualmente *ngOnDestroy*(), e `ChangeDetectorRef`, necessario per forzare la rilevazione dei cambiamenti in alcuni percorsi di aggiornamento asincrono.

La gestione del form é implementata con Reactive Forms tramite un `FormGroup` che raggruppa i `FormControl` per masurement, range temporale, aggregazione e filtri. Questo approccio rende lo stato del form interamente osservabile tramite stream RxJS: quando l'utente seleziona un measurement, il componente sottoscrive gli Observable restituiti da *listFieldKeys()* e *listTagValues()* per popolare dinamicamente i menu di selezione successivi, garantendo che le opzioni disponibili riflettano sempre lo schema reale dei dati presenti in InfluxDB. l'unica codifica statica implementata riguarda il mapping per le etichette dei measurement per poterli visualizzare con nomi più esplicativi.

La modalità avanzata, che permette di combinare più measurement con aggregazioni indipendenti, é gestita tramite un array di `FieldAggregation` mantenuto come `signal` Angular. L'utilizzo di signal consente al template di reagire alle modifiche dell'array senza dover invocare manualmente *ChangeDetectorRef.detectChanges()*, che rimane invece necessario nei percorsi di aggiornamento asincrono non coperti dai signal stessi. Questo giustifica la presenza di entrambe le dipendenze nel costruttore del componente.

5.2.3 Servizi e gestione operazioni asincrone

Tutti i servizi sono dichiarati come `providedIn:'root'`, rendendoli singleton condivisi tra tutti i componenti senza necessità di dichiararli in un modulo.

`AuthService` gestisce il ciclo di vita dell'autenticazione tramite due signal: `isAuthenticated` e `currentUser`, aggiornati al momento del login insieme alla persistenza del token JWT in `localStorage`. Il computed signal `isAdmin` deriva automaticamente da `currentUser` verificando se l'email corrisponde all'account amministratore, centralizzando questa logica nel servizio ed evitando duplicazioni nei componenti che lo consumano. Il token JWT viene allegato automaticamente a ogni richiesta HTTP tramite authInterceptor, una `HttpInterceptorFn` funzionale che clona la request aggiungendo l'header `Authorization: Bearer<token>`, con esclusione esplicita degli endpoint di login e registrazione che non richiedono autenticazione.

`ReportService` gestisce la generazione dei report PDF. La cancellazione é implementata come meccanismo distribuito tra due servizi: `ReportStatusService` possiede il `Subject<void>` e ne espone l'Observable tramite *getCancellationSignal()*, separando la gestione dello stato UI dalla logica HTTP. `ReportService` riceve questo Observable come parametro opzionale e lo applica tramite *takeUntil()* alla richiesta in corso: quando l'utente annulla, il Subject emette un valore e viene completato, e l'operatore rileva l'emissione causando l'abbandono della connessione.

Listing 5.8: Meccanismo di cancellazione distribuito tra ReportStatusService e ReportService

```
// ReportStatusService: ownership del Subject
private cancelSubject = new Subject<void>();

getCancellationSignal(): Observable<void> {
  return this.cancelSubject.asObservable();
}

cancel() {
  this.currentStatus.set({ ...current,
    stage: ReportGenerationStage.CANCELLED });
  this.cancelSubject.next();
  this.cancelSubject.complete();
}

// ReportService: applicazione del takeUntil sulla HTTP request
async generateReport(
  config: ReportConfigDto,
  cancellation$?: Observable<void>
): Promise<Blob> {
  let request$ = this.http.post(
    `${this.base}/generate`, config,
    { responseType: 'blob', observe: 'response' }
  );
  if (cancellation$) {
    request$ = request$.pipe(takeUntil(cancellation$));
  }
  return (await firstValueFrom(request$)).body;
}
```

L'abbandono della connessione viene rilevato lato backend tramite un listener sull'evento `close` della response Express, registrato prima dell'inizio dell'elaborazione. Il flag `clientCancelled` viene passato a `ReportGeneratorService` tramite la funzione *isCancelled()* che il servizio controlla nei punti critici della pipeline per interrompere la generazione, evitando così di consumare token API inutilmente.

Listing 5.9: report.controller.ts: rilevamento cancellazione tramite evento close

```
@Post('generate')
async generateReport(@Body() config: ReportConfigDto,
                     @Res() res: Response) {
  let clientCancelled = false;

```

```
  res.on('close', () => {
    if (!res.headersSent) clientCancelled = true;
  });

  const isCancelled = () => clientCancelled;
  const result = await this.reportGeneratorService
    .generateReport(config, influxData, isCancelled);

  if (clientCancelled) return;
  // ... invio risposta
}
```

`QueryHistoryService` persiste la cronologia delle query e preferiti in `localStorage` con chiavi distinte per utente. Il ricaricamento dei dati al cambio di sessione é implementato tramite un *effect()* Angular che reagisce al signal `currentUser` di `AuthService`: ogni volta che l'identità dell'utente cambia, l'effect si riesegue automaticamente caricando la cronologia associata al nuovo account, senza che i componenti che usano il servizio debbano gestire il ciclo di vita esplicitamente.

5.3 Containerizzazione e Deployment

L'intero sistema é containerizzato tramite Docker e orchestrato con Docker Compose, che definisce tre servizi distinti. Il servizio `api` esegue il backend NestJS ed é dotato di un volume persistente per il database SQLite, garantendo che i dati degli utenti sopravvivano ai riavvi del container. Il servizio `redis` esegue Redis 7 Alpine e mantiene la persistenza della coda dei job di export tramite un volume dedicato. Il servizio `frontend` esegue Nginx con l'applicazione Angular compilata in modalità production. I tre servizi comunicano tramite una rete bridge Docker interna `influx-dashboard-network`.

Il Dockerfile del frontend adotta un build multi-stage. Nella prima fase, un'immagine `node:20-slim` compila l'applicazione Angular in modalità production, producendo i file statici ottimizzati con tree-shaking e minificazione. Nella seconda fase, solo i file statici prodotti vengono copiati in un'immagine `nginx:alpine` escludendo node_modules, sorgenti TypeScript e tutti gli artefatti di build. Questo riduce significativamente le dimensioni dell'immagine finale rispetto a un approccio single-stage.

Nginx svolge un duplice ruolo: serve i file statici dell'applicazione Angular e funge da reverse proxy verso il backend NestJS per tutte le richieste dirette a `/api/`. Questa configurazione evita problemi CORS in produzione, poiché frontend e backend appaiono sullo stesso dominio dal punto di vista del browser. Un aspetto specifico della configurazione riguarda i timeout del blocco proxy: `proxy_read_timeout`, `proxy_send_timeout`

e `proxy_connect_timeout`, sono tutti impostati a 600 secondi, coerentemente con il timeout esteso nel backend per la generazione dei report LLM.

Listing 5.10: nginx.conf: configurazione reverse proxy

```
location /api/ {
    proxy_pass http://api:3000;
    proxy_http_version 1.1;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;

    # Timeout estesi per operazioni lunghe (generazione report LLM)
    proxy_connect_timeout 600s;
    proxy_send_timeout    600s;
    proxy_read_timeout    600s;
}

location / {
    try_files $uri $uri/ /index.html;
    add_header Cache-Control "no-cache, no-store, must-revalidate";
}
```

Il blocco `location` utilizza il pattern `try_files $uri $uri/ index.html`: Nginx tenta prima di servire il file statico corrispondente al percorso richiesto, e solo se non lo trova restituisce `index.html`. Questo é necessario perché in una SPA il routing é gestito interamente lato client da Angular. Percorsi come `/dashboard` non corrispondono a file fisici sul server, quindi senza questo pattern Nginx restituirebbe un errore 404 invece di lasciare il router Angular gestisca la navigazione.

Le immagini Docker sono pubblicate su Docker Hub, consentendo il deployment del sistema completo tramite un singolo `docker-compose up` senza necessità di compilare il codice sorgente localmente.

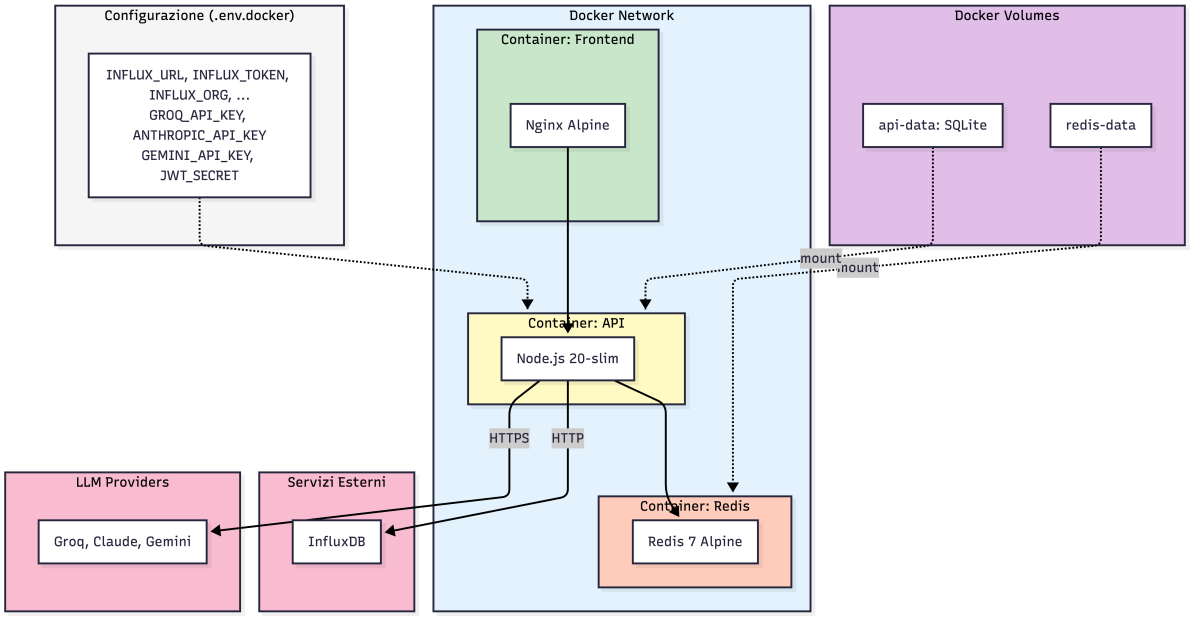


Figura 5.2: Architettura di deployment Docker.

Capitolo 6

Valutazione

Questo capitolo presenta la valutazione sperimentale della piattaforma su due dimensioni principali. La prima riguarda la scalabilità del motore di query, analizzando come la latenza varia al crescere dell'arco temporale interrogato e al variare della finestra di aggregazione. La seconda riguarda il modulo di generazione automatica di report, valutando le prestazioni e la qualità delle analisi prodotte dai tre provider LLM integrati nel sistema. Per ciascuna dimensione vengono descritti il protocollo di test adottato e i risultati ottenuti, con osservazioni sulle implicazioni pratiche per il contesto agricolo di AgroSense.

6.1 Metodologia di Valutazione

La valutazione ha due obiettivi distinti. Il primo é verificare la scalabilità del motore di query al crescere del volume di dati interrogati, con l'obiettivo di identificare i regimi operativi in cui la latenza rimane accettabile per un uso interattivo della piattaforma. Il secondo obiettivo é confrontare i tre provider LLM integrati in AgroSense - Groq, Gemini e Claude - su dimensioni complementari: velocità di risposta, costo per chiamata, capacità di descrivere correttamente i dati e accuratezza nel riportare i valori numerici precisi, al fine di fornire indicazioni pratiche sulla scelta del provider in funzione del contesto d'uso. Per ottenere ciò, la valutazione sperimentale è stata strutturata in due ambiti distinti, ciascuno con un protocollo di test dedicato.

Per la valutazione della scalabilità del motore di query è stata sviluppata una suite di benchmark deterministica in Python, progettata per misurare la latenza end-to-end delle richieste HTTP all'API del backend. Il determinismo delle query garantisce la riproducibilità dei risultati: a parità di configurazione, le stesse combinazioni di measurement e archi temporali vengono interrogate in ogni esecuzione, eliminando la variabilità delle

scelte casuali. Sono stati eseguiti due test distinti. Il primo tiene fissa la finestra di aggregazione a 15 minuti e varia l'arco temporale tra 1 e 90 giorni, confrontando query con aggregazione e query con dati grezzi su 50 esecuzioni per tipo di query per configurazione, per un totale di 600 misurazioni. Le *query con aggregazione* sono query che richiedono al database di calcolare un'aggregazione statistica - media, minimo, massimo o somma - dei valori all'interno di finestre temporali fisse, nel caso specifico a 15 minuti, restituendo un numero di righe proporzionale al numero di finestre e non al numero di misurazioni originali. Le *query sui dati grezzi*, al contrario, richiedono tutti i punti registrati dai sensori senza alcuna aggregazione lato server, con un volume di dati che cresce linearmente con l'arco temporale. La distinzione è rilevante per AgroSense perché la piattaforma supporta entrambe le modalità: l'utente può scegliere se visualizzare l'andamento aggregato o scaricare l'intera serie storica grezza per analisi esterne.

Il secondo benchmark tiene fisso l'arco temporale a 30 giorni e varia la finestra di aggregazione da 1 minuto ad 1 ora, confrontando query su 2 e 3 measurement su 50 esecuzioni per tipo di query per configurazione, per un totale di 500 misurazioni. Con *2 measurement* si intende una query avanzata che interroga simultaneamente due tipologie di sensori, ad esempio umidità atmosferica e pressione, mentre con *3 measurement* si aggiunge una terza variabile,come la temperatura del suolo. Questa configurazione corrisponde direttamente al caso d'uso della query avanzata di AgroSense, dove l'utente può selezionare più measurement in un'unica interrogazione. I risultati sono rappresentati tramite boxplot per catturare la distribuzione completa della latenza, inclusi quartili e ampiezza interquartilica.

La valutazione del modulo di reportistica si basa invece su un insieme di casi di test per ciascuno dei quali è stata precalcolata una ground truth a partire dai dati reali presenti su InfluxDB. Per ogni test case, il prompt corrispondente viene inviato a ciascuno dei tre provider LLM, raccogliendo le risposte. Tali risposte vengono poi confrontate con la rispettiva ground truth tramite le metriche di similarità semantica e accuratezza numerica descritte in seguito. Per ogni chiamata vengono inoltre registrate latenza e numero di token, in modo da poter confrontare i provider anche sotto il profilo delle prestazioni operative. Per la valutazione del modulo di generazione automatica dei report sono stati definiti 7 casi di test, suddivisi in 3 statici e 4 dinamici. I *casi statici* utilizzano dati di novembre 2025 con valori ground truth noti a priori: il primo riguarda l'umidità atmosferica rilevata dalla stazione WS-5-TRASVERSALE, con media mensile di riferimento pari a 88.31%; il secondo la temperatura della stazione WS-1-AGRIBIO, con media di riferimento di 8.17°C, il terzo la pressione atmosferica della stazione WS-8-VIA LARGA, con media di riferimento di 1019.41 hPa. Per ciascun caso statico il modello riceve le statistiche descrittive del sensore e viene valutato su tre *key points* attesi, ad esempio per l'umidità: presenza di un livello costantemente elevato, media intorno all'88%, e stabilità del trend nel periodo. I *casi dinamici* interrogano direttamente InfluxDB all'avvio per ottenere i valori reali di dicembre 2025 e costruire il ground truth al momento dell'ese-

cuzione: il primo richiede al modello di identificare il giorno con la temperatura media giornaliera più alta tra i dati del sensore EM-500-12; il secondo il giorno con l'umidità del suolo media più alta del sensore EM-500-10; il terzo il giorno con l'umidità atmosferica media più bassa della stazione WS-2-VIA NUOVA. Il quarto caso dinamico, classificato come difficoltà *alta*, richiede di identificare il giorno con la massima escursione termica della stazione WS-1-AGRIBIO, intesa come differenza tra temperatura massima e minima giornaliera: questo scenario richiede al modello un ragionamento multi-step su un valore derivato, non direttamente leggibile nelle statistiche fornite. Ciascun caso di test è stato eseguito sui 3 provider della piattaforma: Claude con `claude-sonnet-4-20250514`, Groq con `llama-3.3-70b-versatile` e Gemini con `gemini-2.5-flash`.

La valutazione delle risposte avviene su due dimensioni: la similarità semantica, che misura quanto il contenuto del report generato sia concettualmente allineato con i key point attesi, indipendentemente dalla formulazione lessicale utilizzata [18]. Il calcolo avviene tramite il modello `jina-embeddings-v3` di Jina configurato con il task di `text-matching` [38]: il modello converte separatamente il key point atteso e il testo del report in vettori numerici nello spazio degli embedding, dopodiché la similarità tra i due vettori viene misurata tramite similarità del coseno, normalizzata nell'intervallo $[0, 1]$, come illustrato in Figura 6.1. Un valore prossimo a 1 indica che i due testi sono semanticamente equivalenti, un valore prossimo allo 0 che non condividono alcun contenuto rilevante. I valori di similarità sono interpretabili in termini relativi: un punteggio più alto indica che il testo generato è semanticamente più vicino al key point atteso, ma non esiste una soglia assoluta al di sopra della quale il concetto possa considerarsi correttamente identificato [38]. I risultati sono quindi utilizzati per confrontare i provider tra loro, non come misura assoluta di correttezza. La seconda dimensione é quella dell'accuratezza numerica, verificata tramite assertion sui valori riportati. Le metriche di performance, infine, includono latenza, token consumati e costo stimato per chiamata.

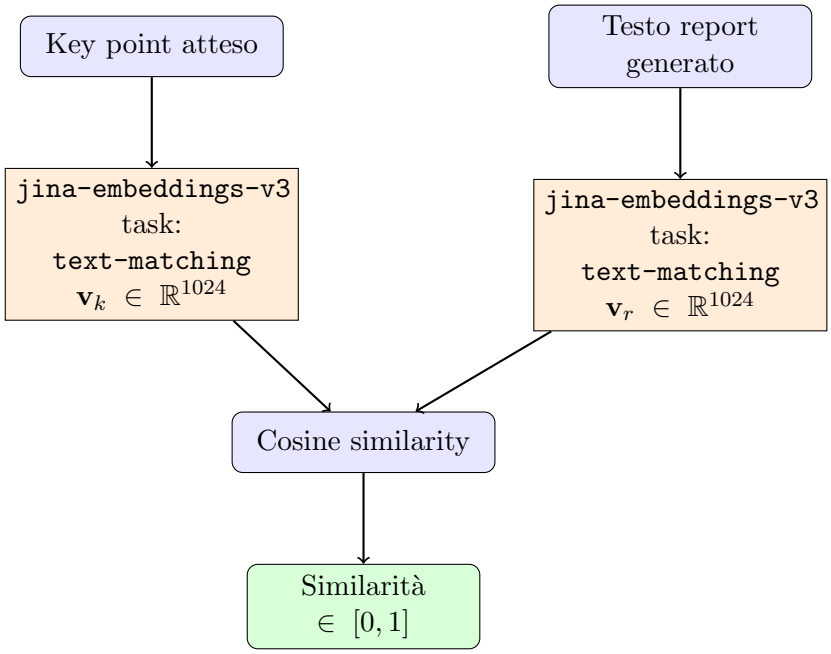


Figura 6.1: Schema del calcolo della similarità semantica per una singola iterazione del confronto tra il testo del report e un key point atteso. La similarità del coseno, il cui range grezzo è $[-1, 1]$, viene normalizzata nell'intervallo $[0, 1]$ prima di essere riportata.

6.2 Scalabilità: Latency vs Time Span (aggregated vs raw)

Il primo benchmark valuta come la latenza delle query vari al crescere dell'arco temporale interrogato, tenendo fissa la finestra di aggregazione a 15 minuti. Il confronto è tra due tipologie di query, entrambe eseguite su 2 measurement:

- **Query aggregate** (`advanced`): il database esegue internamente un'operazione di aggregazione statistica sui valori grezzi all'interno di finestre temporali fisse, nel caso specifico di 15 minuti. Il risultato restituito al backend contiene un punto per finestra: il numero di righe dipende quindi dall'arco temporale e dalla dimensione della finestra, non dal numero di misurazioni effettivamente registrate dai sensori.
- **Query sui dati grezzi** (`advanced_raw`): nessuna aggregazione viene eseguita lato database. Vengono restituiti tutti i punti registrati dai sensori nell'intervallo richiesto, con un volume che cresce linearmente al crescere dell'arco temporale e dipende dalla frequenza di campionamento di ciascuna stazione.

La distinzione è rilevante per AgroSense perché la piattaforma supporta entrambe le modalità: l'utente può scegliere se visualizzare l'andamento aggregato oppure scaricare l'intera serie storica grezza per analisi esterne.

I risultati, riportati in Figura 6.2, mostrano un andamento crescente della latenza al crescere dell'arco temporale per entrambe le tipologie di query, ma con dinamiche marcatamente diverse. Per le query con aggregazione, la crescita é sostanzialmente sublineare: la latenza mediana passa da circa 90 ms per un giorno a circa 400 ms per 30 giorni, fino a circa 1270 ms per 90 giorni, con una variazione contenuta in tutto il range testato. Le query sui dati grezzi mostrano invece una dispersione molto più elevata a partire da 30 giorni. A 90 giorni, la distribuzione mostra valori compresi tra circa 2400 ms e 6000 ms, a seconda del numero di righe restituite dalla combinazione di measurement, e il whisker superiore raggiunge oltre 7000 ms.

Questo comportamento é spiegabile considerando che InfluxDB applica l'aggregazione lato server durante l'esecuzione delle query Flux: indipendentemente dalla quantità di dati grezzi presenti nel database, il volume dei dati trasferiti verso il backend rimane proporzionale al numero di bucket temporali definiti dalle finestre, non al numero di misurazioni originali. Al contrario, le query senza aggregazione devono trasferire tutti i punti grezzi, il cui numero cresce linearmente con l'arco temporale. A titolo di confronto, a 30 giorni una query aggregata a 15 minuti restituisce circa 2800 righe, mentre la corrispondente query grezza ne restituisce tra 42.000 e 165.000 a seconda della stazione selezionata, con un evidente impatto sulla latenza.

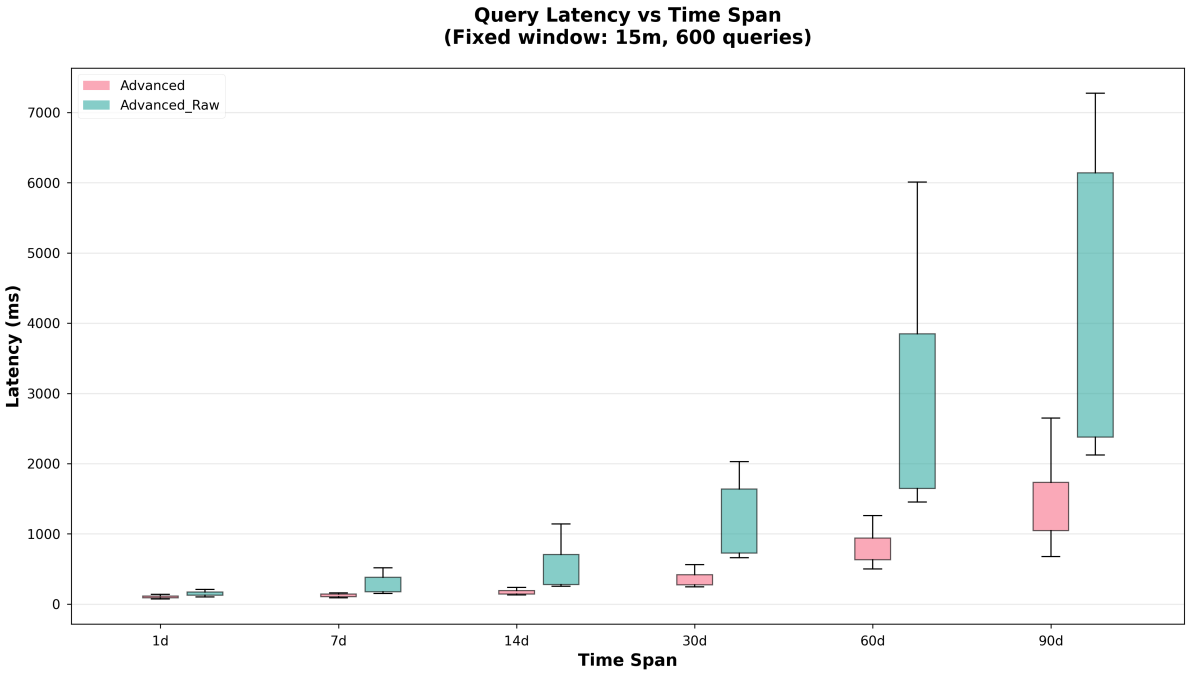


Figura 6.2: Latenza delle query al variare dell'arco temporale (finestra di aggregazione fissa: 15 minuti, 600 misurazioni totali). Il confronto tra query aggregate (`advanced`) e query sui dati grezzi (`advanced_raw`) evidenzia l'impatto dell'aggregazione lato server sulla scalabilità.

6.3 Performance: Latency vs Aggregation Window (2 vs 3 measurements)

Il secondo benchmark valuta come la latenza vari al variare della finestra di aggregazione, tenendo fisso l'arco temporale a 30 giorni. Il confronto è tra due configurazioni di query avanzata, che differiscono per il numero di measurement interrogati simultaneamente:

- **Query su 2 measurement** (`advanced_2`): una singola richiesta interroga in parallelo due tipologie di sensori distinte, ad esempio umidità atmosferica e pressione. Il backend costruisce due sub-query Flux separate, ciascuna con il proprio filtro su device e metodo di aggregazione, e riorganizza i risultati per timestamp prima di restituirli al frontend.
- **Query su 3 measurement** (`advanced_3`): la stessa struttura estesa a tre tipologie di sensori, aggiungendo ad esempio la temperatura del suolo. Ogni measurement aggiuntivo introduce una sub-query ulteriore e aumenta il volume di dati che il backend deve combinare prima della risposta.

L'obiettivo è isolare separatamente l'effetto della granularità temporale e del numero di measurement sulla latenza, in una configurazione che corrisponde direttamente al caso d'uso della modalità avanzata di AgroSense.

I risultati, riportati in Figura 6.3, mostrano una relazione inversa netta tra ampiezza della finestra e latenza: al crescere della finestra di aggregazione il numero di bucket temporali prodotti dalla query Flux diminuisce proporzionalmente, riducendo sia il carico computazionale lato server che il volume dei dati trasferiti. La latenza mediana per query su 2 measurement passa da circa 2000 ms con finestra a 1 minuto a circa 355 ms con finestra a 15 minuti, fino a circa 140 ms con finestra a 1 ora.

Un aspetto rilevante emerge a finestra di 1 minuto: la distribuzione delle query su 2 measurement presenta una varianza molto elevata, con IQR pari a circa 1500 ms e whisker superiore che raggiunge circa 5200 ms. Questo comportamento riflette la variabilità nel numero di righe restituite a seconda della stazione selezionata: a 30 giorni con finestra a 1 minuto sia le query su 2 che su 3 measurement possono restituire tra 27.000 e 43.000 righe, un volume che introduce instabilità nelle latenze. A partire da finestre di 5 minuti la varianza si riduce significativamente e il sistema si stabilizza.

Per quanto riguarda il confronto tra 2 e 3 measurement, il pattern atteso – latenza crescente con il numero di measurement - si osserva chiaramente a finestre intermedie di 5 e 15 minuti, dove le query su 3 measurement presentano latenze mediane superiori di circa 100-200 ms rispetto alle corrispondenti query su 2. A finestra di 1 minuto questo pattern non é verificabile direttamente: l'elevata varianza del 2-measurement spinge la sua mediana a valori superiori a quelli del 3-measurement, rendendo il confronto poco significativo. A finestre di 30 minuti e 1 ora le differenze diventano invece trascurabili, con scostamenti inferiori a 50 ms, suggerendo che a queste granularità il collo di bottiglia non é più il volume dei dati ma l'overhead fisso della richiesta HTTP.

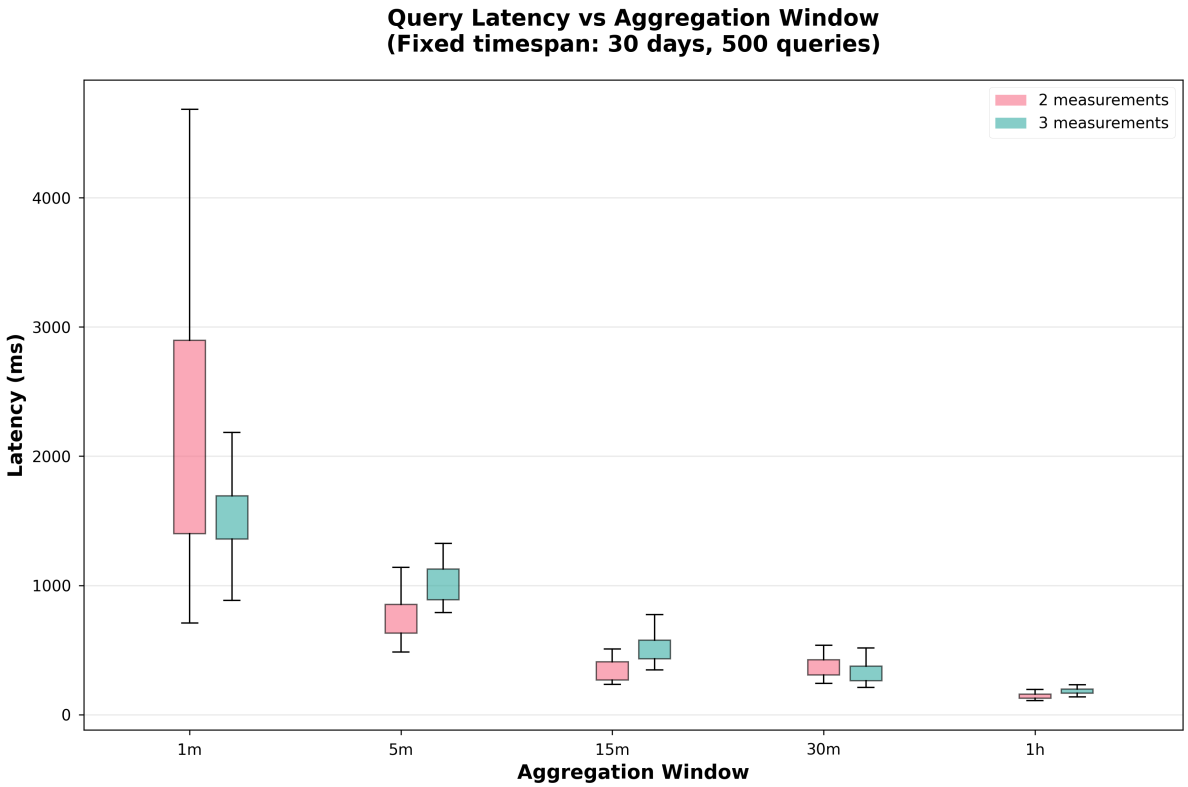


Figura 6.3: Latenza delle query al variare della finestra di aggregazione (arco temporale fisso: 30 giorni, 500 misurazioni totali). Il confronto tra query su 2 e 3 measurement evidenzia l'impatto del numero di measurement sulla latenza, in particolare per finestre temporali ridotte.

6.4 Confronto Provider LLM

6.4.1 Metriche di performance

Il confronto delle prestazioni rivela differenze talvolta marcate tra i tre provider. Groq registra una latenza media di 1633ms ed un valore P95 di 2018ms - dove il P95, o 95° percentile, indica la latenza al di sotto della quale ricade il 95% delle richieste, rappresentando quindi il comportamento del sistema nelle condizioni peggiori esclusi i casi estremi - , risultando circa sei volte più veloce di Gemini, che si attesta ad una latenza media di 10205ms e un P95 di 16282ms, e circa sette volte più veloce di Claude, che riporta valori di 11938ms e 14520ms rispettivamente di latenza e P95. Il divario è attribuibile all'infrastruttura di inferenza hardware-accelerata di Groq [28], progettata specificamente per minimizzare la latenza di generazione. Il numero di token medi consumati per chiamata è invece sostanzialmente equivalente tra i tre provider - 1178 per Gemini, 1170

per Claude e 1120 per Groq - a conferma che la differenza di latenza non é imputabile ad output più lunghi o più corti, ma solo alla velocità di elaborazione. Infine il success rate si attesta al 100% per tutti i provider.

Sul piano del costo, Gemini risulta il provider più economico con un costo per chiamata di $0.000353 e un totale di $0.0074 per le 21 chiamate, seguito da Groq a $0.000738 per chiamata ($0.0155 totali). Claude si posiziona su una fascia di costo significativamente superiore, con $0.008721 per chiamata e $0.1831 per l'intera sessione di test, un valore circa 25 volte maggiore rispetto a Gemini. Questo differenziale di costo è rilevante per il contesto applicativo di AgroSense, dove la generazione di report potrebbe essere richiesta con frequenza elevata da parte di più utenti.

Tabella 6.1: Metriche di performance dei provider LLM a confronto.

Metrica	**Gemini**	**Claude**	**Groq (Llama)**
Chiamate	21	21	21
Success Rate	100%	100%	100%
Latenza Media	10205 ms	11938 ms	**1633 ms**
Latenza P95	16282 ms	14520 ms	**2018 ms**
Token Medi	1178	1170	**1120**
Costo Totale	**$0.0074**	$0.1831	$0.0155
Costo per Chiamata	**$0.000353**	$0.008721	$0.000738

6.4.2 Qualità semantica

L'analisi della similarità semantica è stata effettuata su diversi scenari, come quello dell'umidità relativa riportato in Tabella 6.2, verificando la capacità di ciascun provider di identificare e descrivere correttamente tre key point specifici del test. Per quello sull'umidità i key point proposti richiedevano: la presenza di un livello di umidità costantemente elevato, il valore medio dell'umidità intorno all'88%, e la stabilità del trend nel periodo analizzato.

I risultati, ottenuti tramite il modello `jina-embeddings-v3` con task `text-matching`, mostrano una sostanziale equivalenza tra i tre provider: per il primo key point le similarità si attestano tra 81.2% (Groq), 82.3% (Gemini) e 83.1% (Claude); per il secondo tra 74.3% (Groq), 76.0% (Claude) e 77.8% (Gemini); per il terzo Groq ottiene il punteggio più alto (75.6%), seguito da Claude (74.0%) e Gemini (69.2%).

È comunque interessante osservare alcune tendenze trasversali. Il primo key point, che richiede una valutazione qualitativa del livello di umidità, ottiene i punteggi più alti per i tre provider: questo suggerisce che descrivere un fenomeno in termini generali

è un compito in cui i modelli si comportano meglio rispetto a quelli che richiedono precisione numerica. Il secondo key point, che chiede di citare esplicitamente il valore medio dell'88%, registra invece punteggi più bassi, un risultato coerente con il fatto che la similarità semantica misura la vicinanza concettuale tra testi, non l'accuratezza specifica di un valore. Il terzo key point, relativo alla stabilità del trend, mostra la varianza maggiore tra i provider (69.2% Gemini, 75.6% di Groq), indicando che l'interpretazione della dinamica temporale è l'aspetto su cui i modelli divergono di più. L'assenza di un vincitore netto su questa dimensione conferma che, per scenari di analisi descrittiva, tutti e tre i provider generano risposte semanticamente comparabili, e che le differenze qualitative vanno cercate nella dimensione dell'accuratezza numerica analizzata nella sezione successiva.

Tabella 6.2: Similarità semantica tra i key point attesi e le risposte generate (Humidity Analysis – WS-5-TRASVERSALE, Novembre 2025).

Key Point	**Claude**	**Groq**	**Gemini**
I dati mostrano un livello di umidità costantemente elevato	**83.1%**	81.2%	82.3%
La media dell'umidità si attesta intorno all'88%	76.0%	74.3%	**77.8%**
Il trend è stabile nel periodo analizzato	74.0%	**75.6%**	69.2%

6.4.3 Accuratezza numerica

La valutazione dell'accuratezza numerica ha evidenziato due pattern distinti a seconda della complessità dello scenario. Nei casi più semplici, come l'identificazione del giorno con la temperatura media giornaliera più alta (`Max Temperature Day`) tutti e tre i provider hanno raggiunto un'accuratezza del 100% su cinque run consecutive, identificando correttamente il 23 dicembre 2025 e riportando il valore di circa 9.3°C in ogni generazione, come da Tabella 6.3. Questo risultato conferma che, in presenza di valori direttamente osservabili nei dati aggregati, la capacità di reportistica numerica dei modelli è equivalente e affidabile.

Tabella 6.3: Valutazione del giorno con temperatura massima (Max Temperature Day – EM-500-12, Dicembre 2025). Ogni provider ha eseguito 5 run per assertion.

Assertion	**Gemini**	**Claude**	**Groq (Llama)**
Identifica il giorno con media giornaliera più alta (23 dic. 2025)	**5/5 (100%)**	**5/5 (100%)**	**5/5 (100%)**
Menziona il valore di temperatura media giornaliera (˜9.3°C)	**5/5 (100%)**	**5/5 (100%)**	**5/5 (100%)**
Accuracy Complessiva	**100%**	**100%**	**100%**

Le differenze emergono invece in scenari ad alta specificità. Il caso `Max Thermal Excursion Day` - identificazione del giorno con la maggiore escursione termica, intesa come differenza tra temperatura massima e minima giornaliera - ha richiesto ai modelli di ragionare su un valore derivato piuttosto che direttamente leggibile nei dati. L'identificazione del giorno corretto (11 dicembre 2025) è riuscita a Claude e Gemini con il 100% di successo su cinque run mentre Groq ha ottenuto l'80% con 4/5; il riconoscimento del valore numerico dell'escursione, calcolato in circa 16.2°C, ha invece prodotto esiti marcatamente asimmetrici: Gemini lo ha riportato correttamente in tutte e cinque le run (100%) mentre Claude si attesta a 1/5 (20%) e Groq su 2/5, come si evince dalla Tabella 6.4.

Tabella 6.4: Valutazione numerica dell'escursione termica massima (Max Thermal Excursion Day – WS-1-AGRIBIO, Dicembre 2025). Ogni provider ha eseguito 5 run per assertion.

Assertion	**Groq (Llama)**	**Claude**	**Gemini**
Identifica il giorno con la massima escursione termica (11 dic. 2025)	4/5 (80%)	**5/5 (100%)**	**5/5 (100%)**
Menziona il valore dell'escursione termica (˜16.2°C)	2/5 (40%)	1/5 (20%)	**5/5 (100%)**
Accuracy Complessiva	60%	60%	**100%**

È opportuno precisare che i valori percentuali riportati rappresentano medie su più esecuzioni dello stesso test. La varianza osservata nelle singole run è stata significativa: in alcune esecuzioni tutti i provider identificavano e riportavano correttamente il valore dell'escursione, mentre in altre la stessa risposta risultava assente o approssimata. Claude e Groq hanno mostrato la variabilità più alta su questo specifico scenario, con alcune run in cui l'accuratezza numerica era particolarmente bassa, come documentato nella Tabella 6.5. Questo pattern suggerisce che la capacità di riportare valori derivati complessi non è stabile tra generazioni successive dello stesso modello.

Tabella 6.5: Esempio di esecuzione con bassa accuratezza numerica (Max Thermal Excursion Day – WS-1-AGRIBIO, Dicembre 2025), a dimostrazione della variabilità tra run successive.

Assertion	**Groq (Llama)**	**Claude**	**Gemini**
Identifica il giorno con la massima escursione termica (11 dic. 2025)	4/5 (80%)	**5/5 (100%)**	**5/5 (100%)**
Menziona il valore dell'escursione termica (˜16.2°C)	0/5 (0%)	0/5 (0%)	**5/5 (100%)**
Accuracy Complessiva	40%	50%	**100%**

6.4.4 Considerazioni comparative

I risultati complessivi delineano tre profili distinti. Groq si afferma come la scelta ottimale per contesti in cui la latenza è il vincolo principale: la sua velocità di risposta, combinata con un costo moderato e una qualità semantica allineata agli altri provider, lo rende la scelta predefinita della piattaforma. Claude si posiziona come il provider più costoso senza che questo si traduca in un vantaggio misurabile né sulla qualità semantica, dove i tre provider risultano sostanzialmente equivalenti, né sull'accuratezza numerica, dove registra la variabilità più elevata nei casi complessi. Gemini emerge come il provider più bilanciato: offre il costo per chiamata più basso tra i tre e la migliore accuratezza numerica nei casi ad alta specificità, a fronte di una latenza P95 che nei casi peggiori raggiunge $16.282ms$, superiore a quella di Claude. Per contesti in cui la correttezza dei valori derivati è prioritaria rispetto alla velocità di risposta, Gemini rappresenta quindi la scelta più indicata.

Capitolo 7

Conclusioni

7.1 Contributi

Questo lavoro ha affrontato la progettazione, lo sviluppo e la valutazione di AgroSense, una piattaforma web per l'accesso e l'analisi personalizzata dei dati raccolti dall'infrastruttura IoT del progetto TRACE. Il punto di partenza è stato l'analisi dei limiti degli strumenti esistenti, in particolare di Grafana. Le esigenze da soddisfare comprendevano: interrogare i dati senza conoscere Flux, esportarli in formati standard e ottenere analisi interpretative automatiche.

L'architettura a tre livelli con separazione tra presentazione, logica applicativa e dati ha consentito di sviluppare ogni componente in modo indipendente. La scelta di un data layer multi-database - InfluxDB per le serie temporali, SQLite per i dati applicativi, Redis per la gestione asincrona dei job - riflette i diversi requisiti di ogni caso d'uso.

Il confronto sperimentale tra i tre provider LLM ha prodotto risultati che non si limitano a certificare il funzionamento del sistema, ma offrono indicazioni pratiche per la scelta del modello in base al contesto d'uso. Groq si distingue per la latenza, circa sette volte inferiore a Claude, a parità di qualità semantica. Gemini risulta il più economico e il più accurato nei casi che richiedono il calcolo di valori derivati. Claude, nonostante il costo significativamente superiore, non mostra vantaggi misurabili sulle altre dimensioni valutate.

7.2 Limitazioni

Il lavoro presenta alcune limitazioni che é necessario documentare. La valutazione è stata condotta attraverso benchmark prestazionali e test strutturati sui provider LLM,

ma non include una validazione con utenti reali. Verificare che il query builder risulti perfettamente accessibile per agronomi e tecnici del campo richiederebbe sessioni di testing con utenti reali. Questa mancanza non compromette la validità tecnica del sistema, ma lascia aperta la questione di quanto l'interfaccia risponda in pratica alle aspettative dell'utilizzatore finale.

La generazione di report avviene come operazione sincrona, con un timeout molto esteso, necessario per gestire la pipeline completa - cleaning, statistiche, correlazioni, chiamata al modello e generazione PDF - con operazioni su grandi dataset. Questa scelta funziona bene in scenari di utilizzo sequenziale, ma rappresenta un potenziale collo di bottiglia se più utenti richiedono report in parallelo.

I test sull'accuratezza degli LLM hanno evidenziato differenze, anche marcate, negli scenari più complessi che prevedono il calcolo di valori derivati, come quello dell'escursione termica massima giornaliera. Questo indica che i report generati automaticamente richiedono sempre una verifica umana prima di essere usati per decisioni operative.

7.3 Sviluppi Futuri

La direzione di sviluppo più immediata riguarda la trasformazione del processo di generazione report da sincrono ad asincrono, sfruttando l'infrastruttura Bull/Redis già presente nel sistema per l'export. Questo consentirebbe di eliminare il timeout esteso e di supportare richieste concorrenti senza degradare la disponibilità per gli altri utenti.

Un'estensione che potrebbe migliorare sostanzialmente la qualità delle analisi è l'introduzione di un layer di Retrieval-Augmented Generation: fornire al modello contesto agronomico specifico sulle MAPs - valori di riferimento stagionali, soglie critiche per le specie monitorate, correlazioni attese tra variabili ambientali e qualità del prodotto - ridurrebbe le imprecisioni nelle analisi e renderebbe le osservazioni più rilevanti per il contesto specifico del progetto TRACE. La letteratura esaminata nel Capitolo 2 documenta l'impatto di questo approccio sulla qualità delle risposte in scenari analoghi.

Infine, l'aggiunta di un sistema di soglie e notifiche permetterebbe di estendere la piattaforma dal dominio dell'analisi retrospettiva a quello del supporto decisionale proattivo: la possibilità di ricevere alert automatici al superamento di valori critici per temperatura, umidità o conducibilità sarebbe utile soprattutto per la prevenzione tempestiva di stress idrici o termici nelle coltivazioni di piante officinali, dove le condizioni ambientali influenzano direttamente la concentrazione dei principi attivi.

Bibliografia

[1] M. Ayaz, M. Ammad-Uddin, Z. Sharif, A. Mansour, and E.-H. M. Aggoune, "Internet-of-things (iot)-based smart agriculture: Toward making the fields talk," *IEEE Access*, vol. 7, pp. 129551–129583, 2019.

[2] K. Chicaiza, R. X. Paredes, I. M. Sarzosa, S. G. Yoo, and N. Zang, "Smart farming technologies: A methodological overview and analysis," *IEEE Access*, vol. 12, pp. 164922–164941, 2024.

[3] C. Kamienski, J.-P. Soininen, M. Taumberger, R. Dantas, A. Toscano, T. Salmon Cinotti, R. Filev Maia, and A. Torre Neto, "Smart water management platform: Iot-based precision irrigation for agriculture," *Sensors*, vol. 19, no. 2, 2019.

[4] B. Queté, A. Heideker, I. Zyrianoff, D. Ottolini, J. H. Kleinschmidt, J.-P. Soininen, and C. Kamienski, "Understanding the tradeoffs of lorawan for iot-based smart irrigation," in *2020 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor)*, pp. 73–77, 2020.

[5] K. Sharma and S. K. Shivandu, "Integrating artificial intelligence and internet of things (iot) for enhanced crop monitoring and management in precision agriculture," *Sensors International*, vol. 5, p. 100292, 2024.

[6] I. Zyrianoff, A. Iannoli, F. Montori, L. Sciullo, L. Bononi, A. Baldissara, B. Brintazzoli, E. Dall'Olio, M. Alpi, R. E. Sferrazza, G. Dinelli, I. Marotti, and M. Di Felice, "Traceability and research in the agricultural chain of medicinal and aromatic plants (maps): An iot-based approach," in *2025 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor)*, 2025. Proceedings not yet published.

[7] Grafana Labs, "Grafana documentation," 2024. Ultimo accesso: febbraio 2026.

[8] Elastic N.V., "Kibana: Visualize, explore, and manage data in Elasticsearch," 2024. Ultimo accesso: febbraio 2026.

[9] InfluxData, "Get started with Flux and InfluxDB," 2024. Ultimo accesso: febbraio 2026.

[10] Timescale, "TimescaleDB documentation," 2024. Ultimo accesso: febbraio 2026.

[11] Prometheus Authors, "Overview – Prometheus," 2024. Ultimo accesso: febbraio 2026.

[12] A. Kaloxylos, A. Groumas, V. Sarris, L. Katsikas, P. Magdalinos, E. Antoniou, Z. Politopoulou, S. Wolfert, C. Brewster, R. Eigenmann, and C. Maestre Terol, "A cloud-based farm management system: Architecture and implementation," *Computers and Electronics in Agriculture*, vol. 100, pp. 168–179, 2014.

[13] S. Braun, R. Carbon, and M. Naab, "Piloting a mobile-app ecosystem for smart farming," *IEEE Software*, vol. 33, no. 4, pp. 9–14, 2016.

[14] R. Sapkota, R. Qureshi, M. Usman Hadi, S. Zohaib Hassan, F. Sadak, M. Shoman, M. Sajjad, F. Ali Dharejo, A. Paudel, J. Li, Z. Meng, J. Shutske, and M. Karkee, "Multi-modal llms in agriculture: A comprehensive review," *IEEE Transactions on Automation Science and Engineering*, vol. 22, pp. 22510–22540, 2025.

[15] M. T. Kuska, M. Wahabzada, and S. Paulus, "Ai for crop production – where can large language models (llms) provide substantial value?," *Computers and Electronics in Agriculture*, vol. 221, p. 108924, 2024.

[16] A. Hazarika, J. Guo, K. Parsons, Y. Nagai, T. Sumi, P. Orlik, and M. Rahmati, "Agrinex: Next-gen smart agriculture with llm-integrated uav-iot solutions," in *2025 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 646–652, 2025.

[17] L. Fang, W. Xiang, J. Jin, K. Liao, C. Liu, Y. Han, F. D. Salim, and Y.-P. P. Chen, "Agri-llm: Prompt-based large language model for emission data analytics in smart agriculture," *IEEE Internet of Things Journal*, vol. 12, no. 23, pp. 49186–49197, 2025.

[18] M. V. Leite, J. M. Abe, M. L. H. Souza, and I. de Alencar Nääs, "Enhancing environmental control in broiler production: Retrieval-augmented generation for improved decision-making with large language models," *AgriEngineering*, vol. 7, no. 1, 2025.

[19] S. N. Z. Naqvi and S. Yfantidou, "Time series databases and InfluxDB," tech. rep., Université libre de Bruxelles, 2017.

[20] C. Wang, J. Qiao, X. Huang, S. Song, H. Hou, T. Jiang, L. Rui, J. Wang, and J. Sun, "Apache iotdb: A time series database for iot applications," *Proc. ACM Manag. Data*, vol. 1, June 2023.

[21] ChirpStack, "ChirpStack: Open-source LoRaWAN network server," 2025. Ultimo accesso: febbraio 2026.

[22] Google, "Angular: Web application framework," 2025. Ultimo accesso: febbraio 2026.

[23] Angular Material, "Angular Material: UI component library for Angular," 2025. Ultimo accesso: febbraio 2026.

[24] F5, Inc., "NGINX: High-performance web server and reverse proxy," 2025. Ultimo accesso: febbraio 2026.

[25] NestJS, "NestJS: A progressive Node.js framework," 2025. Ultimo accesso: febbraio 2026.

[26] Redis Ltd., "Redis: In-memory data store," 2025. Ultimo accesso: febbraio 2026.

[27] SQLite Consortium, "SQLite: Self-contained, serverless, relational database engine," 2025. Ultimo accesso: febbraio 2026.

[28] Groq, "Inside the LPU: Deconstructing Groq's speed," 2025. Ultimo accesso: febbraio 2026.

[29] Anthropic, "Claude: AI assistant by Anthropic," 2025. Ultimo accesso: febbraio 2026.

[30] Google, "Gemini: Google's most capable AI model," 2025. Ultimo accesso: febbraio 2026.

[31] Taskforce.sh, "Bull: Premium queue package for Node.js," 2025. Ultimo accesso: febbraio 2026.

[32] bettersqlite, "better-sqlite3: The fastest and simplest SQLite library for Node.js," 2025. Ultimo accesso: febbraio 2026.

[33] National Institute of Standards and Technology, "Recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC," 2007. NIST Special Publication 800-38D. Ultimo accesso: marzo 2026.

[34] Groq, "Llama 3.3 70B Versatile: Large language model," 2025. Ultimo accesso: febbraio 2026.

[35] Anthropic, "Claude sonnet 4: Anthropic model documentation," 2025. Ultimo accesso: febbraio 2026.

[36] Google, "Gemini 2.5 Flash: Generative AI model documentation," 2025. Ultimo accesso: febbraio 2026.

[37] Nx, "Nx: Smart monorepos," 2025. Ultimo accesso: febbraio 2026.

[38] Jina AI, “Jina embeddings,” 2024. Ultimo accesso: febbraio 2026.