Alma Mater Studiorum · Università di Bologna

Dipartimento di Informatica — Scienza e Ingegneria
Corso di Laurea Triennale in Informatica per il Management

Integrazione di Large Language Models con database strutturati: valutazione comparativa di metodologie con applicazione a dati IoT

Tesi di Laurea in Basi di Dati

Relatore:
Chiar.mo Prof.
MARCO DI FELICE

Presentata da:
GIADA FRANCESCHINI

Correlatore:
Dott.
LEONARDO CIABATTINI

Sessione MARZO 2026
Anno Accademico 2024/2025

A chi ha creduto in me prima che ci fosse "qualcosa in cui credere".
A chi ha aspettato, senza che glielo chiedessi, che i cerchi si chiudessero.
A chi mi ha insegnato che in fondo "stiamo tutti vivendo per la prima volta".

Ho sempre avuto una paura precisa: quella di non vivere abbastanza.
Di arrivare alla fine con le tasche piene di "avrei dovuto".
Questa tesi è la prova che quella paura, alla fine, vince.
Non è nata dall'ordine, né dalla linearità.
È nata dalla tenacia – quella silenziosa, cocciuta, che non si spegne.
E da chi, intorno a me, ha insistito quando io avevo smesso di farlo: se sono arrivata fin qui è anche perché c'era qualcuno che credeva ci arrivassi.

L'abitudine costruisce ciò che il talento da solo non può.

Ho imparato che la paura non scompare. Si fa piccola. Si fa meno importante di quello che stai costruendo.
E che c'è un modo solo di vivere senza rimpianti:
Riempire la propria vita di "non posso credere di averlo fatto".
Non di "avrei dovuto farlo".
Questa tesi è una di quelle cose.

A mamma Cristina e papà Graziano.

Indice

Introduzione **vii**

1 Stato dell'arte e background **1**

1.1 Large Language Model per dati strutturati 1

1.2 Il problema text-to-SQL 4

1.3 Metodologie di interazione LLM-database 6

2 Metodologia **13**

2.1 Task sperimentali 13

2.2 Ipotesi sperimentali 14

2.3 Disegno sperimentale 15

2.3.1 Esperimenti 15

2.3.2 Variabili sperimentali 15

2.3.3 Configurazioni 17

2.4 Test set e ground truth 19

2.5 Metriche di valutazione 22

3 Implementazione **27**

3.1 Dataset 27

3.2 Architettura del sistema 30

3.3 Approccio SQL-based 32

3.4 Approccio diretto (*context stuffing*) 34

3.5 Approccio *code interpreter* 36

3.6 Pipeline di analisi dati 37

3.7 Framework agentici e report generation 39

4 Risultati sperimentali 43

4.1 Setup sperimentale 43
4.2 Confronto tra approcci 44
4.3 Impatto del prompt engineering 50
4.4 Risorse, report e lingua 52
4.5 Discussione 55
4.6 Dashboard interattiva 58

Conclusioni 61

A Prompt utilizzati 63

A.1 Prompt per la generazione SQL – modello general-purpose . . 63
A.1.1 P1 – Solo schema 63
A.1.2 P2 – Schema e dati campione 64
A.1.3 P3 – Schema e dizionario dati 64
A.2 Prompt per la generazione SQL – modello specializzato 67
A.3 Prompt per l'approccio diretto (*context stuffing*) 68
A.4 Prompt per il *code interpreter* 69

B Interrogazioni di test 71

B.1 Interrogazioni con valutazione automatica (Q01–Q13) 71
B.2 Interrogazioni con valutazione semantica (Q14–Q18) 76

Bibliografia 79

Elenco delle figure

1.1 Tassonomia delle metodologie di interazione LLM-database implementate. 10

3.1 Distribuzione dei record per tipo di misura nel database. . . . 30
3.2 Periodo di attività dei 28 dispositivi IoT. I dispositivi EM-500-5, WS-4 e EM-500-19 hanno interrotto la trasmissione prima della fine del periodo di osservazione. 31
3.3 Architettura a quattro livelli del sistema di valutazione comparativa. 31
3.4 Flusso della pipeline SQL-based con meccanismo di retry. . . . 33
3.5 Flusso dell'approccio diretto (*context stuffing*) con estrazione programmatica dei dati. 36
3.6 Flusso dell'approccio code interpreter con esecuzione in sandbox e retry. 37
3.7 Ciclo ReACT dell'agente LangChain con tool calling e fallback programmatico. 42

4.1 Accuratezza e tasso di allucinazione per le 14 configurazioni testate. 47
4.2 Confronto multi-dimensionale dei 6 approcci con punteggio composito più elevato. 49
4.3 Heatmap degli esiti per categoria di query e configurazione. . . 49
4.4 Heatmap della matrice prompt × modello: punteggio composito per le 6 configurazioni. 51

4.5 Consumo di token per configurazione, suddiviso in token di prompt e di completamento. 53
4.6 Occupazione di memoria RAM prima e dopo l'esecuzione per ciascuna configurazione. 53
4.7 Flusso dati dalla pipeline di benchmark alla dashboard interattiva. 58
4.8 Pagina Dashboard: panoramica dei risultati con grafico a barre e tabella di ranking. 59
4.9 Pagina Experiments: tab E2 con la matrice prompt × modello e grafico comparativo. 60

Elenco delle tabelle

1.1 Confronto fra i principali benchmark text-to-SQL. 6
1.2 Confronto teorico delle metodologie di interazione LLM-database. 11

2.1 Ipotesi sperimentali. 14
2.2 Esperimenti e ipotesi associate. 15
2.3 Configurazioni per il task di retrieval. 18
2.4 Configurazioni per il task di analisi. 18
2.5 Le 18 interrogazioni del test set con categoria, difficoltà e tipo di valutazione. 20
2.6 Tipi di valutazione nel test set. 21
2.7 Interrogazioni edge case con la trappola specifica e il comportamento atteso. 22
2.8 Metriche di valutazione e relativi pesi nel punteggio composito. 25

3.1 Schema del database SQLite con le tre tabelle principali. . . . 28
3.2 Disponibilità delle misure per tipo di dispositivo. 29
3.3 Sei configurazioni SQL-based (3 prompt × 2 modelli). 34
3.4 Confronto fra le pipeline di retrieval e di analisi per gli approcci diretto e code interpreter. 38
3.5 Confronto operativo fra pipeline *bare-metal* e framework agentici. 40
3.6 Strumenti disponibili per l'agente di generazione report. 41

4.1 Le 14 configurazioni testate con approccio, modello, prompt e lingua. 45
4.2 Ranking delle 14 configurazioni per punteggio composito. S: semplicità, Fl: flessibilità, In: interpretabilità (metriche qualitative, costanti per tipo di approccio). 46
4.3 Risultati delle sei configurazioni SQL-based (3 prompt × 2 modelli). 50
4.4 Efficienza delle query SQL generate vs ground truth (approccio Qwen).* 52
4.5 Confronto fra italiano e inglese per entrambi i modelli con prompt P3. 54
4.6 Verifica delle otto ipotesi sperimentali. 55

Introduzione

La crescente diffusione di dispositivi connessi e sistemi di monitoraggio ha portato alla generazione di volumi di dati strutturati sempre più consistenti. In ambiti come il monitoraggio ambientale e infrastrutturale, reti di sensori IoT producono milioni di record al mese, archiviati in basi di dati con strutture complesse e tipi di misura eterogenei. L'accesso a queste informazioni è tipicamente predisposto tramite dashboard interattive, ma la possibilità di condurre analisi avanzate non previste dalle dashboard - interrogazioni personalizzate, correlazioni fra variabili, rilevamento di anomalie - resta riservata a chi possiede competenze specifiche di interrogazione dei dati.

I Large Language Model (LLM) rappresentano una possibile risposta a questo problema. Grazie alla loro capacità di comprendere domande formulate in linguaggio naturale e di generare codice o interrogazioni strutturate, i modelli linguistici di grandi dimensioni possono fungere da interfaccia fra l'utente e il database, traducendo richieste in operazioni eseguibili sul sistema di gestione dei dati. Il problema noto come *text-to-SQL*, ovvero la traduzione automatica di una domanda in linguaggio naturale nella corrispondente query SQL, è stato oggetto di intensa attività di ricerca negli ultimi anni, con la pubblicazione di benchmark consolidati come Spider [2] e BIRD [12] e lo sviluppo di modelli specializzati per questo compito. Tuttavia, la generazione di SQL non è l'unica strategia disponibile: un modello linguistico può anche ricevere direttamente i dati rilevanti nel proprio prompt e produrre una risposta senza generare interrogazioni, oppure scrivere un programma Python che interroga il database in modo autonomo. A queste strategie si

aggiungono i framework agentici, come LangChain [4] e LlamaIndex [5], che orchestrano il modello attraverso cicli di ragionamento e azione, dotandolo di strumenti per ispezionare lo schema e correggere i propri errori.

Nonostante la varietà di approcci disponibili, manca nella letteratura un confronto sistematico che li valuti tutti sullo stesso dataset reale, con gli stessi criteri e con modelli di dimensioni contenute eseguibili su hardware consumer. I benchmark accademici utilizzano tipicamente basi di dati sintetiche, e le valutazioni si concentrano su modelli di grandi dimensioni accessibili tramite API cloud. In molti contesti applicativi, tuttavia, la riservatezza dei dati impedisce l'invio a servizi esterni: dati di monitoraggio infrastrutturale, informazioni aziendali riservate o dati personali richiedono un'elaborazione locale. Questa esigenza di riservatezza motiva l'adozione di modelli di dimensioni contenute, eseguibili su hardware consumer senza GPU dedicata. La domanda pratica rimane aperta: quale metodologia funziona meglio quando il database è reale, le interrogazioni sono formulate nella lingua dell'utente e il modello opera in locale?

Il presente lavoro di tesi affronta questa domanda attraverso un'analisi comparativa di quattro metodologie di interazione fra LLM e basi di dati strutturate, applicate a un caso d'uso reale di monitoraggio IoT. Il dataset comprende 12,6 milioni di record provenienti da 28 dispositivi - otto stazioni meteorologiche e 20 sensori di suolo - distribuiti in Emilia-Romagna e monitorati per circa 12 mesi. Le metodologie confrontate sono la generazione SQL, il *context stuffing*, l'interpretazione di codice e i framework agentici, declinate in 14 configurazioni che combinano due modelli (uno general-purpose e uno specializzato, entrambi con sette miliardi di parametri), tre strategie di prompt con livelli crescenti di informazione contestuale e due lingue. La valutazione si basa su un punteggio composito a sei metriche: accuratezza, tasso di allucinazione, latenza, semplicità, flessibilità e interpretabilità. Lo studio è guidato da otto ipotesi sperimentali, verificate attraverso cinque esperimenti che isolano l'effetto di ciascuna variabile.

I risultati mostrano che la configurazione più efficace è quella SQL-based

con un'istruzione arricchita da un dizionario dei dati, che raggiunge il 76,9% di accuratezza con un tasso di allucinazione nullo. L'analisi rivela una forte interazione fra prompt engineering e scelta del modello: il dizionario dati produce un incremento di 38,4 punti percentuali sul modello general-purpose, ma solo 7,7 sul modello specializzato. I framework agentici, progettati per modelli di dimensioni maggiori, si sono rivelati inadatti con sette miliardi di parametri, ottenendo accuratezze del 7,7%. L'intero sistema di valutazione, comprensivo dei risultati e delle visualizzazioni, è stato reso accessibile attraverso una dashboard interattiva resa disponibile pubblicamente.

Il resto del manoscritto è organizzato come segue. Il Capitolo 1 presenta i fondamenti teorici del problema text-to-SQL, lo stato dell'arte sulle metodologie di interazione fra LLM e database e i framework agentici rilevanti. Il Capitolo 2 definisce il framework metodologico: i task sperimentali, le otto ipotesi, i cinque esperimenti, le variabili sperimentali e le metriche di valutazione. Il Capitolo 3 descrive il dataset, l'architettura del sistema e l'implementazione di ciascun approccio. Il Capitolo 4 presenta i risultati dei cinque esperimenti, la verifica delle ipotesi e la dashboard interattiva. Le conclusioni riassumono i contributi del lavoro, ne riconoscono i limiti e suggeriscono possibili direzioni future.

Capitolo 1

Stato dell'arte e background

L'interrogazione di basi di dati strutturate tramite linguaggio naturale è un problema che coinvolge due aree di ricerca: i modelli linguistici di grandi dimensioni, capaci di generare codice e interrogazioni a partire da testo, e le interfacce per l'accesso ai dati, che storicamente richiedono competenze tecniche specifiche. Questo capitolo presenta lo stato dell'arte su entrambi i fronti: l'architettura e le capacità dei modelli linguistici rilevanti per l'interazione con i dati, il problema *text-to-SQL* con i relativi benchmark e le diverse metodologie proposte in letteratura per connettere un modello a una base di dati.

1.1 Large Language Model per dati strutturati

I Large Language Model (LLM) sono modelli di apprendimento automatico basati sull'architettura *Transformer*, introdotta da Vaswani et al. [1] nel 2017. L'intuizione alla base del Transformer è il meccanismo di *self-attention*, che consente al modello di pesare in modo differenziato le diverse parti della sequenza di input quando produce ciascun elemento dell'output. A differenza delle architetture ricorrenti che elaborano i token in ordine sequenziale, il Transformer processa l'intera sequenza in parallelo, il che ha permesso di scalare l'addestramento a quantità di dati e dimensioni di modello preceden-

temente irrealizzabili. I modelli linguistici di grandi dimensioni sono, nella loro forma più diffusa, modelli generativi autoregressivi: dato un contesto testuale, predicono il token successivo più probabile, ripetendo il processo fino a completare la risposta.

La dimensione di un LLM viene espressa in numero di parametri, cioè i pesi della rete neurale che vengono appresi durante l'addestramento. I modelli più recenti raggiungono centinaia di miliardi di parametri, ma la ricerca ha dimostrato che anche modelli con sette miliardi di parametri possono raggiungere prestazioni soddisfacenti su compiti specifici, a patto di ricevere un addestramento mirato. In particolare, sono stati sviluppati modelli specializzati per la generazione di codice, capaci di produrre interrogazioni SQL, script Python e altro codice strutturato a partire da istruzioni in linguaggio naturale. La distinzione fra modelli *general-purpose* e modelli specializzati è rilevante per il presente lavoro, in quanto entrambe le tipologie sono state impiegate nella sperimentazione.

Per eseguire modelli con sette miliardi di parametri su hardware consumer, è necessario ricorrere a tecniche di *quantizzazione*, che riducono la precisione numerica dei pesi del modello - ad esempio da 16 bit in virgola mobile a quattro bit interi - con una perdita di qualità generalmente contenuta. La quantizzazione riduce il fabbisogno di memoria, rendendo possibile l'inferenza su un laptop senza GPU dedicata. Piattaforme come Ollama [9] semplificano questo processo fornendo un'interfaccia per scaricare ed eseguire modelli quantizzati in locale, senza dipendere da servizi cloud. L'inferenza locale offre vantaggi in termini di riservatezza dei dati, assenza di costi variabili e controllo completo sull'ambiente di esecuzione, aspetti particolarmente rilevanti in contesti aziendali dove i dati non possono essere trasmessi a server esterni.

Fra i modelli con sette miliardi di parametri rilevanti per questo contesto, Qwen2.5-Coder [13], sviluppato dal team Qwen di Alibaba, è un modello general-purpose per la generazione di codice, addestrato su un corpus che include oltre 5,5 trilioni di token fra codice sorgente, testo e dati.

Qwen2.5-Coder raggiunge l'88,4% sul benchmark HumanEval [3], che valuta la capacità di generare funzioni Python corrette a partire da una descrizione testuale.

SQLCoder [8], sviluppato da Defog.ai, è un modello specializzato per la generazione SQL, ottenuto tramite fine-tuning di StarCoder [10] - un modello da 15 miliardi di parametri per la generazione di codice - con addestramento aggiuntivo su coppie domanda-query SQL. SQLCoder utilizza un formato di istruzione a completamento, strutturato con sezioni marcate da intestazioni testuali, anziché il formato conversazionale adottato dai modelli di chat.

Arctic-Text2SQL-R1 [16], sviluppato da Snowflake con tecniche di apprendimento per rinforzo, rappresenta lo stato dell'arte per i modelli a sette miliardi di parametri sul benchmark BIRD (68,5% di execution match). Al momento della sperimentazione, questo modello non era disponibile sulla piattaforma Ollama, il che ha motivato la scelta di SQLCoder come modello specializzato.

Un aspetto che ha assunto particolare rilevanza nel presente lavoro è il *prompt engineering*, ovvero la progettazione del testo di input fornito al modello per guidarne il comportamento. A differenza del *fine-tuning*, che modifica i pesi del modello attraverso un addestramento aggiuntivo su dati specifici, il prompt engineering opera esclusivamente sul testo della richiesta, arricchendolo con informazioni contestuali, istruzioni esplicite e vincoli. Nel contesto dell'interazione con database, ciò si traduce nella possibilità di includere nel prompt la struttura delle tabelle, esempi di dati, un dizionario che descriva il significato delle colonne e avvertenze sulle particolarità del dataset. Come evidenziato in un recente studio di Chang e Fosler-Lussier [7], la formulazione dell'istruzione ha un impatto significativo sulle prestazioni dei modelli linguistici nella generazione di SQL, e la scelta delle informazioni da includere può risultare più determinante della scelta del modello stesso.

1.2 Il problema text-to-SQL

Il problema *text-to-SQL* consiste nella traduzione automatica di una domanda espressa in linguaggio naturale nella corrispondente query SQL, dato lo schema di una base di dati relazionale. In termini formali, dato un input composto da una domanda Q in linguaggio naturale e dallo schema S di un database, il compito è produrre una query SQL Y tale che la sua esecuzione sul database restituisca la risposta corretta alla domanda Q. Si tratta di un problema che si colloca all'intersezione fra l'elaborazione del linguaggio naturale e i sistemi di gestione di basi di dati, e che ha ricevuto attenzione crescente negli ultimi anni grazie alla disponibilità di modelli linguistici sempre più capaci di generare codice strutturato.

L'evoluzione degli approcci al text-to-SQL ha attraversato diverse fasi. I primi sistemi, sviluppati a partire dagli anni Settanta, si basavano su regole scritte manualmente per interpretare le domande e mapparle su strutture SQL predefinite. Con l'avvento delle tecniche di apprendimento automatico, sono stati proposti modelli di tipo *sequence-to-sequence* che trattano la domanda come una sequenza di input e l'interrogazione SQL come una sequenza di output, apprendendo la traduzione da coppie di esempi annotati. L'introduzione dell'architettura Transformer e dei modelli linguistici pre-addestrati ha segnato un cambiamento sostanziale: i LLM possono affrontare il compito in modo *zero-shot* o *few-shot*, senza necessità di addestramento specifico su coppie domanda-query, sfruttando la conoscenza della sintassi SQL acquisita durante il pre-addestramento su grandi corpora di codice.

La comunità scientifica ha prodotto diversi benchmark per valutare in modo standardizzato le prestazioni dei sistemi text-to-SQL. Il più noto è Spider, presentato da Yu et al. [2] nel 2018. Spider comprende 10.181 domande distribuite su 200 basi di dati in 138 domini diversi, con la peculiarità di richiedere la generalizzazione *cross-domain*: i database presenti nel test set non compaiono nel training set. Le interrogazioni coprono diversi livelli di complessità, dalle semplici selezioni a quelle con sotto-interrogazioni annidate, operazioni di *join* e aggregazioni con raggruppamento. Spider utilizza

il formato SQLite per i suoi database, il che ha reso questa variante di SQL lo standard di fatto per la valutazione.

Un benchmark più recente è BIRD, introdotto da Li et al. [12] nel contesto del *Datasets and Benchmarks Track* della conferenza NeurIPS 2023. BIRD si distingue da Spider per l'utilizzo di basi di dati più grandi e realistiche, con strutture complesse e dati reali, e per l'introduzione esplicita della necessità di conoscenza esterna al database: molte domande richiedono informazioni sul dominio che non sono desumibili dallo schema. Il benchmark include 12.751 coppie domanda-SQL distribuite su 95 basi di dati, e ha evidenziato come i modelli che ottengono buone prestazioni su Spider incontrino difficoltà significative su dati più vicini a scenari applicativi reali. Il miglior risultato noto su BIRD al momento della stesura di questo lavoro, ottenuto da un modello a 7 miliardi di parametri, è il 68,5% di *Execution Match* raggiunto da Arctic-Text2SQL-R1 [16], un modello addestrato da Snowflake con tecniche di apprendimento per rinforzo.

Oltre ai benchmark, diversi lavori hanno proposto strategie per guidare i modelli linguistici nella generazione SQL. Pourreza e Rafiei [11] hanno proposto DIN-SQL, un framework che scompone la generazione SQL in sotto-problemi risolti in sequenza con autocorrezione. Gao et al. [15], in uno studio pubblicato sulla rivista PVLDB, hanno proposto DAIL-SQL e condotto un confronto estensivo delle strategie di prompting per text-to-SQL, mostrando come la scelta della rappresentazione del prompt influenzi in misura rilevante il ranking dei modelli ed evidenziando la necessità di valutazioni multi-dimensionali.

Le metriche di valutazione adottate dai benchmark meritano una breve discussione. L'*exact match* verifica se l'interrogazione SQL generata è identica a quella di riferimento, a meno di variazioni sintattiche irrilevanti. Questa metrica è tuttavia limitata, poiché esistono spesso più interrogazioni corrette per la stessa domanda. L'*execution match*, adottata come metrica primaria sia da Spider che da BIRD, confronta i risultati dell'esecuzione: la query generata è considerata corretta se produce gli stessi record della query di

riferimento quando eseguita sul database.

Un aspetto emerso di recente riguarda la qualità stessa dei benchmark. Jin et al. [17] hanno analizzato le annotazioni di Spider e BIRD scoprendo errori pervasivi nelle interrogazioni di riferimento: il 52,8% delle domande nel sottoinsieme BIRD Mini-Dev contiene almeno un errore di annotazione, che altera in modo significativo le classifiche dei modelli. Questo risultato rafforza la motivazione per condurre valutazioni su dataset reali con ground truth verificata manualmente.

Le sfide principali del text-to-SQL restano molteplici: l'ambiguità del linguaggio naturale, la complessità della struttura (il modello deve comprendere le relazioni fra le tabelle per generare *join* corretti) e la necessità di conoscenza del dominio, evidenziata in modo particolare da BIRD. La Tabella 1.1 riassume le caratteristiche dei principali benchmark. LiveSQLBench [18], un benchmark con aggiornamento continuo, ha riportato che anche modelli di dimensioni molto superiori raggiungono solo il 28,67% su domande formulate in modo colloquiale, a conferma del fatto che il text-to-SQL su dati reali rimane un problema aperto.

Benchmark	Interrogazioni	Database	Anno	Caratteristica principale
Spider	10.181	200	2018	Cross-domain, complessità graduata
BIRD	12.751	95	2023	Dati reali, conoscenza esterna
LiveSQLBench	variabile	—	2025	Interrogazioni colloquiali, aggiornamento continuo

Tabella 1.1: Confronto fra i principali benchmark text-to-SQL.

1.3 Metodologie di interazione LLM-database

La generazione diretta di query SQL non è l'unico modo in cui un modello linguistico può interagire con una base di dati strutturata. La letteratura recente, sintetizzata nell'indagine di Hong et al. [14], individua diverse strategie che si differenziano per il grado di accesso al database concesso al modello, per il tipo di artefatto generato (SQL, codice eseguibile, testo) e per il livello

di autonomia del modello nel processo di interrogazione. La presente sezione descrive le principali metodologie dal punto di vista concettuale.

Il primo e più studiato approccio è quello SQL-based, in cui il modello riceve la domanda in linguaggio naturale e lo schema del database, e produce come output una query SQL. L'interrogazione viene poi eseguita sul database e il risultato viene restituito all'utente. Il vantaggio principale di questo approccio è la trasparenza: l'SQL generato è ispezionabile e verificabile, e l'esecuzione avviene attraverso il motore del database, che garantisce correttezza e ottimizzazione. I limiti risiedono nella dipendenza dalla qualità dell'SQL generato: se il modello produce una query sintaticamente corretta ma semanticamente errata - ad esempio riferendosi alla tabella sbagliata o confondendo i nomi delle colonne - il risultato sarà scorretto senza che l'errore sia immediatamente evidente. Studi in letteratura riportano che una quota significativa degli errori nei sistemi text-to-SQL è dovuta a confusione fra tabelle e a nomi di colonna errati.

Il secondo approccio, denominato approccio diretto o *context stuffing*, inverte la logica: anziché chiedere al modello di scrivere una query, i dati rilevanti vengono estratti programmaticamente dal database e inseriti direttamente nel prompt, insieme alla domanda. Il modello non interagisce con il database, ma risponde sulla base dei dati che gli vengono presentati nel contesto. L'estrazione programmatica può avvalersi di tecniche di analisi della domanda basate su espressioni regolari e parole chiave per selezionare le tabelle, le misure e gli intervalli temporali pertinenti. Il vantaggio di questo approccio è che il modello non può generare interrogazioni errate, poiché non ne produce; lo svantaggio è il rischio di allucinazione - il modello potrebbe produrre risposte plausibili ma non supportate dai dati forniti - e il vincolo della finestra di contesto, che limita la quantità di dati inseribili nel prompt. Con un database da 12,6 milioni di record, è evidente che non è possibile inserire tutti i dati nel contesto; la selezione diventa quindi un passaggio critico.

Il terzo approccio è il *code interpreter*, in cui il modello genera un pro-

gramma completo - tipicamente in Python - che interroga il database, elabora i risultati e produce l'output. Il codice generato viene eseguito in un ambiente isolato (*sandbox*) per ragioni di sicurezza, e il risultato dell'esecuzione viene restituito all'utente. Rispetto all'approccio SQL-based, il code interpreter offre maggiore flessibilità: il modello può eseguire calcoli, aggregazioni e trasformazioni che sarebbero complesse da esprimere in SQL puro, come il calcolo di correlazioni fra variabili o la generazione di grafici. Il costo è una latenza più elevata, dovuta alla generazione di codice più lungo e alla necessità di eseguirlo in un processo separato, e una minore trasparenza, poiché il codice Python è generalmente più difficile da verificare di una query SQL.

Ai tre approcci di retrieval si affiancano due approcci di analisi dati. La distinzione fra retrieval e analysis riguarda la natura dell'output atteso: nel retrieval, si cerca un valore numerico puntuale o un insieme di record che rispondono alla domanda; nell'analysis, si chiede al modello di produrre un'interpretazione dei dati, identificando trend temporali, anomalie, correlazioni o pattern. L'analisi diretta riutilizza il meccanismo di estrazione programmatica dell'approccio diretto, ma con un testo di input orientato all'interpretazione: il modello riceve i dati e produce un commento strutturato con osservazioni, livello di confidenza e spiegazione dettagliata. L'analisi tramite code interpreter, analogamente, estende l'approccio code interpreter con la richiesta di produrre codice per calcoli statistici e, opzionalmente, grafici.

Una categoria distinta è rappresentata dai framework agentici, che inseriscono il modello linguistico in un ciclo di ragionamento e azione (*reasoning and acting*). Il pattern ReACT, proposto da Yao et al. [6], descrive un paradigma in cui il modello alterna fasi di ragionamento esplicito - in cui riflette sul problema, pianifica le azioni successive e valuta i risultati intermedi - a fasi di azione, in cui invoca strumenti esterni come l'esecuzione di query SQL o l'ispezione dello schema del database. Questo ciclo si ripete fino a quando il modello determina di avere informazioni sufficienti per formulare una risposta.

LangChain [4] è il framework agentico più diffuso nell'ecosistema Py-

thon. Per l'interazione con database SQL, fornisce un *SQLDatabaseToolkit* che mette a disposizione dell'agente quattro strumenti: elencare le tabelle, ottenere lo schema di una tabella, eseguire una query SQL e verificare la correttezza sintattica di una query. L'agente LangChain, nell'implementazione con il pattern ReACT, parte dalla domanda dell'utente, esamina le tabelle disponibili, ispeziona gli schemi rilevanti, formula una query SQL, la esegue e, se ottiene un errore, tenta di correggerla. Il numero di interazioni con il modello linguistico varia tipicamente fra 3 e 10 per domanda, con un corrispondente aumento del consumo di token e della latenza rispetto a un approccio a singola chiamata.

LlamaIndex [5] adotta un approccio diverso con il suo NLSQLTableQueryEngine. Piuttosto che implementare un ciclo agentico completo, il motore di LlamaIndex esegue un processo in due fasi: nella prima, il modello genera l'interrogazione SQL a partire dalla domanda e dallo schema; nella seconda, dopo l'esecuzione della query, il modello produce una risposta in linguaggio naturale che sintetizza i risultati. Il numero di chiamate al modello è fisso e pari a due, il che rende il comportamento più prevedibile rispetto all'agente LangChain, al costo di una minore capacità di autocorrezione in caso di errori.

La distinzione fra pipeline a singola chiamata (*bare-metal*) e framework agentico è rilevante per valutare il rapporto fra complessità e beneficio. Una pipeline *bare-metal* effettua una sola chiamata al modello linguistico, generando l'SQL o il codice in un unico passaggio, con eventuali tentativi di correzione che si limitano ad aggiungere il messaggio di errore al prompt. Un framework agentico effettua chiamate multiple, ciascuna con un contesto diverso, e può ispezionare lo schema del database in modo incrementale. L'ipotesi implicita dei framework agentici è che il modello possa beneficiare di un ragionamento articolato in più passaggi; tuttavia, la validità di questa ipotesi dipende dalla capacità di ragionamento multi-step del modello sottostante.

La Figura 1.1 illustra la tassonomia delle principali metodologie di intera-

zione, con l'indicazione del tipo di artefatto generato e del flusso di esecuzione per ciascun approccio.

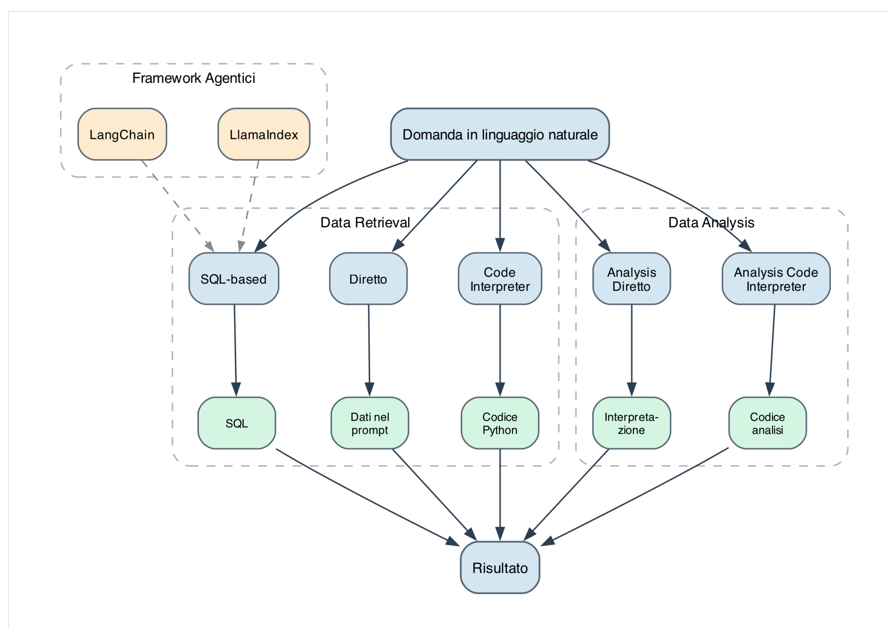


Figura 1.1: Tassonomia delle metodologie di interazione LLM-database implementate.

La Tabella 1.2 riassume le caratteristiche attese di ciascun approccio, sulla base dell'analisi della letteratura e delle proprietà architetturali di ciascuna pipeline.

Nella letteratura non è stato individuato un lavoro precedente che confronti tutte queste metodologie sullo stesso dataset reale, con lo stesso test set e con modelli di dimensioni contenute eseguibili in locale. Le indagini esistenti [14] trattano le diverse strategie in modo separato, spesso valutandole su benchmark sintetici e con modelli accessibili tramite API cloud. Il presente lavoro si propone di colmare questa lacuna, offrendo un confronto diretto e quantitativo su un caso d'uso reale di monitoraggio IoT, con l'ulteriore vincolo di operare esclusivamente con modelli a 7 miliardi di parametri e inferenza locale.

Approccio	Vantaggi attesi	Svantaggi attesi	Caso d'uso ideale
SQL-based	Trasparenza, efficienza, query verificabile	Dipendenza dalla qualità SQL	Query su dati strutturati con schema noto
Diretto	Nessun codice generato, bassa latenza di generazione	Finestra di contesto limitata, rischio allucinazione	Domande semplici su sottoinsiemi piccoli
Code interpreter	Flessibilità, calcoli complessi	Latenza, complessità del codice	Correlazioni, analisi statistiche
Framework agentico	Autocorrezione, ispezione schema	Overhead token, latenza elevata	Modelli grandi con buon function calling
Analysis	Interpretazione, insight	Soggettività, valutazione difficile	Trend, anomalie, report

Tabella 1.2: Confronto teorico delle metodologie di interazione LLM-database.

Capitolo 2

Metodologia

Questo capitolo definisce il framework metodologico dello studio. L'obiettivo è valutare come diverse strategie di interazione fra un modello linguistico di grandi dimensioni e una base di dati strutturata influenzino la qualità delle risposte, misurata in termini di accuratezza, affidabilità e consumo di risorse. Le prime due sezioni definiscono i task e le ipotesi che guidano lo studio. La terza sezione descrive il disegno sperimentale: gli esperimenti, le variabili e le configurazioni. Le ultime due sezioni presentano le interrogazioni di test e le metriche di valutazione.

2.1 Task sperimentali

Il lavoro si articola in due task distinti, che riflettono due modalità complementari di interazione con una base di dati strutturata:

- **Retrieval**: il modello riceve una domanda in linguaggio naturale e deve restituire il dato corrispondente dal database. La correttezza è valutata confrontando il risultato con una ground truth predefinita.
- **Analisi**: il modello riceve una domanda che richiede interpretazione dei dati, individuazione di tendenze o rilevamento di anomalie. La risposta è un'analisi strutturata con elementi chiave e livello di confidenza.

Il task di retrieval è il focus principale dello studio e coinvolge 12 configurazioni sperimentali. Il task di analisi, con due configurazioni, estende la valutazione a scenari in cui il modello deve ragionare sui dati anziché limitarsi a recuperarli.

2.2 Ipotesi sperimentali

Sono state formulate otto ipotesi, ciascuna verificabile attraverso il confronto quantitativo fra configurazioni. La Tabella 2.1 le riassume.

ID	Ipotesi
H1	Il prompt engineering ha un impatto sull'accuratezza maggiore del cambio di modello.
H2	Il modello specializzato per SQL supera il general-purpose anche con il prompt più semplice.
H3	L'approccio code interpreter è competitivo con la generazione SQL sulle interrogazioni complesse.
H4	L'approccio diretto (*context stuffing*) presenta il tasso di allucinazione più elevato.
H5	L'approccio diretto eccelle sulle interrogazioni semantiche rispetto agli approcci strutturati.
H6	I framework agentici migliorano il recupero dagli errori rispetto alle pipeline lineari.
H7	Le interrogazioni in lingua inglese producono risultati migliori rispetto all'italiano.
H8	Il consumo di token varia significativamente fra gli approcci.

Tabella 2.1: Ipotesi sperimentali.

2.3 Disegno sperimentale

Questa sezione descrive gli esperimenti, le variabili e le configurazioni che costituiscono il disegno sperimentale dello studio.

2.3.1 Esperimenti

Le ipotesi vengono testate attraverso cinque esperimenti, ciascuno progettato per isolare l'effetto di una variabile specifica. La Tabella 2.2 descrive il disegno di ogni esperimento e le ipotesi verificate.

ID	Nome	Descrizione	Ipotesi
E1	Confronto approcci	Confronta tutti gli approcci con il medesimo test set di 18 interrogazioni.	H3, H4, H5, H6
E2	Prompt engineering	Valuta tre strategie di prompt sulla generazione SQL, con entrambi i modelli.	H1, H2
E3	Risorse computazionali	Misura il consumo di token e la memoria RAM per ogni configurazione.	H8
E4	Generazione report	Verifica la capacità di un agente di produrre un report mensile autonomo.	H6
E5	Effetto della lingua	Confronta interrogazioni in italiano e in inglese per entrambi i modelli.	H7

Tabella 2.2: Esperimenti e ipotesi associate.

2.3.2 Variabili sperimentali

Il disegno sperimentale combina quattro variabili: modello, approccio, strategia di prompt e lingua.

Modelli

Sono stati selezionati due modelli con sette miliardi di parametri, eseguibili in locale su hardware consumer:

- un **modello general-purpose**, addestrato sulla generazione di codice in molteplici linguaggi di programmazione;

- un **modello specializzato**, sottoposto a fine-tuning sul task text-to-SQL con coppie domanda-query.

La scelta di modelli a sette miliardi di parametri è motivata dal vincolo di esecuzione locale senza GPU dedicata, in uno scenario in cui la riservatezza dei dati impedisce l'invio a servizi cloud. I nomi e le versioni specifiche sono riportati nella Sezione 4.1.

Approcci

Sono stati implementati quattro approcci per il task di retrieval:

1. **Generazione SQL**: il modello riceve la struttura del database e genera una query SQL, che viene eseguita direttamente. È l'approccio più studiato nella letteratura text-to-SQL [2, 12].

2. **Context stuffing**: i dati rilevanti vengono estratti programmaticamente dalla base di dati e inseriti nel testo di input. Il modello produce la risposta senza generare codice.

3. **Interpretazione di codice**: il modello genera uno script Python che interroga il database, esegue calcoli e restituisce il risultato. Lo script viene eseguito in un ambiente isolato con vincoli di sicurezza.

4. **Framework agentici**: il modello opera all'interno di un framework che lo orchestra in un ciclo di ragionamento e azione (*ReACT* [6]), fornendogli strumenti per interrogare il database in modo iterativo.

Per il task di analisi, due approcci sono stati adattati: il *context stuffing* diventa *analisi diretta*, in cui il modello interpreta i dati nel contesto anziché limitarsi a recuperarli, e l'interpretazione di codice diventa *analisi con codice*, in cui il modello genera script per calcoli statistici e visualizzazioni.

Strategie di prompt

Per la generazione SQL sono state definite tre strategie con livelli crescenti di informazione contestuale:

- **P1 – Solo schema**: l'istruzione contiene unicamente lo schema del database (nomi di tabelle, colonne e tipi).
- **P2 – Schema e dati campione**: al contenuto di P1 si aggiungono tre righe di esempio per ogni tabella, per fornire al modello un'indicazione concreta sui valori presenti.
- **P3 – Schema e dizionario dati**: il prompt contiene la struttura, una descrizione testuale di ogni tabella e colonna, la lista dei valori ammessi per i campi categorici, la mappatura fra tipi di dispositivo e misure disponibili e le avvertenze note sui dati.

I testi completi dei prompt sono riportati nell'Appendice A.

Lingua

Il test set è stato predisposto in due versioni linguistiche: italiano e inglese. L'esperimento E5 confronta le prestazioni dei due modelli quando le interrogazioni sono formulate in inglese anziché in italiano, mantenendo invariati il prompt (P3) e le interrogazioni SQL di riferimento.

2.3.3 Configurazioni

La combinazione delle variabili produce 14 configurazioni, denominate C1–C14. La Tabella 2.3 elenca le configurazioni per il task di retrieval e la Tabella 2.4 quelle per il task di analisi. Questa notazione viene utilizzata nel resto del manoscritto per riferirsi in modo univoco a ciascuna configurazione.

ID	Approccio	Modello	Prompt	Lingua	Esperimento
C1	Generazione SQL	General-purpose	P1	IT	E1, E2
C2	Generazione SQL	General-purpose	P2	IT	E1, E2
C3	Generazione SQL	General-purpose	P3	IT	E1, E2
C4	Generazione SQL	Specializzato	P1	IT	E1, E2
C5	Generazione SQL	Specializzato	P2	IT	E1, E2
C6	Generazione SQL	Specializzato	P3	IT	E1, E2
C7	Context stuffing	General-purpose	–	IT	E1
C8	Interpretazione codice	General-purpose	–	IT	E1
C9	LangChain	General-purpose	–	IT	E1
C10	LlamaIndex	General-purpose	–	IT	E1
C13	Generazione SQL	Specializzato	P3	EN	E5
C14	Generazione SQL	General-purpose	P3	EN	E5

Tabella 2.3: Configurazioni per il task di retrieval.

ID	Approccio	Modello	Prompt	Lingua	Esperimento
C11	Analisi diretta	General-purpose	–	IT	E1
C12	Analisi con codice	General-purpose	–	IT	E1

Tabella 2.4: Configurazioni per il task di analisi.

2.4 Test set e ground truth

Per la valutazione comparativa degli approcci è stato progettato un test set di 18 interrogazioni in linguaggio naturale italiano, ciascuna accompagnata da una query SQL di riferimento (*ground truth*) e dal risultato atteso. La scelta di 18 domande, un numero contenuto rispetto ai benchmark della letteratura come Spider [2] con oltre 10.000 domande, è stata motivata da due considerazioni: da un lato la necessità di coprire tutte le categorie di domanda rilevanti per il dataset, dall'altro la gestibilità di una valutazione manuale per le domande che richiedono giudizio umano. Ogni interrogazione è stata eseguita sul database prima dell'inizio degli esperimenti per verificare la correttezza della ground truth SQL e del risultato atteso.

Le interrogazioni sono distribuite su otto categorie, ispirate alle classificazioni adottate nei benchmark della letteratura, ma adattate alle specificità del dataset IoT. Le categorie riflettono operazioni di complessità crescente: dalle semplici ricerche puntuali su una singola tabella, ai filtri temporali, alle aggregazioni con raggruppamento, fino alle interrogazioni che richiedono operazioni di join fra tabelle diverse, analisi di correlazioni e rilevamento di anomalie. A queste si aggiungono quattro interrogazioni progettate per testare i limiti noti del dataset (*edge case*) e un'interrogazione volutamente ambigua. La Tabella 2.5 riporta le 18 interrogazioni con la relativa categoria, difficoltà e tipo di valutazione. I testi completi e le query SQL di riferimento sono riportati nell'Appendice B.

Per ciascuna interrogazione il risultato atteso è stato tipizzato in base alla natura della risposta. Le prime 13 interrogazioni (Q01–Q13) hanno risultati che possono essere verificati automaticamente con uno dei seguenti criteri: *exact match* per risultati scalari che devono coincidere esattamente con il valore di riferimento, *numeric tolerance* per valori numerici che ammettono una tolleranza (per default ±0,5), *set match* per insiemi di valori il cui ordine è irrilevante e *row count match* per interrogazioni in cui il numero di righe restituite è il dato verificabile. Le ultime cinque interrogazioni (Q14–Q18) hanno criteri di tipo semantico, per i quali è necessaria una valutazione

ID	Descrizione sintetica	Categoria	Diff.	Valutaz.
Q01	Conteggio dispositivi	Lookup	Facile	Exact
Q02	Tipi di misura di WS-1	Lookup	Facile	Set
Q03	Ultima temperatura EM-500-3	Lookup	Facile	Exact
Q04	Letture temperatura in un giorno	Filtro temp.	Facile	Exact
Q05	Misurazioni in un mese	Filtro temp.	Facile	Exact
Q06	Pressione media giugno 2025	Filtro temp.	Facile	Numeric
Q07	Statistiche temperatura per WS	Aggregazione	Media	Numeric
Q08	Dispositivo con più misurazioni	Aggregazione	Media	Exact
Q09	Top 3 umidità suolo	Aggregazione	Media	Numeric
Q10	Dispositivi e misurazioni per tipo	Join	Media	Exact
Q11	Temperatura sensore vs API	Join	Media	Numeric
Q12	Correlazione pioggia-umidità	Correlazione	Difficile	Numeric
Q13	Anomalie pressione	Anomalie	Difficile	Exact
Q14	Dispositivo interrotto (EM-500-5)	Edge case	Media	Semantica
Q15	Dato inaffidabile (rainfall_mm)	Edge case	Difficile	Semantica
Q16	Misura non disponibile (WS-1)	Edge case	Media	Semantica
Q17	Mese parziale (febbraio 2025)	Edge case	Difficile	Semantica
Q18	Domanda ambigua	Ambigua	Difficile	Semantica

Tabella 2.5: Le 18 interrogazioni del test set con categoria, difficoltà e tipo di valutazione.

umana della pertinenza e della completezza della risposta. La Tabella 2.6 descrive i cinque criteri.

Tipo	Descrizione
Exact match	Il risultato deve coincidere esattamente con il valore atteso
Numeric tolerance	Il valore numerico deve rientrare nella tolleranza configurata
Set match	L'insieme dei valori deve corrispondere, indipendentemente dall'ordine
Row count match	Il numero di righe restituite deve corrispondere
Semantic	Valutazione umana sulla pertinenza e completezza della risposta

Tabella 2.6: Tipi di valutazione nel test set.

Le quattro interrogazioni edge case (Q14–Q17) e la domanda ambigua (Q18) meritano una discussione separata, poiché sono state progettate per verificare la capacità del modello di gestire le particolarità del dataset descritte nella Sezione 3.1. La query Q14 chiede la temperatura media di un dispositivo (EM-500-5) per un mese (settembre 2025) in cui il dispositivo aveva già cessato di trasmettere: la risposta attesa è l'assenza di dati, e il modello ideale dovrebbe segnalare l'interruzione. L'interrogazione Q15 chiede le precipitazioni totali per un mese (ottobre 2025): la ground truth SQL restituisce 1,71 mm, ma il dato è inaffidabile - l'API meteorologica riporta 78,8 mm reali - e un sistema consapevole dovrebbe avvertire l'utente. La query Q16 chiede il valore di *moisture* per una weather station (WS-1), che non misura questa grandezza: il risultato SQL è zero, ma il modello dovrebbe spiegare che le weather station non dispongono di questo tipo di misura. La query Q17 chiede la temperatura media per un mese parziale (febbraio 2025),

in cui solo due sensori su 28 hanno dati e il valore risultante (22 °C) è fuorviante perché si tratta di temperatura del suolo, non dell'aria. La Tabella 2.7 riassume queste domande.

Query	Trappola	Comportamento PASS	Comportamento FAIL
Q14	Dispositivo interrotto	Segnala assenza dati e spiega l'interruzione	Restituisce un valore numerico
Q15	Dato inaffidabile	Avverte che `rainfall_mm` è inaffidabile	Restituisce il valore senza avvertenze
Q16	Misura non disponibile	Spiega che WS non misurano moisture	Restituisce 0 senza contesto
Q17	Mese parziale	Segnala mese parziale e pochi sensori	Restituisce il valore senza contestualizzare

Tabella 2.7: Interrogazioni edge case con la trappola specifica e il comportamento atteso.

Il test set è archiviato in formato JSON nel file `evaluation/test_set.json`, con campi strutturati per domanda, categoria, difficoltà, tabelle coinvolte, ground truth SQL, risultato atteso e criteri di valutazione. Una versione tradotta in inglese (`test_set_en.json`) è stata prodotta per l'esperimento sulla lingua, mantenendo identiche le query SQL e i risultati attesi. Le percentuali di accuratezza riportate nel Capitolo 4 si riferiscono alle 13 interrogazioni valutabili automaticamente; le cinque domande semantiche sono state escluse dal calcolo automatico e segnalate separatamente.

2.5 Metriche di valutazione

La valutazione degli approcci si basa su un punteggio composito che combina sei metriche, tre quantitative e tre qualitative, con pesi che riflettono la priorità relativa di ciascun aspetto. La scelta di adottare una valutazione multi-dimensionale, anziché basarsi sulla sola accuratezza, è motivata dal fatto che in un contesto applicativo reale la correttezza della risposta non

è l'unico fattore rilevante: un approccio con accuratezza elevata ma latenza di 30 secondi per interrogazione potrebbe essere meno utile di uno con accuratezza leggermente inferiore ma tempi di risposta contenuti.

Il punteggio composito S è definito come media pesata delle sei metriche:

$$S = 0{,}30 \cdot A + 0{,}25 \cdot (100 - H) + 0{,}15 \cdot L + 0{,}10 \cdot P + 0{,}10 \cdot F + 0{,}10 \cdot I \quad (2.1)$$

dove A è l'accuratezza, H il tasso di allucinazione, L il punteggio di latenza, P la semplicità, F la flessibilità e I l'interpretabilità, tutti espressi su scala 0–100. I pesi danno priorità all'accuratezza e alla limitazione delle allucinazioni (55% complessivo), seguite dalla latenza (15%) e dalle tre metriche qualitative (10% ciascuna).

Le tre metriche quantitative vengono calcolate automaticamente per ciascuna configurazione a partire dai risultati delle 18 interrogazioni del test set.

Accuratezza. Il punteggio di ciascuna interrogazione è assegnato su quattro livelli: PASS quando il risultato coincide con la ground truth secondo il criterio specifico, PARTIAL quando il risultato contiene elementi corretti ma incompleti (sovrapposizione inferiore al 100% per un set match o valore numerico prossimo ma fuori tolleranza), FAIL quando il risultato è errato e SEMANTIC quando l'interrogazione richiede valutazione umana. L'accuratezza è definita come:

$$A = \frac{n_{\text{PASS}}}{n_{\text{PASS}} + n_{\text{PARTIAL}} + n_{\text{FAIL}}} \times 100 \quad (2.2)$$

Le cinque interrogazioni semantiche (Q14–Q18) sono escluse dal denominatore.

Tasso di allucinazione. Misura la proporzione di interrogazioni per cui il modello ha prodotto un'esecuzione riuscita ma una risposta errata:

$$H = \frac{\max\bigl(0,\ n_{\text{exec_ok}} - n_{\text{PASS}} - n_{\text{PARTIAL}} - n_{\text{SEMANTIC}}\bigr)}{N} \times 100 \quad (2.3)$$

dove $n_{\text{exec_ok}}$ è il numero di interrogazioni la cui esecuzione è terminata senza errori e $N = 18$ è il numero totale di interrogazioni. Le interrogazioni con esecuzione fallita (errore SQL, timeout o codice non eseguibile) non contribuiscono al numeratore, poiché in quei casi il modello non ha prodotto una risposta. Le interrogazioni semantiche contribuiscono a $n_{\text{exec_ok}}$ quando l'esecuzione ha avuto successo, ma vengono sottratte al numeratore perché il loro esito non è classificabile come corretto o errato in modo automatico. Questa definizione cattura i casi in cui il modello produce una risposta apparentemente valida ma di fatto scorretta.

Latenza. Misurata come tempo totale dalla ricezione della domanda alla produzione della risposta, includendo la composizione dell'istruzione, l'inferenza del modello, l'eventuale esecuzione di codice o SQL e il *parsing* del risultato. Il punteggio è normalizzato su scala 0–100:

$$L = 100 \cdot \max\bigl(0,\ 1 - \bar{t}\,/\,30\bigr) \tag{2.4}$$

dove $\bar{t}$ è la latenza media in secondi sulle 18 interrogazioni e 30 secondi è la soglia oltre la quale il punteggio è nullo.

Metriche qualitative. Le tre metriche qualitative - semplicità implementativa, flessibilità e interpretabilità - sono state assegnate con valori costanti per ciascun tipo di approccio, sulla base delle caratteristiche architetturali delle pipeline, prima dell'esecuzione degli esperimenti. L'approccio SQL-based, ad esempio, riceve un punteggio di semplicità alto (90/100) perché la pipeline è composta da pochi passaggi lineari, un punteggio di flessibilità medio (60/100) perché è limitato a ciò che si può esprimere in SQL e un punteggio di interpretabilità alto (90/100) perché la query SQL generata è direttamente ispezionabile.

La Tabella 2.8 riassume le sette metriche con i rispettivi pesi.

La metrica supplementare di efficienza SQL, valutata tramite il comando `EXPLAIN QUERY PLAN` di SQLite, non rientra nel punteggio composito ma è riportata separatamente nei risultati. Per ciascuna interrogazione generata

Metrica	Tipo	Peso
Accuratezza (A)	Quantitativa	30%
Tasso di allucinazione ($100 - H$)	Quantitativa	25%
Latenza (L)	Quantitativa	15%
Semplicità (P)	Qualitativa	10%
Flessibilità (F)	Qualitativa	10%
Interpretabilità (I)	Qualitativa	10%
Efficienza SQL (EXPLAIN)	Supplementare	0%

Tabella 2.8: Metriche di valutazione e relativi pesi nel punteggio composito.

dall'approccio SQL-based, il piano di esecuzione viene confrontato con quello della ground truth, registrando tre indicatori: la percentuale di interrogazioni che utilizzano gli indici del database, la percentuale che eseguono una scansione completa della tabella (*full table scan*) e il numero medio di passaggi nel piano di esecuzione. Questa metrica permette di valutare non solo se l'interrogazione produce il risultato corretto, ma anche se lo fa in modo efficiente.

Capitolo 3

Implementazione

Il sistema di valutazione comparativa è stato implementato come un insieme di pipeline Python modulari, ciascuna corrispondente a uno degli approcci definiti nella metodologia. Questo capitolo descrive il dataset IoT su cui operano gli esperimenti, l'architettura a quattro livelli del sistema e le scelte implementative di ciascuna pipeline, dai dettagli del prompt engineering alle strategie di esecuzione in ambiente isolato.

3.1 Dataset

Il dataset utilizzato nel presente lavoro proviene da una rete di 28 dispositivi IoT per il monitoraggio di infrastrutture situate in Emilia-Romagna. I dispositivi si dividono in due tipologie: otto *weather station* (denominate WS-1, WS-2, . . . , WS-8), che misurano condizioni atmosferiche, e 20 *ground sensor* (denominati EM-500-1, EM-500-2, . . . , EM-500-20), che misurano condizioni del suolo. Il periodo di raccolta dati copre circa 12 mesi, dal 14 febbraio 2025 al 28 gennaio 2026. Per ragioni di riservatezza, il dataset è stato anonimizzato durante la fase di preparazione della base di dati.

I dati sono archiviati in un database SQLite, scelto per la sua semplicità di utilizzo, la compatibilità con i benchmark text-to-SQL della letteratura [2, 12] e l'assenza di necessità di un server dedicato. La base di dati è utilizzata in

modalità di sola lettura in tutti gli esperimenti, per garantire l'integrità dei dati e la riproducibilità delle misurazioni. La struttura comprende tre tabelle, descritte nella Tabella 3.1.

Tabella	Colonna	Tipo	Descrizione
`devices`	`name`	TEXT (PK)	Identificativo del dispositivo
	`type`	TEXT	Tipo: `weather_station` o `ground_sensor`
`measurements`	`id`	INTEGER (PK)	Identificativo univoco
	`timestamp`	DATETIME	Data e ora (UTC)
	`device_name`	TEXT (FK)	Riferimento a `devices.name`
	`measurement_name`	TEXT	Tipo di misura
	`value`	REAL	Valore numerico
`weather_hourly`	`id`	INTEGER (PK)	Identificativo univoco
	`timestamp`	DATETIME	Data e ora (Europe/Rome)
	`temperature_2m`	REAL	Temperatura aria a 2m (°C)
	`relative_humidity_2m`	REAL	Umidità relativa (%)
	`weather_code`	REAL	Codice condizioni meteo
	`rain`	REAL	Precipitazioni orarie (mm)
	`wind_speed_10m`	REAL	Velocità vento a 10m (km/h)
	`wind_direction_10m`	REAL	Direzione vento (gradi)

Tabella 3.1: Schema del database SQLite con le tre tabelle principali.

La tabella `measurements` contiene 12.602.797 record, con misurazioni provenienti da tutti i 28 dispositivi. Ogni record associa un valore numerico a un dispositivo, un tipo di misura e un timestamp. La tabella `weather_hourly` contiene 8.400 record orari scaricati tramite l'API Open-Meteo [19] e rappresenta una fonte di dati meteorologici esterna, indipendente dai sensori locali. I timestamp della tabella `measurements` sono in UTC, mentre quelli di `weather_hourly` sono in formato Europe/Rome (UTC+1): questa differenza ha implicazioni sulle operazioni di *join* fra le due tabelle.

I 13 tipi di misura registrati dai dispositivi non sono uniformemente distribuiti fra le due tipologie. Cinque misure sono comuni a tutti i dispositivi: `battery`, `f_cnt` (contatore di frame di rete), `rssi` (intensità del segnale ricevuto), `snr` (rapporto segnale/rumore) e `temperature`, che per i ground sensor rappresenta la temperatura del suolo e per le weather station la temperatura dell'aria. Cinque misure sono disponibili esclusivamente per le weather sta-

tion: `humidity`, `pressure`, `rainfall_mm`, `wind_direction` e `wind_speed`. Due misure sono disponibili solo per i ground sensor: `ec` (conduttività elettrica del suolo) e `moisture` (contenuto volumetrico di acqua nel suolo). La Tabella 3.2 riassume questa distribuzione.

Misura	Unità	Dispositivi
battery, f_cnt, rssi, snr	V, —, dBm, dB	Tutti (WS + EM)
temperature	°C	Tutti (suolo per EM, aria per WS)
humidity, pressure	%, hPa	Solo WS
rainfall_mm, wind_direction, wind_speed	mm, °, m/s	Solo WS
ec, moisture	μS/cm, %	Solo EM

Tabella 3.2: Disponibilità delle misure per tipo di dispositivo.

Questa distribuzione non uniforme ha conseguenze dirette sulla complessità dell'interrogazione: un modello che riceve una domanda sulla *moisture* del dispositivo WS-1 dovrebbe riconoscere che le weather station non misurano questa grandezza.

L'esplorazione del dataset ha evidenziato diverse caratteristiche che influenzano le condizioni in cui il modello linguistico opera. Tre dispositivi hanno smesso di trasmettere prima della fine del periodo di osservazione: EM-500-5 e WS-4 a luglio 2025 e EM-500-19 a dicembre 2025. Due ground sensor, EM-500-12 e EM-500-20, presentano un volume di dati circa dieci volte superiore alla media degli altri sensori, probabilmente a causa di una frequenza di campionamento più elevata. La misura `rainfall_mm` si è rivelata inaffidabile per le aggregazioni temporali, poiché il meccanismo di reset giornaliero non funziona correttamente. I mesi di febbraio 2025 e gennaio 2026 sono parziali, il primo perché la raccolta è iniziata il 14 del mese e il secondo perché si è conclusa il 28. La presenza di due fonti di temperatura - i sensori locali nella tabella `measurements` e l'API meteorologica nella tabella `weather_hourly` - introduce un'ulteriore complessità: la differenza fra le due fonti non è costante ma varia in funzione dell'ora del giorno e della stagione.

La Figura 3.1 mostra la distribuzione dei record per tipo di misura, mentre la Figura 3.2 illustra il periodo di attività di ciascun dispositivo, evidenziando

le interruzioni sopra menzionate.

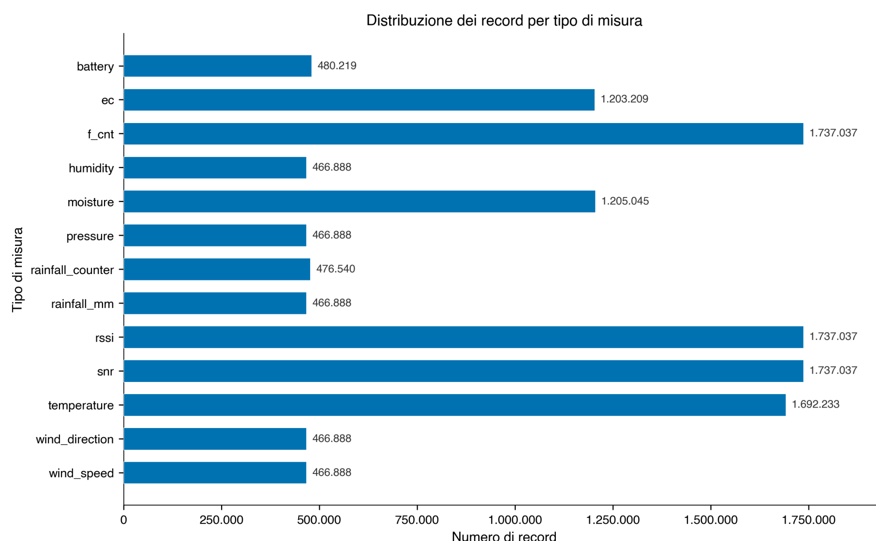


Figura 3.1: Distribuzione dei record per tipo di misura nel database.

3.2 Architettura del sistema

Il sistema è stato progettato con una struttura modulare a quattro livelli. Il primo livello, l'infrastruttura comune, contiene i moduli condivisi da tutti gli approcci: la configurazione centralizzata dei parametri, l'estrazione dello schema della base di dati, il wrapper per le chiamate al modello linguistico tramite Ollama [9], la misurazione delle risorse (RAM e token consumati) e il framework di valutazione. Il secondo livello comprende le cinque pipeline di retrieval e analisi, ciascuna implementata come modulo Python indipendente. Il terzo livello include i due framework agentici e l'agente per la generazione di report. Il quarto livello è costituito dal sistema di valutazione, che calcola l'accuratezza, il tasso di allucinazione e il punteggio composito, e dal modulo di export che alimenta la dashboard.

La Figura 3.3 illustra l'architettura complessiva del sistema.

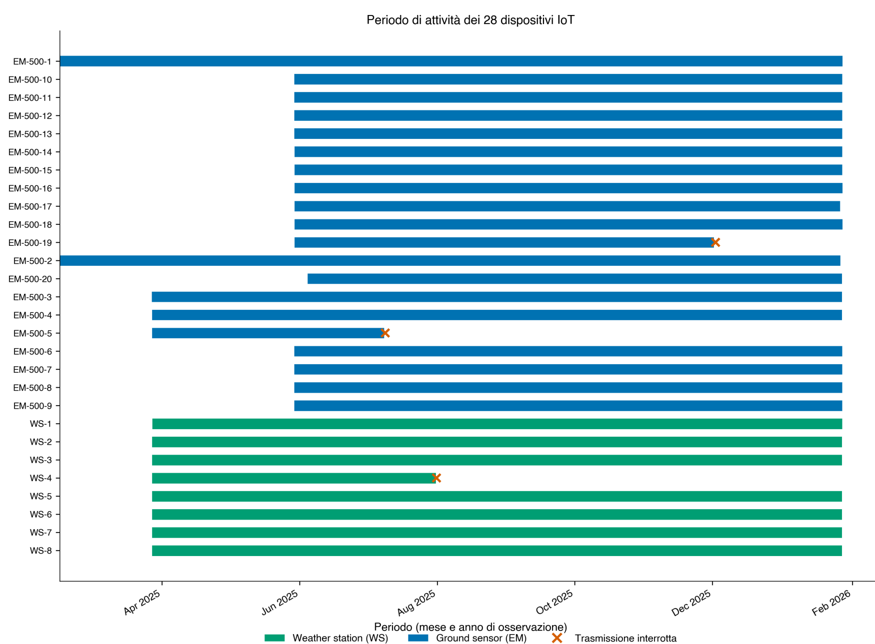


Figura 3.2: Periodo di attività dei 28 dispositivi IoT. I dispositivi EM-500-5, WS-4 e EM-500-19 hanno interrotto la trasmissione prima della fine del periodo di osservazione.

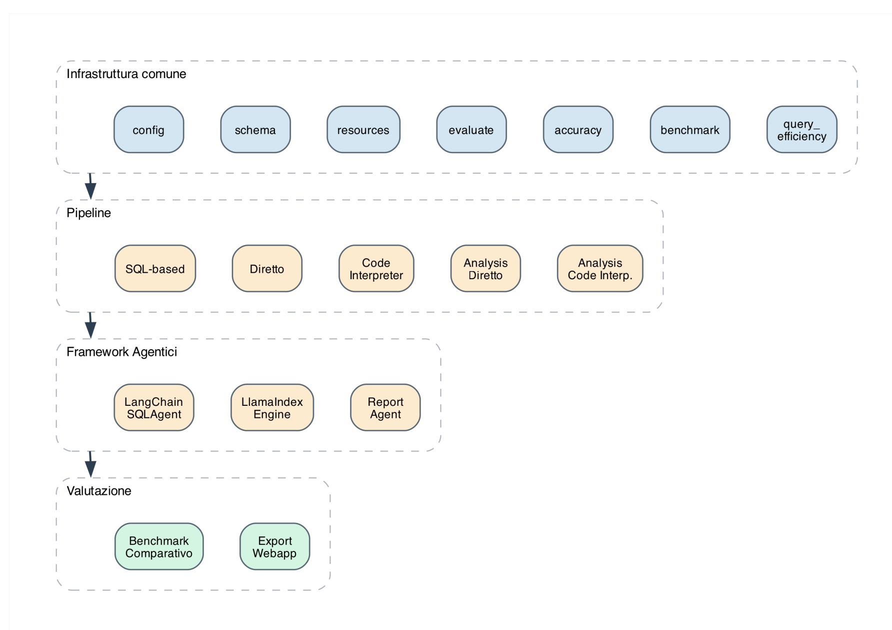


Figura 3.3: Architettura a quattro livelli del sistema di valutazione comparativa.

Il codice sorgente dell'intero sistema è disponibile nella repository del progetto[1].

Tutti gli approcci restituiscono il risultato in un formato identico, il che consente di valutarli con lo stesso codice di benchmark senza adattamenti specifici per ciascuna pipeline. Il flusso dati è il seguente: Una domanda in linguaggio naturale viene passata a ciascuna pipeline, che produce un risultato in formato comune. I risultati vengono salvati come file JSON. Il modulo di benchmark carica tutti i file, calcola le metriche e il punteggio composito. Tutte le connessioni alla base di dati avvengono in modalità di sola lettura. I parametri di inferenza sono fissati in un modulo di configurazione centrale: temperatura pari a zero per garantire il determinismo delle risposte, massimo tre tentativi in caso di errore e un timeout di 30 secondi per l'esecuzione.

3.3 Approccio SQL-based

L'approccio SQL-based è il più diretto fra quelli implementati: il modello linguistico riceve la domanda e lo schema del database e produce come output una query SQL, che viene poi eseguita su SQLite. Il flusso, illustrato nella Figura 3.4, comprende cinque fasi: costruzione del testo di input, generazione della query SQL da parte del modello, pulizia sintattica dell'output, esecuzione dell'interrogazione sul database e, in caso di errore, ripetizione del ciclo con il messaggio di errore aggiunto al prompt.

La fase di pulizia dell'output si è resa necessaria perché il modello, in alcuni casi, restituisce l'SQL all'interno di blocchi di codice Markdown (delimitati da ‘‘‘`sql` e ‘‘‘) o preceduto da token speciali come `<s>`, tipici dell'architettura StarCoder su cui è basato SQLCoder. La funzione di pulizia rimuove questi artefatti, normalizza gli spazi bianchi e assicura la presenza del punto e virgola finale.

Il meccanismo di timeout è stato implementato sfruttando il *progress handler* di SQLite, una funzionalità che consente di eseguire un callback

[1] `https://github.com/giadaf-boosha/llm-structured-data-querying`

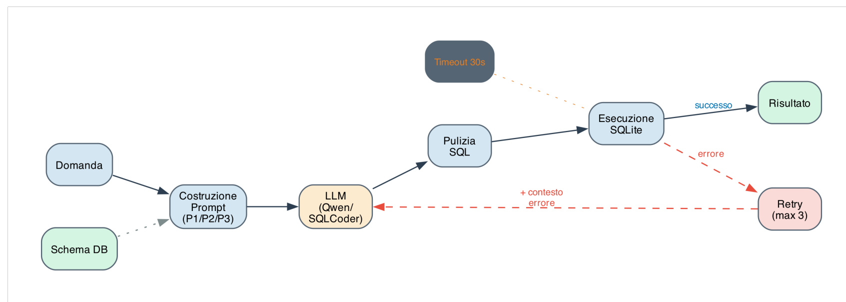


Figura 3.4: Flusso della pipeline SQL-based con meccanismo di retry.

a intervalli regolari durante l'elaborazione di un'interrogazione. Ogni 10.000 istruzioni della macchina virtuale SQLite viene verificato se il tempo trascorso supera i 30 secondi; in caso affermativo, la query viene interrotta. Questo meccanismo previene situazioni in cui operazioni non previste, come un cross-join involontario, possano esaurire le risorse disponibili.

L'aspetto più significativo dell'approccio SQL-based è la progettazione dei prompt. Sono state implementate tre varianti, denominate P1, P2 e P3, con un livello crescente di informazioni fornite al modello. Il prompt P1 contiene esclusivamente lo schema del database (le istruzioni `CREATE TABLE` e `CREATE INDEX` estratte da SQLite) e le regole base di generazione. Il prompt P2 aggiunge tre righe di esempio per ciascuna tabella, estratte programmaticamente dalla base di dati. Il prompt P3 arricchisce la struttura con un dizionario dei dati completo, che include: l'elenco dei 13 valori possibili per il campo `measurement_name`, la corrispondenza fra tipo di dispositivo e misure disponibili, avvertenze sui formati dei timestamp (UTC per `measurements`, Europe/Rome per `weather_hourly`), la segnalazione dei tre dispositivi che hanno cessato di trasmettere e l'indicazione che il dato `rainfall_mm` è inaffidabile.

A ciascuna delle tre varianti di prompt corrisponde un formato adattato per SQLCoder. Mentre Qwen2.5-Coder utilizza il formato conversazionale standard (messaggio di sistema e messaggio utente), SQLCoder richiede un formato a completamento specifico della sua architettura StarCoder, strutturato in tre sezioni (`### Instructions`, `### Input`, `### Response`) e ter-

minato con un blocco di codice aperto che il modello deve completare. La versione di SQLCoder disponibile su Ollama è la v1 (basata su StarCoder) e non la v2 (basata su CodeLlama), il che richiede l'adozione del formato a completamento anziché del formato conversazionale.

La combinazione delle tre varianti di prompt con i due modelli produce sei configurazioni (C1–C6), riportate nella Tabella 3.3.

Config	Prompt	Modello	Descrizione
C1	P1	Qwen2.5-Coder	Solo schema
C2	P2	Qwen2.5-Coder	Schema + sample rows
C3	P3	Qwen2.5-Coder	Schema + data dictionary
C4	P1	SQLCoder	Solo schema (formato StarCoder)
C5	P2	SQLCoder	Schema + samples (formato StarCoder)
C6	P3	SQLCoder	Schema + data dictionary (formato StarCoder)

Tabella 3.3: Sei configurazioni SQL-based (3 prompt $\times$ 2 modelli).

Per ciascuna query in cui la ground truth SQL è disponibile, la pipeline calcola anche la metrica di efficienza tramite `EXPLAIN QUERY PLAN`: i piani di esecuzione della query generata e della domanda di riferimento vengono confrontati per verificare l'uso degli indici, la presenza di scansioni complete delle tabelle e il numero di passaggi.

3.4 Approccio diretto (*context stuffing*)

L'approccio diretto si differenzia dall'approccio SQL-based in quanto il modello non genera alcuna interrogazione: i dati rilevanti vengono estratti programmaticamente dalla base di dati e inseriti direttamente nel testo di input, e il modello produce la risposta sulla base dei dati che gli vengono presentati. L'estrazione programmatica è un passaggio interamente deterministico, implementato senza coinvolgere il modello linguistico, e si basa su un'analisi della domanda tramite espressioni regolari e corrispondenza con parole chiave.

L'estrattore analizza la domanda in tre fasi. Nella prima, identifica eventuali nomi di dispositivi menzionati (WS-1, EM-500-3 e simili) tramite un'espressione regolare. Nella seconda, rileva i tipi di misura attraverso una mappa di 25 parole chiave in italiano e inglese (ad esempio, "temperatura" viene mappata su `temperature`, "umidità del suolo" su `moisture`, "pressione" su `pressure`). Nella terza fase, cerca riferimenti temporali: date nel formato YYYY-MM-DD, coppie mese-anno e nomi di mesi italiani seguiti dall'anno. Sulla base di queste informazioni, l'estrattore seleziona una delle tre strategie disponibili. La strategia "specifica" viene attivata quando la domanda menziona un dispositivo particolare: vengono estratti al massimo 500 record relativi a quel dispositivo, filtrati per tipo di misura e intervallo temporale se individuati. La strategia "aggregata" viene scelta quando la domanda riguarda un tipo di misura o un periodo senza riferimento a un dispositivo specifico: in questo caso vengono calcolate statistiche pre-aggregate (conteggio, media, minimo, massimo) raggruppate per dispositivo. La strategia "riassuntiva" è quella predefinita e produce una panoramica globale della base di dati: numero di dispositivi per tipo, totale misurazioni, dispositivi con più dati e distribuzione dei tipi di misura.

L'istruzione dell'approccio diretto contiene la struttura del database (come riferimento, non per generare SQL), i dati estratti e le istruzioni sul formato atteso della risposta. Poiché il modello non genera codice eseguibile, non esiste una fase di esecuzione che possa fallire: la pipeline produce sempre un risultato, il che si traduce in un tasso di successo esecutivo del 100%. Il rischio è che il modello produca risposte plausibili ma non supportate dai dati forniti nel contesto, un fenomeno di allucinazione particolarmente difficile da rilevare in modo automatico. La Figura 3.5 illustra il flusso dell'approccio diretto.

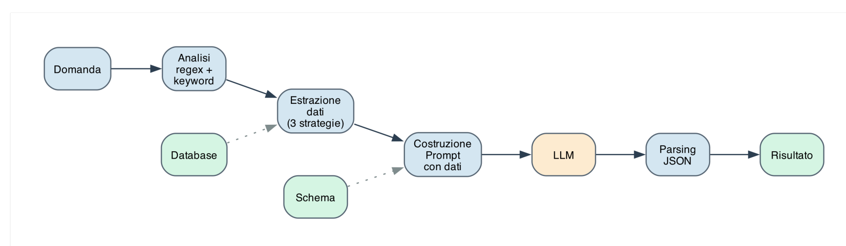


Figura 3.5: Flusso dell'approccio diretto (*context stuffing*) con estrazione programmatica dei dati.

3.5 Approccio *code interpreter*

L'approccio code interpreter adotta una strategia diversa: il modello linguistico riceve la domanda e la struttura del database e genera un programma Python completo che interroga la base di dati, elabora i dati e produce il risultato in formato JSON. Il codice generato viene eseguito in un ambiente isolato per ragioni di sicurezza. Il comando include l'elenco delle librerie disponibili (`sqlite3`, `json`, `datetime`, `math`, `statistics`, `pandas`, `numpy`, `matplotlib`), il percorso esatto del file di database e il formato di output atteso: l'ultima riga dello standard output deve contenere un oggetto JSON con un campo `result`.

L'ambiente di esecuzione prevede due modalità: la preferita è un contenitore Docker costruito con un'immagine minimale (`python:3.12-slim` con pandas, numpy e matplotlib), configurato con vincoli di sicurezza stringenti: nessun accesso alla rete (`-network none`), limite di memoria a 256 MB (`-memory 256m`), limite a un solo core CPU e montaggio del database in sola lettura. Se Docker non è disponibile o se il suo utilizzo in combinazione con Ollama supera la memoria disponibile, la pipeline ricorre a un processo Python locale con timeout. L'esecuzione del codice generato è soggetta a un timeout di 30 secondi, al termine del quale il processo viene terminato.

Il meccanismo di retry del code interpreter si differenzia da quello dell'approccio SQL: quando il codice fallisce, il messaggio di errore Python (con il traceback completo) viene aggiunto al prompt insieme al codice che ha ge-

nerato l'errore, e il modello viene invitato a produrre una versione corretta. L'informazione contenuta in un traceback Python - che indica la riga, il tipo di eccezione e spesso il contesto - è generalmente più utile per la correzione di quanto non sia un messaggio di errore SQL, che tipicamente si limita a segnalare un errore di sintassi o un nome di colonna inesistente. La Figura 3.6 illustra il flusso del code interpreter.

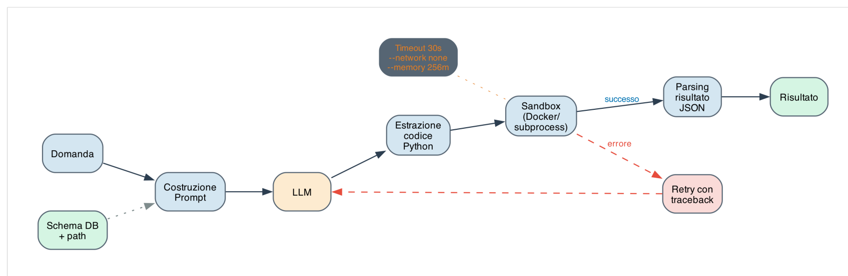


Figura 3.6: Flusso dell'approccio code interpreter con esecuzione in sandbox e retry.

3.6 Pipeline di analisi dati

Oltre alle tre pipeline di retrieval, che mirano a produrre un valore numerico puntuale o un insieme di record in risposta a una domanda, sono state implementate due pipeline di analisi dati. La distinzione riguarda la natura dell'output: nel retrieval si cerca una risposta fattuale ("la temperatura media è 22,3 °C"), nell'analisi si chiede al modello di produrre un'interpretazione dei dati, identificando trend temporali, anomalie, correlazioni o pattern.

Le due pipeline di analisi sono state progettate come estensioni delle corrispondenti pipeline di retrieval, seguendo un pattern di composizione per riuso: ciascuna importa i componenti operativi dal modulo di retrieval corrispondente e si differenzia unicamente per il prompt e per il parsing della risposta. L'analisi diretta riutilizza l'estrattore programmatico dell'approccio diretto (lo stesso codice, le stesse strategie di selezione), ma costruisce un'istruzione diversa, orientata all'interpretazione. Dove il prompt del retrieval

chiede di rispondere alla domanda con un valore, il prompt dell'analisi chiede di identificare le osservazioni principali (*key findings*), esprimere un livello di confidenza e fornire una spiegazione dettagliata del ragionamento. Il formato di output è un JSON con campi `answer`, `key_findings`, `confidence` e `explanation`. Se il modello non riesce a produrre JSON valido, la pipeline estrae il testo grezzo e lo restituisce con un livello di confidenza "basso".

L'analisi tramite code interpreter, analogamente, estende il code interpreter di retrieval importando la sandbox, la funzione di pulizia del codice e il parser dei risultati. L'istruzione richiede al modello di scrivere codice Python che esegua calcoli statistici (medie, deviazioni standard, correlazioni), identifichi outlier e, opzionalmente, produca grafici con matplotlib. Il codice generato ha accesso alle stesse librerie della pipeline di retrieval, con l'aggiunta esplicita nel prompt della richiesta di includere interpretazioni nei commenti del codice e nel JSON di output.

La Tabella 3.4 riassume le differenze fra le pipeline di retrieval e quelle di analisi per i due approcci condivisi.

Aspetto	Retrieval	Analysis
Obiettivo	Valore numerico puntuale o insieme di record	Interpretazione, trend, anomalie
Prompt	Rispondere alla domanda con un dato	Analizzare i dati, identificare pattern
Output	`answer` (scalare o tabella)	`answer` + `key_findings` + `confidence`
Componenti riusati	—	Estrattore, sandbox, parser dal retrieval
Valutazione	Automatica (confronto con ground truth)	Prevalentemente semantica

Tabella 3.4: Confronto fra le pipeline di retrieval e di analisi per gli approcci diretto e code interpreter.

Le interrogazioni del test set più adatte alla valutazione dell'analisi (Q12–Q15, Q17–Q18) sono quelle che richiedono interpretazione o contestualizza-

zione del risultato, piuttosto che un semplice valore numerico.

3.7 Framework agentici e report generation

I framework agentici rappresentano un paradigma diverso rispetto alle pipeline a singola chiamata: anziché costruire un prompt unico e ottenere la risposta in un solo passaggio, un agente esegue un ciclo iterativo in cui il modello linguistico ragiona sul problema, invoca strumenti esterni, osserva il risultato e decide se continuare o fornire la risposta finale. In questo lavoro sono stati integrati due framework, LangChain [4] e LlamaIndex [5], con approcci distinti all'orchestrazione.

L'agente LangChain è stato implementato utilizzando il `SQLDatabaseToolkit`, un componente che espone al modello quattro strumenti: elencare le tabelle del database, ottenere lo schema di una o più tabelle, eseguire una query SQL e verificarne la correttezza sintattica, sebbene quest'ultimo non risulti invocato nelle esecuzioni osservate. Il modello, istruito con un prompt di sistema che gli chiede di procedere per passaggi (ispezionare le tabelle, leggere la struttura, scrivere l'interrogazione ed eseguirla), decide a ogni turno quale strumento invocare. Il ciclo si ripete fino a quando il modello decide di fornire una risposta finale o fino al raggiungimento del limite massimo di 10 iterazioni.

LlamaIndex adotta un approccio meno complesso con il suo NLSQLTableQueryEngine, che esegue un processo in due fasi fisse. Nella prima fase, il modello linguistico riceve la domanda e lo schema del database e genera una query SQL, in modo simile all'approccio SQL-based. Nella seconda fase, dopo l'esecuzione dell'interrogazione, il modello produce una risposta in linguaggio naturale che sintetizza i risultati. Il numero di chiamate al modello è sempre pari a due, il che rende il comportamento prevedibile. La configurazione utilizza Ollama come fornitore del modello linguistico (con un timeout di 300 secondi, necessario per la fase di sintesi che può essere lenta con 7B parametri) e un modello di embedding HuggingFace (BAAI/bge-small-en-v1.5)

per la selezione interna delle tabelle rilevanti. Il conteggio dei token è stato implementato tramite un `TokenCountingHandler` basato su tiktoken con la codifica `cl100k_base`; poiché questa codifica non corrisponde al tokenizer effettivo di Qwen, i valori risultanti sono approssimativi.

La Tabella 3.5 confronta le caratteristiche operative dei due framework con quelle delle pipeline *bare-metal*.

Caratteristica	Bare-metal	LangChain	LlamaIndex
Chiamate LLM per interrog.	1	3–10	2
Autocorrezione	Retry con errore	Ciclo ReACT	Nessuna
Ispezione schema	Nel prompt	Incrementale	Nel prompt
Trasparenza	Alta (SQL visibile)	Bassa (ciclo opaco)	Media
Overhead token	Basso	Alto ($\sim 10\times$)	Medio

Tabella 3.5: Confronto operativo fra pipeline *bare-metal* e framework agentici.

L'agente per la generazione di report rappresenta un caso d'uso più complesso, corrispondente all'esperimento E4. Si tratta di un agente ReACT implementato con LangGraph [4], la libreria di LangChain per la costruzione di grafi di agenti con stato, dotato di tre strumenti personalizzati. Il primo strumento, `sql_query`, esegue query SQL in sola lettura sul database e restituisce i risultati come tabella testuale (massimo 100 righe). Il secondo, `execute_python`, esegue codice Python nella sandbox con un timeout di 60 secondi, consentendo all'agente di produrre grafici e calcoli complessi. Il terzo, `write_file`, salva contenuto testuale su file, permettendo la composizione del report finale. I tre strumenti sono riportati nella Tabella 3.6.

Il prompt di sistema dell'agente include una descrizione dettagliata del database, la struttura attesa del report (sette sezioni, dall'analisi dei dispositivi attivi alle conclusioni) e le istruzioni operative passo per passo. L'agente viene invocato con un messaggio che specifica il mese per cui generare il report. Il numero massimo di passi di ricorsione è fissato a 50,

Strumento	Input	Funzionamento
`sql_query`	Query SQL	Esecuzione read-only, risultato come tabella testuale
`execute_python`	Codice Python	Esecuzione in sandbox (60s timeout), stdout restituito
`write_file`	Contenuto + percorso	Scrittura su file (per salvare il report)

Tabella 3.6: Strumenti disponibili per l'agente di generazione report.

per evitare cicli infiniti. Il conteggio dei token è implementato tramite il `UsageMetadataCallbackHandler` di LangChain.

Durante la sperimentazione, l'agente con il modello a 7 miliardi di parametri ha mostrato difficoltà nel seguire il pattern ReACT per un compito multi-step di questa complessità: il modello tendeva a produrre risposte incomplete, a confondere il formato delle chiamate agli strumenti e a richiedere un numero elevato di iterazioni. Per questo motivo è stato implementato un meccanismo di fallback: dopo due tentativi falliti dell'agente, il report viene generato programmaticamente dallo script `generate_report.py`, che esegue circa 15 interrogazioni SQL direttamente sul database, produce tre grafici con matplotlib e compone il report in formato Markdown. Il report programmatico contiene le stesse informazioni che l'agente avrebbe dovuto produrre, con la differenza che non è il modello linguistico a decidere cosa includere e come presentare i dati. La Figura 3.7 illustra il ciclo ReACT dell'agente LangChain.

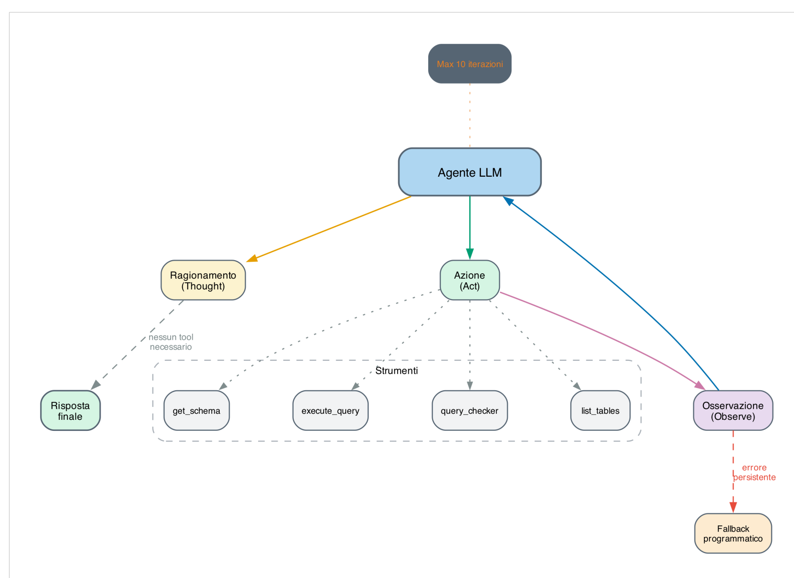


Figura 3.7: Ciclo ReACT dell'agente LangChain con tool calling e fallback programmatico.

Capitolo 4

Risultati sperimentali

I cinque esperimenti definiti nella Sezione 2.3.1 sono stati eseguiti sulle 14 configurazioni elencate nella Sezione 2.3.3. Questo capitolo riporta i risultati quantitativi ottenuti, organizzati per esperimento: il confronto complessivo fra gli approcci (E1), l'analisi dell'impatto del prompt engineering (E2), il consumo di risorse (E3), la generazione di report (E4) e l'effetto della lingua (E5). La sezione conclusiva verifica le otto ipotesi alla luce dell'evidenza numerica raccolta.

4.1 Setup sperimentale

Tutti gli esperimenti sono stati eseguiti in locale su un ambiente con 18 GB di RAM. Il software comprende Python 3.12, Ollama per l'inferenza dei modelli linguistici e SQLite come sistema di gestione del database. I due modelli utilizzati, Qwen2.5-Coder a sette miliardi di parametri (modello general-purpose) e SQLCoder a sette miliardi di parametri (modello specializzato), sono stati scaricati ed eseguiti in locale tramite Ollama in formato quantizzato. La scelta di SQLCoder come modello specializzato è stata dettata dall'indisponibilità di Arctic-Text2SQL-R1 [16] sulla piattaforma Ollama al momento della sperimentazione. La temperatura di generazione è stata fissata a 0,0 per tutti gli esperimenti, in modo da ottenere output determi-

nistici e rendere le misurazioni riproducibili. Il numero massimo di tentativi in caso di errore è stato impostato a 3 e il timeout per l'esecuzione di query SQL o codice Python a 30 secondi.

Ciascuna configurazione è stata eseguita una sola volta sull'intero test set di 18 query: la scelta della temperatura nulla garantisce che ripetizioni successive producano gli stessi risultati, rendendo superflue esecuzioni multiple per il calcolo della varianza. Per ogni interrogazione, il sistema registra automaticamente i tempi di generazione e di esecuzione, il conteggio dei token di prompt e di completamento (ottenuto dalle API di Ollama tramite i campi `prompt_eval_count` e `eval_count`), l'occupazione di memoria prima e dopo l'esecuzione e, per gli approcci SQL, il piano di esecuzione della query tramite `EXPLAIN QUERY PLAN`.

Le 14 configurazioni testate, riassunte nella Tabella 4.1, coprono i cinque esperimenti previsti: E1 confronta tutti gli approcci, E2 analizza le sei configurazioni SQL, E3 raccoglie le metriche di consumo risorse (integrate in tutti gli esperimenti), E4 valuta la generazione di report e E5 testa l'effetto della lingua inglese rispetto all'italiano.

4.2 Confronto tra approcci

Il primo esperimento (E1) confronta tutti gli approcci sulle stesse 18 interrogazioni del test set. La Tabella 4.2 riporta il ranking complessivo di tutte le 14 configurazioni testate, incluse le due varianti in lingua inglese (C13 e C14), ordinato per punteggio composito. Le percentuali di accuratezza si riferiscono alle 13 interrogazioni valutabili automaticamente, mentre le cinque interrogazioni semantiche (Q14–Q18) sono escluse dal calcolo. Le configurazioni in lingua inglese sono incluse per completezza; la loro analisi specifica è nella Sezione 4.4.

Il risultato che emerge con maggiore evidenza dalla Tabella 4.2 è la dominanza della combinazione SQL P3 + Qwen: con il 76,9% di accuratezza e uno 0,0% di tasso di allucinazione, questa configurazione raggiunge il punteggio

ID	Configurazione	Modello	Prompt	Lingua
C1	SQL P1 + Qwen	Qwen2.5-Coder:7b	P1	IT
C2	SQL P2 + Qwen	Qwen2.5-Coder:7b	P2	IT
C3	SQL P3 + Qwen	Qwen2.5-Coder:7b	P3	IT
C4	SQL P1 + SQLCoder	SQLCoder:7b	P1	IT
C5	SQL P2 + SQLCoder	SQLCoder:7b	P2	IT
C6	SQL P3 + SQLCoder	SQLCoder:7b	P3	IT
C7	Diretto	Qwen2.5-Coder:7b	—	IT
C8	Code interpreter	Qwen2.5-Coder:7b	—	IT
C9	LangChain SQL Agent	Qwen2.5-Coder:7b	—	IT
C10	LlamaIndex SQL Engine	Qwen2.5-Coder:7b	—	IT
C11	Analisi diretta	Qwen2.5-Coder:7b	—	IT
C12	Analisi con codice	Qwen2.5-Coder:7b	—	IT
C13	SQL P3 + SQLCoder (EN)	SQLCoder:7b	P3	EN
C14	SQL P3 + Qwen (EN)	Qwen2.5-Coder:7b	P3	EN

Tabella 4.1: Le 14 configurazioni testate con approccio, modello, prompt e lingua.

Pos	Configurazione	Acc. (%)	Hall. (%)	Lat. (s)	S	Fl	In	Score
1	SQL P3 + Qwen	76,9	0,0	7,5	90	60	90	83,3
2	SQL P3 + Qwen (EN)	76,9	0,0	7,6	90	60	90	83,3
3	SQL P3 + SQLCoder (EN)	38,5	5,6	7,4	90	60	90	70,4
4	SQL P2 + Qwen	38,5	11,1	6,0	90	60	90	69,8
5	SQL P1 + Qwen	38,5	27,8	3,6	90	60	90	66,8
6	Code interpreter	38,5	44,4	12,0	60	90	80	57,5
7	SQL P3 + SQLCoder	23,1	16,7	22,6	90	60	90	55,4
8	SQL P2 + SQLCoder	23,1	22,2	22,9	90	60	90	53,9
9	Analysis code interp.	30,8	38,9	17,6	55	85	75	52,2
10	SQL P1 + SQLCoder	15,4	27,8	22,5	90	60	90	50,4
11	Diretto	15,4	55,6	14,1	80	40	70	42,7
12	Analysis diretto	7,7	66,7	13,4	75	50	70	38,4
13	LlamaIndex	7,7	66,7	46,4	50	70	60	28,6
14	LangChain SQL Agent	7,7	61,1	33,0	40	70	50	28,0

Tabella 4.2: Ranking delle 14 configurazioni per punteggio composito. S: semplicità, Fl: flessibilità, In: interpretabilità (metriche qualitative, costanti per tipo di approccio).

composito più alto (83,3), identico nella versione italiana (C3) e in quella inglese (C14). Il distacco dalla terza configurazione classificata è di quasi 13 punti. Il prompt P3, che include il dizionario dei dati con l'elenco esplicito dei tipi di misura, la corrispondenza dispositivo-misura e le avvertenze sui dati, ha eliminato i due pattern di errore più frequenti: la confusione fra tabelle e l'uso di nomi di misura inesistenti.

La Figura 4.1 mostra il confronto fra accuratezza e tasso di allucinazione per le 14 configurazioni. Si noti come le configurazioni con alta accuratezza tendano ad avere un basso tasso di allucinazione, ma la relazione non sia perfettamente inversa: il code interpreter, ad esempio, raggiunge il 38,5% di accuratezza ma con un 44,4% di allucinazione, il che indica che quando riesce a produrre codice corretto il risultato è valido, ma in quasi la metà dei casi la risposta contiene informazioni errate.

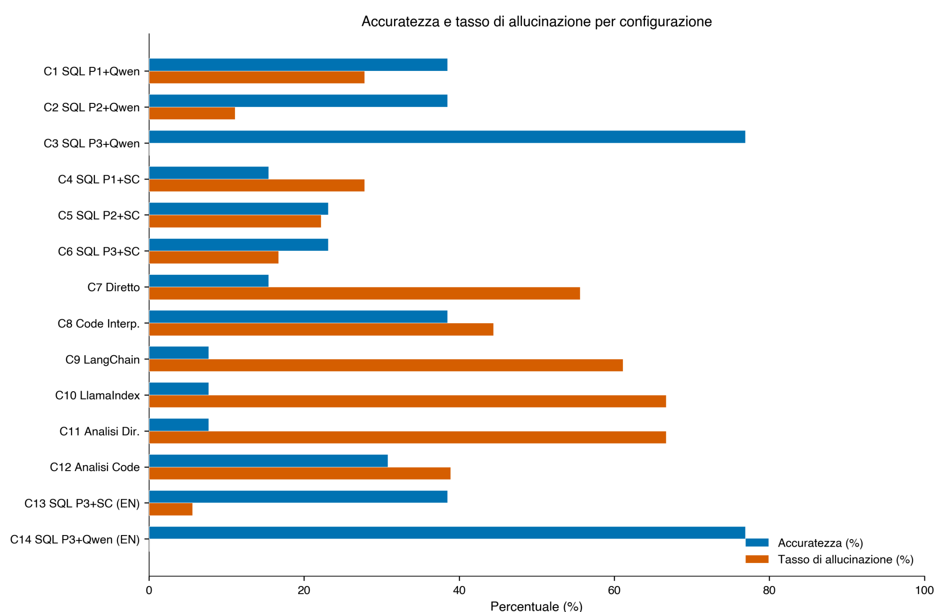


Figura 4.1: Accuratezza e tasso di allucinazione per le 14 configurazioni testate.

I framework agentici si collocano in fondo alla classifica, con un'accuratezza del 7,7% per entrambi. LangChain, nonostante il suo ciclo ReACT con ispezione incrementale della struttura, consuma in media 5.273 token di prompt per interrogazione - circa dieci volte più del corrispondente approccio

SQL *bare-metal* - e impiega in media 33 secondi per interrogazione, senza che questo overhead si traduca in una migliore accuratezza. LlamaIndex è ancora più lento (46,4 secondi di media), principalmente a causa della fase di sintesi in linguaggio naturale che segue l'esecuzione dell'interrogazione. La latenza di LlamaIndex rende il suo punteggio di latenza normalizzato prossimo allo zero, il che penalizza lo punteggio composito complessivo.

L'approccio diretto (*context stuffing*) presenta un profilo particolare: il tasso di successo esecutivo è del 100% - poiché il modello non genera codice eseguibile - ma il tasso di allucinazione è il secondo più alto (55,6%), e l'accuratezza è solo del 15,4%. Ciò significa che nella maggioranza dei casi il modello produce risposte che appaiono plausibili ma non corrispondono ai dati forniti nel contesto.

La Figura 4.2 presenta un confronto multi-dimensionale dei sei approcci con lo punteggio composito più elevato, normalizzato su quattro assi: accuratezza, latenza (invertita), token consumati (invertiti) e punteggio composito. La configurazione SQL P3 + Qwen domina su tutti gli assi tranne la latenza, dove le configurazioni con istruzioni meno ricche risultano leggermente più veloci per il minor numero di token nell'istruzione.

La Figura 4.3 mostra la distribuzione degli esiti (PASS, PARTIAL, FAIL) per categoria di domanda e configurazione, evidenziando i pattern specifici discussi di seguito.

L'analisi per categoria di domanda rivela pattern specifici. Sulle interrogazioni di tipo lookup (Q01–Q03), tutti gli approcci SQL con Qwen ottengono il 100% di accuratezza, a indicare che le domande semplici su una singola tabella non richiedono informazioni aggiuntive nello schema. Sulle interrogazioni di aggregazione (Q07–Q09), il prompt P3 si distingue per la capacità di selezionare le colonne corrette, mentre P1 e P2 tendono a confondere i nomi delle misure. Le interrogazioni più difficili, come Q11 (confronto temporale cross-tabella) e Q12 (correlazione mensile fra tabelle), producono fallimenti su tutte le configurazioni, per timeout nel primo caso e per un errore sui nomi delle colonne nel secondo. Il code interpreter riesce a risolvere domande

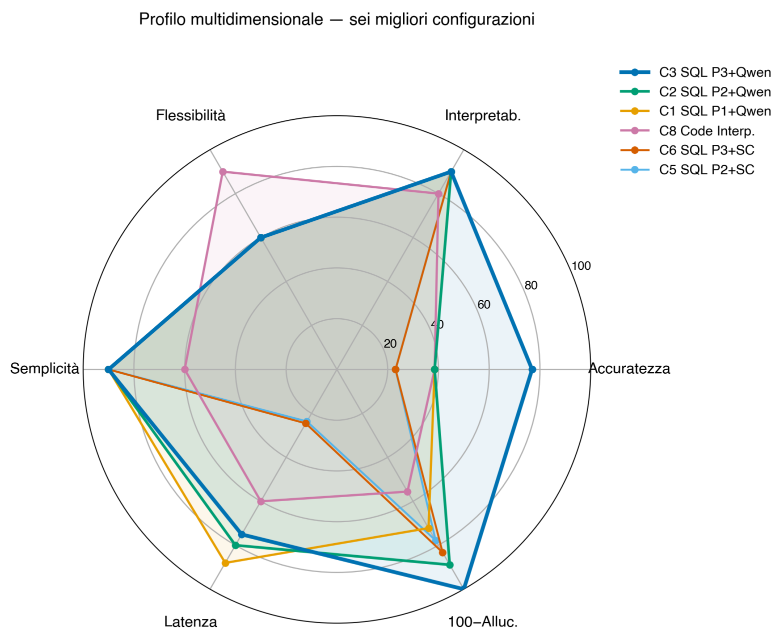
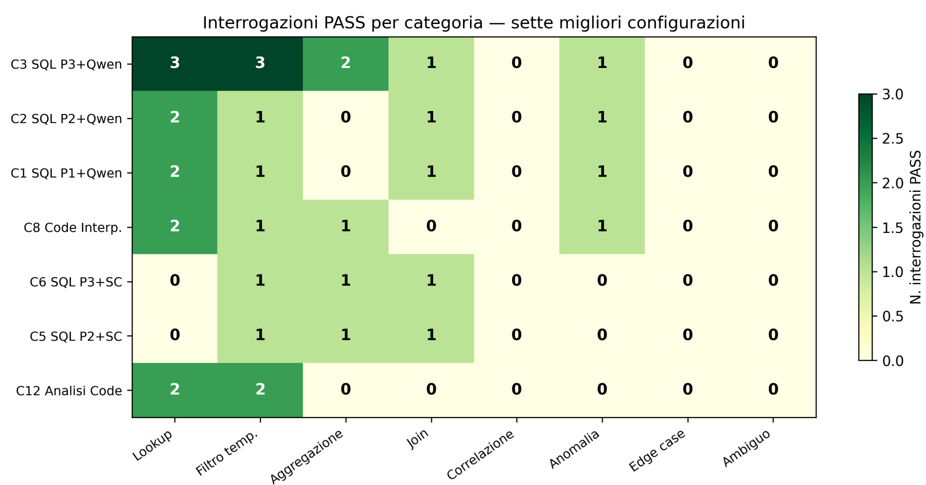


Figura 4.2: Confronto multi-dimensionale dei 6 approcci con punteggio composito più elevato.



	Lookup	Filtro temp.	Aggregazione	Join	Correlazione	Anomalia	Edge case	Ambiguo
C3 SQL P3+Qwen	3	3	2	1	0	1	0	0
C2 SQL P2+Qwen	2	1	0	1	0	1	0	0
C1 SQL P1+Qwen	2	1	0	1	0	1	0	0
C8 Code Interp.	2	1	1	0	0	1	0	0
C6 SQL P3+SC	0	1	1	1	0	0	0	0
C5 SQL P2+SC	0	1	1	1	0	0	0	0
C12 Analisi Code	2	2	0	0	0	0	0	0

Figura 4.3: Heatmap degli esiti per categoria di query e configurazione.

di tipo join (Q10) che gli approcci SQL-based con P1 e P2 non gestiscono, generando codice Python che esegue il join programmaticamente.

4.3 Impatto del prompt engineering

Il secondo esperimento (E2) si concentra sulle sei configurazioni SQL (3 prompt × 2 modelli), isolando l'effetto dell'istruzione e del modello sull'accuratezza. La Tabella 4.3 riporta i risultati completi.

Prompt	Modello	Acc. (%)	Hall. (%)	Lat. (s)	Score
P1	Qwen2.5-Coder	38,5	27,8	3,6	66,8
P2	Qwen2.5-Coder	38,5	11,1	6,0	69,8
P3	Qwen2.5-Coder	76,9	0,0	7,5	83,3
P1	SQLCoder	15,4	27,8	22,5	50,4
P2	SQLCoder	23,1	22,2	22,9	53,9
P3	SQLCoder	23,1	16,7	22,6	55,4

Tabella 4.3: Risultati delle sei configurazioni SQL-based (3 prompt × 2 modelli).

L'effetto dell'istruzione è particolarmente evidente su Qwen2.5-Coder: il passaggio da P1 o P2 a P3 raddoppia l'accuratezza (dal 38,5% al 76,9%) e azzera il tasso di allucinazione (dal 27,8% e 11,1% rispettivamente allo 0,0%). Il fatto che P1 e P2 producano la stessa accuratezza del 38,5% ma con tassi di allucinazione diversi (27,8% contro 11,1%) suggerisce che l'aggiunta delle righe di esempio nel P2 non migliora la correttezza delle interrogazioni ma riduce la tendenza del modello a produrre risposte inventate. Il prompt P3, con il dizionario dei dati e le avvertenze specifiche, rappresenta un salto qualitativo: il modello dispone delle informazioni necessarie per evitare gli errori più comuni e genera interrogazioni più precise.

Su SQLCoder, l'effetto del testo di input è meno marcato. L'accuratezza passa dal 15,4% con P1 al 23,1% con P2 e P3, con un miglioramento conte-

nuto. La latenza di SQLCoder è significativamente più alta rispetto a Qwen (oltre 22 secondi in media contro 3,6–7,5 secondi). La differenza di prestazioni fra i due modelli è attribuibile principalmente al fatto che SQLCoder, pur essendo specializzato per la generazione SQL, è stato addestrato su coppie generiche domanda-query e non dispone degli strumenti per sfruttare le informazioni contestuali del prompt P3, che il modello general-purpose riesce invece a utilizzare in modo efficace.

La Figura 4.4 visualizza i risultati delle sei configurazioni come heatmap, con colori proporzionali al punteggio composito. La configurazione C3 (P3 + Qwen, punteggio 83,3) si distingue nettamente dalle altre cinque, confermando che il fattore di maggior impatto è la combinazione di un testo di input ricco con un modello general-purpose capace di sfruttarne le informazioni.

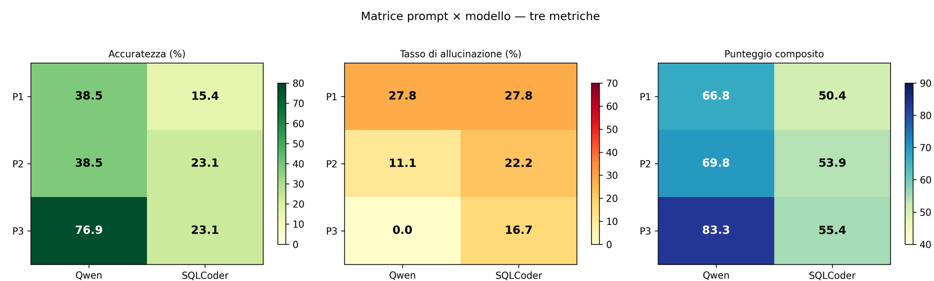


Figura 4.4: Heatmap della matrice prompt × modello: punteggio composito per le 6 configurazioni.

L'analisi dell'efficienza delle query tramite `EXPLAIN QUERY PLAN` fornisce un'ulteriore prospettiva. La Tabella 4.4 confronta i piani di esecuzione delle query generate con quelli delle query ground truth per le tre varianti di istruzione con Qwen.

Come si può osservare, il prompt P3 produce interrogazioni che raggiungono il 100% di utilizzo degli indici e una percentuale di scansioni complete (7,1%) identica a quella delle query ground truth. I prompt P1 e P2 generano query meno efficienti, con una percentuale di scansioni complete pari al 40% e al 33,3% rispettivamente, a fronte del 13,3% e 16,7% delle query di riferimento. In altri termini, il prompt P3 non solo migliora l'accuratezza ma

	Query generata			Ground truth		
Prompt	Indici (%)	Full scan (%)	Step	Indici (%)	Full scan (%)	Step
P1	93,3	40,0	2,4	100	13,3	2,4
P2	91,7	33,3	2,1	100	16,7	2,3
P3	100	7,1	1,8	100	7,1	1,6

Tabella 4.4: Efficienza delle query SQL generate vs ground truth (approccio Qwen).*

*I valori della ground truth variano fra le righe perché la metrica EXPLAIN è calcolata solo sulle interrogazioni eseguite correttamente, il cui insieme cambia a seconda della configurazione. Con P1 sono 15 su 18, con P2 sono 12 e con P3 sono 14.

produce anche interrogazioni più efficienti dal punto di vista del database, un risultato che suggerisce come la conoscenza esplicita dello schema e dei dati aiuti il modello a formulare condizioni di filtro più precise.

4.4 Risorse, report e lingua

Questa sezione raggruppa i risultati dei tre esperimenti rimanenti, ciascuno focalizzato su un aspetto specifico della valutazione.

L'esperimento E3, relativo al consumo di risorse, è stato integrato nella raccolta dati di tutti gli altri esperimenti attraverso il modulo di misurazione delle risorse. La Figura 4.5 mostra il consumo di token suddiviso per token di prompt e di completamento per ciascuna configurazione, mentre la Figura 4.6 riporta l'occupazione di memoria.

Il dato più rilevante riguarda il consumo di token: LangChain SQL Agent utilizza in media 5.273 token di prompt e 447 token di completamento per query, per un totale di oltre 102.000 token sull'intero test set. A confronto, la pipeline SQL P1 + Qwen consuma in media 487 token di prompt, circa un decimo. Questa differenza è dovuta al ciclo iterativo dell'agente, in cui ogni messaggio scambiato con il modello viene accumulato nella conver-

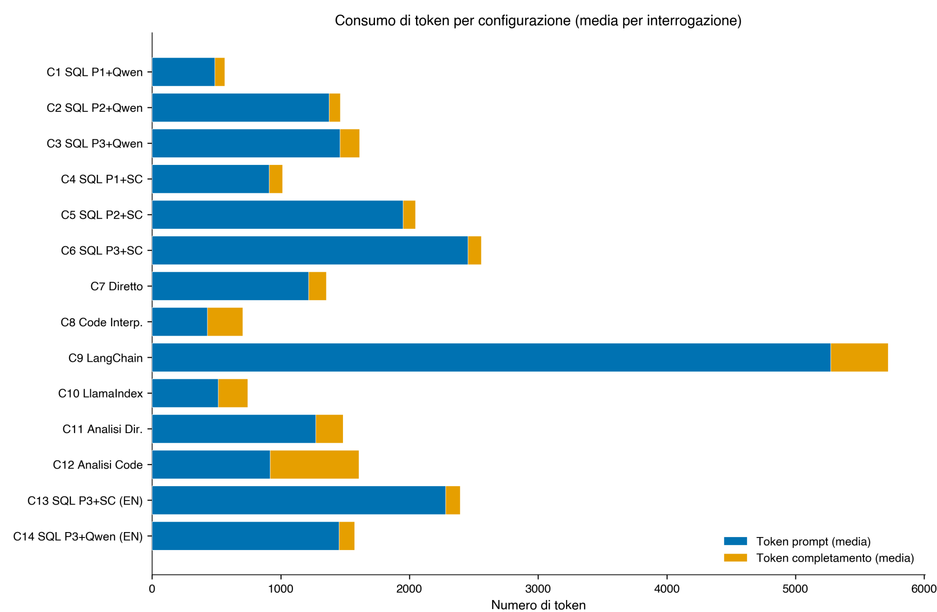


Figura 4.5: Consumo di token per configurazione, suddiviso in token di prompt e di completamento.

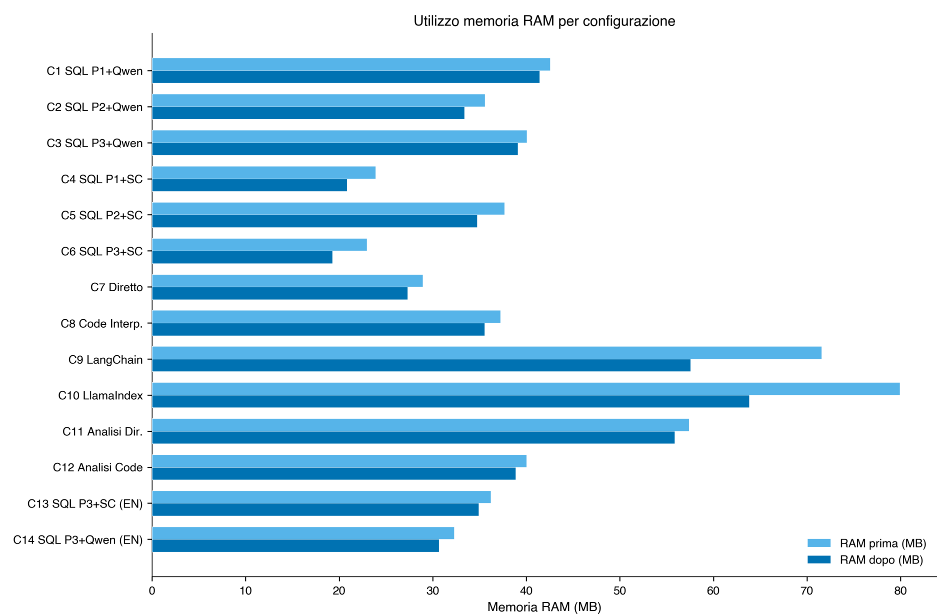


Figura 4.6: Occupazione di memoria RAM prima e dopo l'esecuzione per ciascuna configurazione.

sazione. L'approccio analysis code interpreter genera il maggior numero di token di completamento (690 in media), coerentemente con la lunghezza del codice Python che il modello produce per le analisi statistiche. In termini di memoria, l'impatto delle pipeline *bare-metal* è contenuto (35–42 MB di RAM), mentre LangChain e LlamaIndex raggiungono rispettivamente 72 MB e 80 MB prima della prima query, presumibilmente per il caricamento dei framework e dei loro componenti.

L'esperimento E4 ha valutato la generazione di un report mensile IoT per il mese di giugno 2025, con un agente LangGraph dotato di tre strumenti. L'agente non è riuscito a portare a termine il compito con il modello a sette miliardi di parametri: il ciclo ReACT si è interrotto dopo numerose iterazioni senza produrre un report completo, a causa di errori nel formato delle chiamate agli strumenti e di risposte incomplete. L'esperimento conferma che i modelli con sette miliardi di parametri non sono in grado di gestire compiti multi-step complessi con il pattern ReACT.

L'esperimento E5 ha confrontato l'effetto della lingua sulle prestazioni di entrambi i modelli, eseguendo le 18 interrogazioni tradotte in inglese con il prompt P3. La Tabella 4.5 riporta i risultati.

	SQLCoder (C6/C13)		Qwen (C3/C14)	
Metrica	IT	EN	IT	EN
Accuratezza (%)	23,1	38,5	76,9	76,9
Tasso di allucinazione (%)	16,7	5,6	0,0	0,0
Latenza media (s)	22,6	7,4	7,5	7,6
Punteggio composito	55,4	70,4	83,3	83,3

Tabella 4.5: Confronto fra italiano e inglese per entrambi i modelli con prompt P3.

L'effetto della lingua è asimmetrico. Per SQLCoder, il passaggio all'inglese produce un miglioramento su tutte le metriche: l'accuratezza aumenta

di 15,4 punti percentuali, il tasso di allucinazione scende dall'16,7% al 5,6% e la latenza si riduce di un fattore tre. Questo risultato è coerente con il fatto che SQLCoder è stato addestrato prevalentemente su dati in lingua inglese. Per Qwen, la lingua non ha alcun effetto misurabile: accuratezza, tasso di allucinazione e punteggio composito sono identici nelle due versioni. Il modello general-purpose, addestrato su un corpus multilingue, gestisce entrambe le lingue con la stessa efficacia.

4.5 Discussione

I risultati presentati nelle sezioni precedenti consentono di verificare le otto ipotesi formulate nella Sezione 2.2. La Tabella 4.6 riassume gli esiti con l'evidenza numerica di supporto.

ID	Ipotesi	Esito	Evidenza
H1	Prompt > modello	Parziale	$\bar{\Delta}_{\text{prompt}} = 23{,}1$ pp $< \bar{\Delta}_{\text{modello}} = 30{,}8$ pp
H2	Specializzato > general-purpose	Smentita	Qwen > SQLCoder su P1, P2, P3
H3	Code interpreter competitivo	Parziale	C8: 2/7 complesse, come C1/C2
H4	Diretto ha più allucinazioni	Parziale	C7: 55,6%, 4° posto
H5	Diretto eccelle su semantiche	Non verif.	Score SEMANTIC per tutti
H6	Framework migliorano recovery	Smentita	C9/C10: 7,7%, 10,1× token
H7	Inglese migliora risultati	Parziale	SQLCoder: +15,4 pp; Qwen: +0,0 pp
H8	Token variano fra approcci	Confermata	Rapporto 10,1×

Tabella 4.6: Verifica delle otto ipotesi sperimentali.

H1 – Parziale. L'impatto del prompt engineering e del cambio di modello non sono separabili a causa di una forte interazione. L'incremento medio di accuratezza ottenuto passando da P1 a P3, a modello fisso, è:

$$\bar{\Delta}_{\text{prompt}} = \frac{(76{,}9 - 38{,}5) + (23{,}1 - 15{,}4)}{2} = 23{,}05 \text{ pp} \tag{4.1}$$

L'incremento medio ottenuto passando da SQLCoder a Qwen, a istruzione fissa, è:

$$\bar{\Delta}_{\text{modello}} = \frac{(38{,}5 - 15{,}4) + (38{,}5 - 23{,}1) + (76{,}9 - 23{,}1)}{3} = 30{,}77 \text{ pp} \quad (4.2)$$

Con la media semplice, il cambio di modello ha un impatto maggiore (30,8 pp > 23,1 pp). Tuttavia, l'effetto è dominato dall'interazione: il prompt P3 produce un incremento di 38,4 pp su Qwen ma solo 7,7 pp su SQLCoder. Il modello general-purpose è in grado di sfruttare le informazioni aggiuntive del dizionario dati, mentre il modello specializzato, vincolato dal formato di completamento di StarCoder, non ne trae beneficio. L'effetto non è attribuibile a un singolo fattore ma alla combinazione di entrambi.

H2 – Smentita. Qwen supera SQLCoder su tutti i livelli di istruzione: +23,1 pp con P1, +15,4 pp con P2, +53,8 pp con P3. La versione di SQLCoder disponibile (v1, basata su StarCoder) non supporta il beam search indicato come necessario nel paper originale [8] e il suo addestramento su coppie generiche domanda-query non gli fornisce gli strumenti per sfruttare le informazioni contestuali del prompt P3.

H3 – Parziale. Sulle sette interrogazioni complesse (Q07–Q13), il code interpreter (C8) ottiene 2/7 PASS, come le configurazioni SQL con P1 e P2 (C1, C2). La configurazione SQL con P3 (C3) raggiunge 4/7 PASS. Il code interpreter è competitivo con la generazione SQL senza dizionario dati, ma non con la configurazione migliore.

H4 – Parziale. L'approccio diretto (C7) presenta un tasso di allucinazione del 55,6%, elevato ma non il più alto: LlamaIndex (C10) e l'analisi diretta (C11) raggiungono il 66,7%, LangChain (C9) il 61,1%. L'approccio diretto si colloca al quarto posto. Il profilo delle allucinazioni è diverso: nel diretto il modello produce risposte plausibili ma non supportate dai dati nel contesto; nei framework agentici il modello genera interrogazioni SQL errate che

restituiscono risultati apparentemente validi ma riferiti a tabelle o colonne sbagliate.

H5 – Non verificabile. Tutte le interrogazioni semantiche (Q14–Q18) ricevono punteggio SEMANTIC per ogni configurazione, poiché il sistema di valutazione automatica non è in grado di giudicare la qualità delle risposte interpretative. Si ipotizza che la limitazione sia dovuta alla dimensione del modello (sette miliardi di parametri), che non fornisce capacità di ragionamento critico sufficienti per analisi semantiche dei dati. Senza un confronto con modelli di dimensioni superiori, non è possibile confermare né smentire questa ipotesi.

H6 – Smentita (con modelli a sette miliardi di parametri). LangChain (C9) e LlamaIndex (C10) ottengono entrambi il 7,7% di accuratezza, il peggior risultato fra tutte le configurazioni. LangChain consuma 102.962 token sull'intero test set, 10,1 volte più della pipeline SQL più leggera (C1: 10.160 token), senza benefici in accuratezza. I modelli a sette miliardi di parametri non riescono a gestire il ciclo ReACT multi-step dei framework agentici.

H7 – Parziale. L'effetto della lingua è asimmetrico. Per SQLCoder, l'inglese migliora l'accuratezza di 15,4 pp (da 23,1% a 38,5%) e riduce il tasso di allucinazione di 11,1 pp. Per Qwen, l'inglese non ha alcun effetto misurabile: accuratezza (76,9%) e tasso di allucinazione (0,0%) sono identici nelle due lingue. Il modello specializzato, addestrato prevalentemente su dati inglesi, beneficia della lingua nativa; il modello general-purpose, addestrato su un corpus multilingue, gestisce entrambe le lingue con la stessa efficacia.

H8 – Confermata. Il consumo di token varia di un fattore 10,1 fra la configurazione più efficiente (C1: 10.160 token) e quella meno efficiente (C9: 102.962 token). Il framework agentico LangChain, con il suo ciclo iterativo,

accumula contesto a ogni turno, generando un consumo di token proporzionale al numero di interazioni con il modello (3–10 per interrogazione).

Alcune limitazioni del lavoro meritano di essere riconosciute. Il test set di 18 interrogazioni, pur coprendo otto categorie diverse, è contenuto rispetto ai benchmark della letteratura come Spider [2] con oltre 10.000 domande. L'utilizzo di modelli a sette miliardi di parametri, imposto dal vincolo hardware, non consente di trarre conclusioni sui modelli di dimensioni superiori, per i quali i framework agentici potrebbero risultare più efficaci. L'esecuzione singola per configurazione, giustificata dalla temperatura nulla, non cattura la variabilità che potrebbe emergere con temperature di generazione diverse da zero. Una valutazione manuale sistematica delle interrogazioni semantiche su tutte le configurazioni rimane un possibile sviluppo futuro.

4.6 Dashboard interattiva

Accanto al sistema di valutazione, è stata sviluppata una dashboard interattiva per la visualizzazione e l'esplorazione dei risultati. I dati del benchmark, prodotti dalla pipeline Python descritta nelle sezioni precedenti, vengono aggregati in un unico file JSON dallo script di esportazione. La dashboard importa il file a tempo di compilazione, rendendo le pagine di consultazione completamente statiche e utilizzabili senza un server dedicato. La Figura 4.7 illustra il flusso dati.

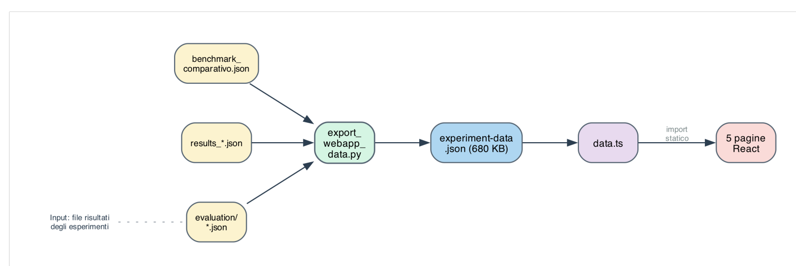


Figura 4.7: Flusso dati dalla pipeline di benchmark alla dashboard interattiva.

La dashboard comprende cinque pagine. La pagina principale offre una panoramica dei risultati con un grafico a barre raggruppate (accuratezza, tasso di allucinazione, tasso di successo per tutte le configurazioni), un grafico radar dei sei approcci migliori e la tabella di ranking ordinabile. La Figura 4.8 ne mostra una vista.

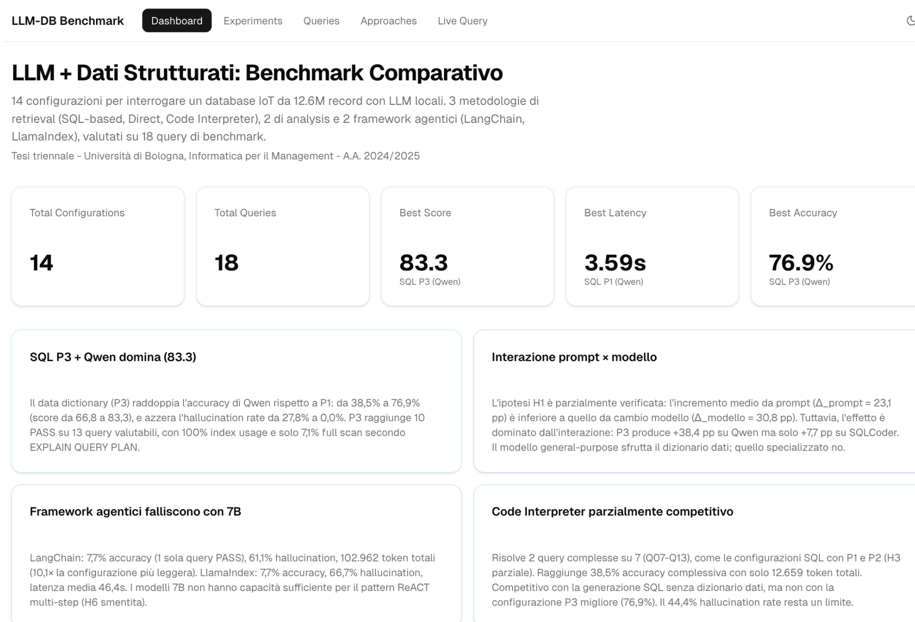


Figura 4.8: Pagina Dashboard: panoramica dei risultati con grafico a barre e tabella di ranking.

La pagina Experiments organizza i risultati dei cinque esperimenti in altrettanti tab, ciascuno con grafici e tabelle specifiche: ranking e distribuzione esiti per E1, matrice prompt × modello per E2, barre impilate per i token in E3, anteprima del report per E4, confronto linguistico per E5. La Figura 4.9 mostra il tab E2.

La pagina Queries consente di esplorare i risultati per singola domanda tramite una heatmap interattiva, con accesso ai dettagli di ogni risposta (SQL o codice generato, contesto dati, tempistiche, token). La pagina Approaches permette di analizzare una singola configurazione su tutte le 18 interrogazioni, con metriche dettagliate e, per gli approcci SQL, i risultati

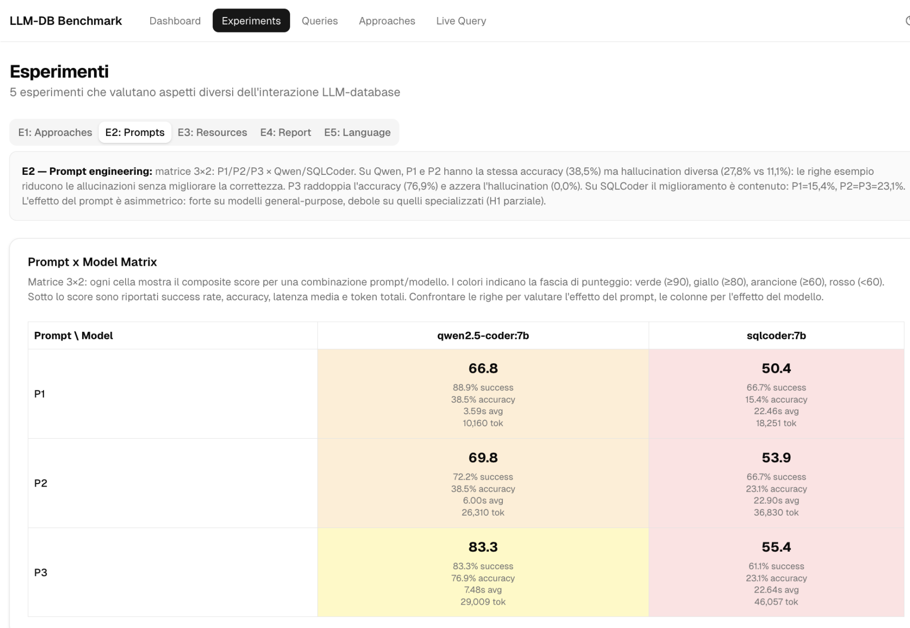


Figura 4.9: Pagina Experiments: tab E2 con la matrice prompt × modello e grafico comparativo.

dell'EXPLAIN QUERY PLAN. La pagina Live Query consente di inviare interrogazioni in tempo reale a un server API locale, selezionando approccio, modello e prompt. Il codice sorgente della dashboard è disponibile nella repository del progetto.

Conclusioni

Il presente lavoro ha affrontato il problema dell'interrogazione di base di dati strutturate tramite modelli linguistici di grandi dimensioni, confrontando quattro metodologie su un caso d'uso reale di monitoraggio IoT. Il confronto, condotto su un dataset di 12,6 milioni di record provenienti da 28 dispositivi distribuiti in Emilia-Romagna, ha coinvolto 14 configurazioni valutate su 18 interrogazioni con risposta attesa verificata, producendo un benchmark multi-dimensionale con un punteggio composito a sei metriche.

Il risultato principale è che il prompt engineering, nella forma di un dizionario dei dati che fornisce al modello la conoscenza esplicita del dominio, si è rivelato il fattore con il maggiore impatto sulle prestazioni. La configurazione SQL-based con prompt arricchito (P3) e modello general-purpose a sette miliardi di parametri ha raggiunto il 76,9% di accuratezza con un tasso di allucinazione nullo, raddoppiando le prestazioni delle configurazioni con istruzioni meno dettagliate. L'analisi ha rivelato una forte interazione fra testo di input e modello: l'incremento medio di accuratezza dovuto al prompt ($\bar{\Delta}_{\text{prompt}} = 23{,}1$ pp) è inferiore a quello dovuto al cambio di modello ($\bar{\Delta}_{\text{modello}} = 30{,}8$ pp), ma l'effetto è dominato dalla combinazione P3 con il modello general-purpose (+38,4 pp), mentre il modello specializzato beneficia in misura molto minore del dizionario dati (+7,7 pp). I framework agentici, progettati per orchestrare il modello in cicli di ragionamento multi-step, si sono rivelati inadatti con sette miliardi di parametri, ottenendo le accuratezze più basse del benchmark (7,7%) con un consumo di risorse fino a dieci volte superiore. L'effetto della lingua è asimmetrico: l'inglese migliora

le prestazioni del modello specializzato (+15,4 pp) ma non ha alcun effetto su quello general-purpose.

Il lavoro presenta alcune limitazioni che ne circoscrivono la portata. Il test set di 18 interrogazioni, sebbene copra otto categorie e tre livelli di difficoltà, è contenuto rispetto ai benchmark della letteratura. L'utilizzo esclusivo di modelli a sette miliardi di parametri non consente di estendere le conclusioni a modelli di dimensioni maggiori, per i quali il bilancio fra framework agentici e pipeline lineari potrebbe essere diverso. La valutazione delle cinque interrogazioni semantiche è rimasta automatizzata, senza una revisione manuale sistematica per tutti gli approcci. Le metriche qualitative del punteggio composito, pur essendo state definite prima degli esperimenti, contengono un elemento di soggettività nella loro assegnazione.

Diverse direzioni di sviluppo futuro potrebbero estendere il lavoro. Una prima possibilità sarebbe la valutazione con modelli di dimensioni superiori, nella fascia 14–32 miliardi di parametri, per verificare se i framework agentici diventano competitivi quando il modello sottostante è più capace. L'adozione di tecniche di RAG (*Retrieval-Augmented Generation*) per la selezione dinamica della struttura potrebbe migliorare le prestazioni su database con un numero elevato di tabelle. L'ampliamento del test set, con l'aggiunta di sequenze di interrogazioni successive e di domande che richiedono ragionamento temporale complesso, consentirebbe una valutazione più approfondita. La sperimentazione con versioni più recenti dei modelli specializzati, come SQLCoder v2 basato su CodeLlama, potrebbe modificare il bilancio fra modello general-purpose e specializzato. Il *fine-tuning* di un modello di dimensioni contenute sul dataset specifico rappresenta un'altra direzione percorribile per migliorare l'accuratezza senza aumentare i requisiti hardware. Sarebbe infine opportuno condurre una valutazione manuale completa delle interrogazioni semantiche su tutte le configurazioni, per integrare il benchmark automatico con una dimensione qualitativa che l'attuale sistema non è in grado di catturare.

Appendice A

Prompt utilizzati

In questa appendice sono riportati i testi completi dei prompt utilizzati negli esperimenti. Per ciascun prompt sono indicati i campi variabili (fra parentesi graffe) che vengono sostituiti a tempo di esecuzione con il contenuto effettivo.

A.1 Prompt per la generazione SQL – modello general-purpose

A.1.1 P1 – Solo schema

```
You are a SQL expert working with a SQLite database.
Given the database schema below, write a valid SQLite query
to answer the user's question.
Return ONLY the SQL query, nothing else.

DATABASE SCHEMA:
{schema}

RULES:
- Use ONLY tables and columns that exist in the schema
- Use SQLite syntax (e.g. strftime for date functions)
- Return ONLY the SQL query, no explanations, no markdown
```

```
- Do NOT use INSERT, UPDATE, DELETE, DROP, or any write
  operation

USER QUESTION: {question}

SQL QUERY:
```

Listato A.1: Prompt P1: solo schema (modello general-purpose).

A.1.2 P2 – Schema e dati campione

```
You are a SQL expert working with a SQLite database.
Given the database schema and sample data below, write a
  valid
SQLite query to answer the user's question.
Return ONLY the SQL query, nothing else.

DATABASE SCHEMA:
{schema}

SAMPLE DATA:
{sample_section}

RULES:
- Use ONLY tables and columns that exist in the schema
- Use SQLite syntax (e.g. strftime for date functions)
- Return ONLY the SQL query, no explanations, no markdown
- Do NOT use INSERT, UPDATE, DELETE, DROP, or any write
  operation

USER QUESTION: {question}

SQL QUERY:
```

Listato A.2: Prompt P2: schema e dati campione (modello general-purpose).

A.1.3 P3 – Schema e dizionario dati

```
You are a SQL expert working with a SQLite database.
Given the schema and detailed data dictionary below, write a
valid SQLite query to answer the user's question.
Return ONLY the SQL query, nothing else.

DATABASE SCHEMA:
{schema}

DATA DICTIONARY:

TABLE: devices
  - Contains 28 local IoT sensors deployed in
    Emilia-Romagna, Italy.
  - name (TEXT, PRIMARY KEY): unique identifier,
    format "WS-N" (weather station)
    or "EM-500-N" (ground sensor).
  - type (TEXT): "weather_station" or "ground_sensor".

TABLE: measurements
  - Contains sensor readings from the 28 local IoT devices
    (12.6M rows, Feb 2025 - Jan 2026).
  - id (INTEGER, PRIMARY KEY): auto-increment row ID.
  - timestamp (TEXT): format "YYYY-MM-DD HH:MM:SS.ns+00:00"
    (UTC timezone).
  - device_name (TEXT): FK to devices.name.
  - measurement_name (TEXT): type of measurement
    (see list below).
  - value (REAL): the measured value.
  - NOTE: This table contains LOCAL sensor data,
    NOT weather API data.
  - NOTE: measurements.temperature is the temperature
    from local sensors (soil for EM, air for WS).

TABLE: weather_hourly
  - Contains hourly weather data from Open-Meteo external API
    (8,400 rows).
  - id (INTEGER, PRIMARY KEY): auto-increment row ID.
  - timestamp (TEXT): format "YYYY-MM-DD HH:MM:SS+01:00"
```

```
    (Europe/Rome timezone).
  - temperature_2m (REAL): air temperature at 2m height
    in Celsius.
  - relative_humidity_2m (REAL): relative humidity
     percentage.
  - weather_code (INTEGER): WMO weather code.
  - rain (REAL): precipitation in mm.
  - wind_speed_10m (REAL): wind speed at 10m in km/h.
  - wind_direction_10m (REAL): wind direction in degrees.
  - NOTE: This is EXTERNAL API data,
    NOT linked to devices table.
  - NOTE: weather_hourly.temperature_2m is air temperature
    from external API.

MEASUREMENT NAMES (13 possible values):
  battery, ec, f_cnt, humidity, moisture, pressure,
  rainfall_counter, rainfall_mm, rssi, snr, temperature,
  wind_direction, wind_speed

DEVICE TYPE -> MEASUREMENT MAPPING:
  Weather stations only (WS-*): humidity, pressure,
    rainfall_counter, rainfall_mm, wind_direction, wind_speed
  Ground sensors only (EM-500-*): ec, moisture
  Both types: battery, f_cnt, rssi, snr, temperature

TIMESTAMP WARNING:
  - measurements.timestamp is in UTC (+00:00)
  - weather_hourly.timestamp is in Europe/Rome
    (+01:00 or +02:00 DST)
  - For JOIN between tables: convert timezone

DATA WARNINGS:
  - EM-500-5 stopped transmitting July 2025
  - WS-4 stopped transmitting July 2025
  - EM-500-19 stopped transmitting December 2025
  - rainfall_mm values are unreliable for SUM aggregations
  - February 2025 and January 2026 are partial months
```

```
RULES:
- Use ONLY tables and columns that exist in the schema
- Use SQLite syntax (e.g. strftime for date functions)
- Return ONLY the SQL query, no explanations, no markdown
- Do NOT use INSERT, UPDATE, DELETE, DROP, or any write
  operation

USER QUESTION: {question}

SQL QUERY:
```

Listato A.3: Prompt P3: schema e dizionario dati (modello general-purpose).

A.2 Prompt per la generazione SQL – modello specializzato

Il modello specializzato (SQLCoder) utilizza un formato a completamento anziché un formato di chat. Le tre strategie P1, P2 e P3 hanno lo stesso contenuto informativo delle corrispondenti versioni per il modello general-purpose, ma sono strutturate secondo il formato richiesto da SQLCoder v1 (basato su StarCoder).

```
### Instructions:
Your task is to convert a question into a SQL query,
given a SQLite database schema.
Adhere to these rules:
- **Deliberately go through the question and database
  schema word by word** to appropriately answer the question
- **Use Table Aliases** to prevent ambiguity.
- Use SQLite syntax (e.g. strftime for date functions)
- Return ONLY the SQL query
- Do NOT use INSERT, UPDATE, DELETE, DROP, or any write
  operation

### Input:
Generate a SQL query that answers the question `{question}`.
```

```
This query will run on a database whose schema is represented
in this string:
{schema}

{data_dictionary_as_sql_comments}

### Response:
Based on your instructions, here is the SQL query I have
generated to answer the question `{question}`:
```sql
```

Listato A.4: Struttura del prompt per SQLCoder (formato a completamento). Il corpo del data dictionary è identico a quello del prompt P3 general-purpose.

A.3 Prompt per l'approccio diretto (*context stuffing*)

```
You are a data analyst. The database schema and relevant data
are provided below.
Answer the user's question based ONLY on the provided data.
Do NOT generate SQL.

DATABASE SCHEMA (for reference):
{schema}

RELEVANT DATA:
{data_context}

RULES:
- Answer based ONLY on the data provided above
- If the data is insufficient, say so
- Return your answer as JSON:
  {"answer": <value>, "explanation": "<brief reasoning>"}

USER QUESTION: {question}
```
```

```
JSON ANSWER:
```

Listato A.5: Prompt dell'approccio diretto: i dati estratti vengono inseriti nel contesto.

A.4 Prompt per il *code interpreter*

```
You are a Python data analyst. Write a complete Python script
to answer the user's question by querying the SQLite
   database.

DATABASE SCHEMA:
{schema}

AVAILABLE LIBRARIES: sqlite3, json, datetime, math,
  statistics, pandas, numpy, matplotlib

DATABASE CONNECTION (use exactly this):
  import sqlite3
  conn = sqlite3.connect("file:{db_path}?mode=ro", uri=True)

OUTPUT FORMAT:
  The script MUST print the final result as JSON using:
  import json
  print(json.dumps({"result": <your_answer>}))

RULES:
- Write a COMPLETE, self-contained Python script
- Use read-only database connection
- Print ONLY the JSON result, no other output
- Handle errors gracefully
- Do NOT use INSERT, UPDATE, DELETE, DROP,
  or any write operation

USER QUESTION: {question}
```

```
PYTHON SCRIPT:
```

Listato A.6: Prompt del code interpreter: il modello genera uno script Python completo.

In caso di errore di esecuzione, il code interpreter utilizza un prompt di retry che include il codice precedente e il traceback dell'errore:

```
You are a Python data analyst. Your previous code produced
an error. Fix the code and write a COMPLETE corrected
Python script.

DATABASE SCHEMA:
{schema}

YOUR PREVIOUS CODE:
{previous_code}

ERROR:
{error}

USER QUESTION: {question}

CORRECTED PYTHON SCRIPT:
```

Listato A.7: Prompt di retry del code interpreter.

Appendice B

Interrogazioni di test

In questa appendice sono riportate le 18 interrogazioni del test set con la relativa ground truth SQL. Per ciascuna interrogazione sono indicati: la domanda in italiano e in inglese, la categoria, la difficoltà, il tipo di valutazione e la query SQL di riferimento. Le interrogazioni Q14–Q18 sono valutate con criterio semantico e non contribuiscono al calcolo automatico dell'accuratezza.

B.1 Interrogazioni con valutazione automatica (Q01–Q13)

Q01 — Lookup, facile, exact match
IT: Quanti dispositivi ci sono nel database?
EN: How many devices are in the database?

```
SELECT COUNT(*) FROM devices;
```

Q02 — Lookup, facile, set match
IT: Quali tipi di misura registra il sensore WS-1?
EN: What measurement types does sensor WS-1 record?

```
SELECT DISTINCT measurement_name
FROM measurements
WHERE device_name = 'WS-1'
```

```
ORDER BY measurement_name;
```

Q03 — Lookup, facile, exact match

IT: Qual è l’ultima lettura di temperatura del sensore EM-500-3?

EN: What is the latest temperature reading from sensor EM-500-3?

```
SELECT timestamp, value
FROM measurements
WHERE device_name = 'EM-500-3'
  AND measurement_name = 'temperature'
ORDER BY timestamp DESC LIMIT 1;
```

Q04 — Filtro temporale, facile, exact match

IT: Quante letture di temperatura ha registrato WS-2 il 15 giugno 2025?

EN: How many temperature readings did WS-2 record on June 15, 2025?

```
SELECT COUNT(*)
FROM measurements
WHERE device_name = 'WS-2'
  AND measurement_name = 'temperature'
  AND timestamp >= '2025-06-15'
  AND timestamp < '2025-06-16';
```

Q05 — Filtro temporale, facile, exact match

IT: Quante misurazioni ha effettuato EM-500-10 a luglio 2025?

EN: How many measurements did EM-500-10 take in July 2025?

```
SELECT COUNT(*)
FROM measurements
WHERE device_name = 'EM-500-10'
  AND timestamp >= '2025-07-01'
  AND timestamp < '2025-08-01';
```

Q06 — Filtro temporale, facile, numeric tolerance

IT: Qual è la pressione atmosferica media registrata dalle weather station a giugno 2025?

EN: What is the average atmospheric pressure recorded by weather stations in June 2025?

```
SELECT ROUND(AVG(value), 2)
FROM measurements
```

```
WHERE measurement_name = 'pressure'
  AND timestamp >= '2025-06-01'
  AND timestamp < '2025-07-01';
```

Q07 — Aggregazione, media, numeric tolerance
IT: Qual è la temperatura media, minima e massima registrata da ogni weather station?
EN: What is the average, minimum, and maximum temperature recorded by each weather station?

```
SELECT device_name,
       ROUND(AVG(value), 2) AS avg_temp,
       ROUND(MIN(value), 2) AS min_temp,
       ROUND(MAX(value), 2) AS max_temp
FROM measurements
WHERE measurement_name = 'temperature'
  AND device_name LIKE 'WS-%'
GROUP BY device_name
ORDER BY device_name;
```

Q08 — Aggregazione, media, exact match
IT: Quale dispositivo ha registrato il maggior numero di misurazioni in totale?
EN: Which device recorded the highest total number of measurements?

```
SELECT device_name, COUNT(*) AS n_measurements
FROM measurements
GROUP BY device_name
ORDER BY n_measurements DESC LIMIT 1;
```

Q09 — Aggregazione, media, numeric tolerance
IT: Quali sono i tre ground sensor con l'umidità del suolo media più alta?
EN: What are the 3 ground sensors with the highest average soil moisture?

```
SELECT device_name,
       ROUND(AVG(value), 2) AS avg_moisture
FROM measurements
WHERE measurement_name = 'moisture'
GROUP BY device_name
ORDER BY avg_moisture DESC LIMIT 3;
```

Q10 — Join, media, exact match
IT: Quanti dispositivi per ogni tipo ci sono e quante misurazioni totali ha registrato ogni tipo?
EN: How many devices of each type are there, and how many total measurements has each type recorded?

```
SELECT d.type,
       COUNT(DISTINCT d.name) AS n_devices,
       COUNT(m.id) AS n_measurements
FROM devices d
LEFT JOIN measurements m ON d.name = m.device_name
GROUP BY d.type
ORDER BY d.type;
```

Q11 — Join, media, numeric tolerance
IT: Qual è la temperatura media del sensore WS-3 e la temperatura media dell'API meteo per agosto 2025?
EN: What is the average temperature from sensor WS-3 and the average temperature from the weather API for August 2025?

```
SELECT 'WS-3 (sensore)' AS source,
       ROUND(AVG(value), 2) AS avg_temp
FROM measurements
WHERE device_name = 'WS-3'
  AND measurement_name = 'temperature'
  AND timestamp >= '2025-08-01'
  AND timestamp < '2025-09-01'
UNION ALL
SELECT 'API meteo' AS source,
       ROUND(AVG(temperature_2m), 2) AS avg_temp
FROM weather_hourly
WHERE timestamp >= '2025-08-01'
  AND timestamp < '2025-09-01';
```

Q12 — Correlazione, difficile, numeric tolerance
IT: Calcola la pioggia totale mensile dall'API meteo e l'umidità media mensile del suolo dai ground sensor, per il periodo giugno–novembre 2025.
EN: Calculate the total monthly rainfall from the weather API and the average monthly soil moisture from ground sensors, for the period June–November 2025.

```
SELECT rain.month, rain.total_rain_mm,
       moist.avg_moisture
FROM (
  SELECT strftime('%Y-%m', timestamp) AS month,
         ROUND(SUM(rain), 2) AS total_rain_mm
  FROM weather_hourly
  WHERE timestamp >= '2025-06-01'
    AND timestamp < '2025-12-01'
  GROUP BY strftime('%Y-%m', timestamp)
) rain
JOIN (
  SELECT strftime('%Y-%m', timestamp) AS month,
         ROUND(AVG(value), 2) AS avg_moisture
  FROM measurements
  WHERE measurement_name = 'moisture'
    AND timestamp >= '2025-06-01'
    AND timestamp < '2025-12-01'
  GROUP BY strftime('%Y-%m', timestamp)
) moist ON rain.month = moist.month
ORDER BY rain.month;
```

Q13 — Rilevamento anomalie, difficile, exact match
IT: Quanti valori di pressione fuori dall'intervallo 990–1035 hPa sono stati registrati dalle weather station?
EN: How many pressure values outside the range 990–1035 hPa were recorded by weather stations?

```
SELECT COUNT(*)
FROM measurements
WHERE measurement_name = 'pressure'
  AND (value < 990 OR value > 1035);
```

B.2 Interrogazioni con valutazione semantica (Q14–Q18)

Le interrogazioni seguenti sono progettate per testare la capacità del modello di riconoscere limiti e particolarità del dataset. La valutazione è di tipo semantico: non esiste una singola risposta corretta, ma il modello ideale dovrebbe segnalare il problema specifico di ogni interrogazione.

Q14 — Edge case, media, semantica

IT: Qual è la temperatura media di EM-500-5 a settembre 2025?

EN: What is the average temperature of EM-500-5 in September 2025?

Trappola: EM-500-5 ha smesso di trasmettere a luglio 2025. Il risultato SQL è NULL.

```
SELECT ROUND(AVG(value), 2)
FROM measurements
WHERE device_name = 'EM-500-5'
  AND measurement_name = 'temperature'
  AND timestamp >= '2025-09-01'
  AND timestamp < '2025-10-01';
```

Q15 — Edge case, difficile, semantica

IT: Quanta pioggia hanno registrato le weather station a ottobre 2025 in media?

EN: How much rainfall did the weather stations record on average in October 2025?

Trappola: `rainfall_mm` è inaffidabile. Il risultato SQL (1,71 mm) è fuorviante rispetto ai 78,8 mm reali dall'API.

```
SELECT ROUND(AVG(value), 2)
FROM measurements
WHERE measurement_name = 'rainfall_mm'
  AND timestamp >= '2025-10-01'
  AND timestamp < '2025-11-01';
```

Q16 — Edge case, media, semantica

IT: Quante misurazioni di umidità del suolo ha effettuato il sensore WS-1?

EN: How many soil moisture measurements has sensor WS-1 taken?

Trappola: WS-1 è una weather station, non misura `moisture`. Il risultato è 0.

```
SELECT COUNT(*)
FROM measurements
WHERE device_name = 'WS-1'
  AND measurement_name = 'moisture';
```

Q17 — Edge case, difficile, semantica

IT: Qual è la temperatura media di tutti i sensori a febbraio 2025?

EN: What is the average temperature from all sensors in February 2025?

Trappola: febbraio 2025 è parziale (dal 14), con solo due sensori attivi (EM-500-1 e EM-500-2, temperatura del suolo). Il valore di 22 °C è fuorviante.

```
SELECT ROUND(AVG(value), 2)
FROM measurements
WHERE measurement_name = 'temperature'
  AND timestamp >= '2025-02-01'
  AND timestamp < '2025-03-01';
```

Q18 — Ambigua, difficile, semantica

IT: Come stanno i sensori?

EN: How are the sensors doing?

Trappola: domanda volutamente vaga. Non esiste una ground truth SQL univoca. Il modello deve decidere autonomamente quali aspetti dello stato dei sensori presentare.

Bibliografia

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, “Attention Is All You Need”, in *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*, Long Beach, CA, USA, pp. 6000–6010, 2017.

[2] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, D. Radev, “Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task”, in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Brussels, Belgium, pp. 3911–3921, 2018.

[3] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. Pinto de Oliveira, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, *et al.*, “Evaluating Large Language Models Trained on Code”, *arXiv preprint*, arXiv: 2107.03374, 2021.

[4] H. Chase, “LangChain”, 2022. [Online]. Available: `https://github.com/langchain-ai/langchain`.

[5] J. Liu, “LlamaIndex”, 2022. [Online]. Available: `https://github.com/run-llama/llama_index`.

[6] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, Y. Cao, “ReAct: Synergizing Reasoning and Acting in Language Mo-

dels", in *Proceedings of the 11th International Conference on Learning Representations (ICLR)*, Kigali, Rwanda, 2023.

[7] S. Chang, E. Fosler-Lussier, "How to Prompt LLMs for Text-to-SQL: A Study in Zero-shot, Single-domain, and Cross-domain Settings", *arXiv preprint*, arXiv: 2305.11853, 2023.

[8] Defog.ai, "SQLCoder: State-of-the-Art LLM for Generating SQL Queries from Natural Language", 2023. [Online]. Available: `https://defog.ai/blog/open-sourcing-sqlcoder`.

[9] J. Morgan *et al.*, "Ollama", 2023. [Online]. Available: `https://ollama.com`.

[10] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim, *et al.*, "StarCoder: May the Source Be with You!", *Transactions on Machine Learning Research (TMLR)*, 2023.

[11] M. Pourreza, D. Rafiei, "DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction", in *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, pp. 36339–36348, 2023.

[12] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Cao, R. Geng, N. Huo, X. Zhou, C. Ma, G. Li, K. C. C. Chang, F. Huang, R. Cheng, Y. Li, "Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs", in *Advances in Neural Information Processing Systems 36 (NeurIPS 2023), Datasets and Benchmarks Track*, 2024.

[13] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Lu, *et al.*, "Qwen2.5-Coder Technical Report", *arXiv preprint*, arXiv: 2409.12186, 2024.

[14] Z. Hong, Z. Yuan, Q. Zhang, H. Chen, J. Dong, F. Huang, X. Huang, "Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL", *IEEE Transactions on Knowledge and Data Engineering*, vol. 37, no. 12, pp. 7328–7345, 2025.

[15] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, J. Zhou, "Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation", *Proceedings of the VLDB Endowment*, vol. 17, no. 5, pp. 1132–1145, 2024.

[16] Z. Yao, G. Sun, L. Borchmann, G. Nuti, Z. Shen, M. Deng, B. Zhai, H. Zhang, A. Li, Y. He, "Arctic-Text2SQL-R1: Simple Rewards, Strong Reasoning in Text-to-SQL", *arXiv preprint*, arXiv: 2505.20315, 2025.

[17] T. Jin, Y. Choi, Y. Zhu, D. Kang, "Pervasive Annotation Errors Break Text-to-SQL Benchmarks and Leaderboards", *arXiv preprint*, arXiv: 2601.08778, 2026.

[18] BIRD Team, Google Cloud, "LiveSQLBench: A Dynamic, Contamination-Free Benchmark for Text-to-SQL", 2025. [Online]. Available: `https://livesqlbench.ai`.

[19] Open-Meteo, "Open-Meteo: Free Weather API", [Online]. Available: `https://open-meteo.com`.

Ringraziamenti

Il mio percorso universitario non è mai stato lineare. Volevo studiare filosofia, affascinata da domande che considero ancora oggi le più autentiche: cosa rende possibile la conoscenza e in quale momento un'idea diventa azione. Ho scoperto poi che l'informatica quelle domande le aveva rese concrete, traducendole in linguaggi, architetture e sistemi che imparano. E dentro l'informatica, l'intelligenza artificiale: l'unico campo che le aveva messe al centro fin dall'inizio.

Ho interrotto gli studi nel 2018, con solo due esami da dare. Ho iniziato a lavorare, ho imparato sul campo, ho sbagliato, ho ricominciato – più e più volte. Quest'anno, per la prima volta, ho avuto il tempo – e la maturità – per tornare e chiudere un cerchio. Questa tesi è nata nell'unico posto in cui poteva nascere: nell'intersezione esatta tra ciò che ho studiato e ciò che costruisco ogni giorno.

Desidero innanzitutto ringraziare il Prof. Marco Di Felice per aver accolto il mio contributo con apertura genuina, lasciandomi portare in queste pagine una prospettiva maturata sul campo. Un ringraziamento sentito va al Dott. Leonardo Ciabattini, correlatore di rara disponibilità e competenza, il cui supporto è stato prezioso in ogni fase del lavoro. A tutti i docenti incontrati lungo questo percorso: molte delle cose che mi avete insegnato le ho capite dopo, quando ne avevo davvero bisogno.

Grazie a mamma Cristina e papà Graziano: avete aspettato questo momento con una pazienza che non ho mai smesso di sentire, anche nei periodi in cui non ve lo dicevo. Non è scontato avere genitori che credono in una

figlia anche quando la strada che sceglie non è quella che avevano immaginato. Questa tesi è tutta vostra.
Grazie ai miei nonni Gianni, Lalla e Tilde – non ci sono più, ma sono stati le stelle polari della mia vita e continuano a esserlo.
Grazie a Roma: se oggi giro il mondo con curiosità invece che con paura, lo devo a te.

Grazie a Mario – amico, socio e compagno di vita. Siamo diversi in quasi tutto e, forse, è proprio questo che ha reso possibile tutto quello che abbiamo costruito insieme. Grazie per avermi sempre spinta oltre i miei limiti e le mie più grandi paure. Questa vita non sarebbe stata così viva senza di te.

Grazie a Nicole: con te basta uno sguardo – a Bologna, all'Havana, ad Hanoi – per capirsi senza spiegazioni. Sei la mia certezza.
Grazie a Luca: con te anni e chilometri non hanno mai intaccato la stima e l'affetto; ogni volta che ci ritroviamo, il tempo sembra non essere passato.
Grazie a Pier: ti ho visto attraversare le tue paure invece di aggirarle, e uscirne ogni volta più forte e più consapevole di chi sei.
Grazie a Francesca: ti ho vista scegliere, non subire – sei la dimostrazione che si può avere coraggio senza perdere tenerezza.
Grazie a Nicolina: trent'anni di amicizia, e tu che affronti ogni sfida con ironia e il sorriso in bocca, anche quando avresti tutto il diritto di non farlo; oggi sei una mamma incredibile.

Grazie ai miei compagni di avventura in Boosha AI e Parla: ho imparato insieme a voi che il talento apre le porte, ma è l'ossessione – quella silenziosa, quotidiana, che non si spegne – a tenerle aperte e a crearne di nuove. Ho imparato che le competenze contano, ma non sono mai state il vero discrimine: ciò che ci ha tenuti insieme e ci ha fatti crescere è qualcosa di più raro – la disponibilità a metterci in discussione, ogni volta, senza riserve. Siamo le persone giuste nel momento e nel posto giusto.

Ho iniziato questo percorso cercando di capire come la conoscenza si trasforma in azione: la risposta era nei libri ma anche in tutto quello che ho vissuto nel mezzo.