

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

UNIVERSITÀ DEGLI STUDI DI MODENA

E REGGIO EMILIA

Dipartimento di Ingegneria "Enzo Ferrari"

Corso di Laurea Magistrale in Ingegneria Informatica (LM-32)

Question Answering su Open Data: valutazione end-to-end di strategie di recupero e configurazioni agentiche in LakeGen

Relatore:

Prof. Giovanni Simonini

Correlatore:

Dott. Bilel Arfaoui

Candidato:

Gabriele Martinelli

Matricola 207026

ANNO ACCADEMICO 2025/2026

Indice

Elenco delle Figure **5**

Elenco delle Tabelle **6**

1 Introduzione **7**

1.1 Motivazione e contesto 7

1.2 Problema affrontato 8

1.3 Obiettivi del lavoro 8

1.4 Contributi 9

1.5 Struttura della tesi 9

2 Contesto e fondamenti **11**

2.1 Open Data e Open Data Lake 11

2.2 Tabular Question Answering 14

2.2.1 Question Answering su una singola tabella 15

2.2.2 Question Answering su più tabelle 16

2.2.3 Text-to-SQL e generazione di codice 17

2.2.4 Configurazioni Known-Table e Open-Table 18

2.3 Recupero 19

2.3.1 Recupero lessicale 20

2.3.2 Recupero semantico 21

2.3.3 Recupero ibrido 22

2.3.4 Riformulazione agentica della query 23

2.4 Large Language Model e sistemi agentici 24

2.4.1 Capacità ed efficienza del modello 25

2.4.2 Tool calling e operazioni deterministiche 25

2.4.3 Agenti specializzati . . . 26
2.4.4 Revisione, feedback degli errori e intervento umano . . . 27
2.4.5 Requisiti di sicurezza dei sistemi agentici . . . 28
2.5 Riepilogo del capitolo . . . 29
3 Stato dell'arte e lavori correlati 31
3.1 Table Retrieval e Dataset Discovery . . . 31
3.2 Retrieval-Augmented Generation su tabelle . . . 33
3.2.1 Ricerca agentica per parole chiave come alternativa al retrieval vettoriale . . . 35
3.3 Scoperta di join e union . . . 36
3.4 Valutazione del Text-to-SQL e del Tabular Question Answering . . . 38
3.5 Approcci agentici al Tabular Question Answering . . . 40
3.6 Benchmark esistenti e pratiche di valutazione . . . 40
3.7 Limiti dei lavori esistenti e research gap . . . 42
3.8 Sintesi del capitolo . . . 43
4 OrQA: generazione di benchmark basati su Open Data 44
4.1 Requisiti del benchmark . . . 44
4.2 Workflow OrQA pubblicato . . . 45
4.2.1 Acquisizione degli Open Data . . . 45
4.2.2 Discovery e valutazione dei candidati . . . 46
4.2.3 Generazione delle domande e delle query di riferimento . . . 46
4.2.4 Dataset pubblicato . . . 47
4.3 La pipeline OrQA aggiornata . . . 47
4.3.1 Pipeline multi-portale configurabile . . . 48
4.3.2 Strategie alternative per la discovery dei candidati . . . 49
4.3.3 Separazione tra piano, codice e validazione . . . 49
4.3.4 Sanificazione delle query generate . . . 49
4.3.5 Solver round-trip del benchmark . . . 50
4.4 Artefatti generati e supporto alla valutazione . . . 51
4.5 Limiti e minacce alla validità . . . 51
4.6 Sintesi del capitolo . . . 52

5 Architettura di LakeGen e configurazioni valutate **53**

5.1 Requisiti di sistema e obiettivi di progettazione 53

5.2 Architettura complessiva 54

5.2.1 Flusso di elaborazione 54

5.2.2 Flusso dei dati, controllo e gestione degli errori 55

5.3 Recupero delle tabelle candidate 55

5.3.1 Elaborazione della domanda e delle parole chiave 55

5.3.2 Recupero lessicale 56

5.3.3 Recupero semantico denso 56

5.3.4 Recupero ibrido 56

5.3.5 Riformulazione iterativa della query 57

5.4 Individuazione e selezione dei dataset 57

5.4.1 Ispezione dei dataset e analisi dei metadati 57

5.4.2 Selezione e combinazione delle tabelle 58

5.5 Piano semantico, generazione ed esecuzione del codice 58

5.5.1 Piano semantico e contesto del coder 58

5.5.2 Generazione ed esecuzione del programma Python 59

5.5.3 Feedback sugli errori e nuovi tentativi 59

5.5.4 Generazione della risposta finale 60

5.6 Configurazioni di interazione e orchestrazione 60

5.6.1 Accesso agentico e contesto orchestrato 60

5.6.2 Intervento umano ed esecuzione autonoma 60

5.7 Configurazioni del contesto del coder 61

5.8 Modelli linguistici 61

5.9 Controlli di sicurezza e di esecuzione 62

5.10 Configurazione e riproducibilità degli esperimenti 62

5.11 Sintesi del capitolo 63

6 Valutazione sperimentale **64**

6.1 Progettazione sperimentale 64

6.1.1 Benchmark e configurazioni 64

6.1.2 Metriche di valutazione . . . 65
6.1.3 Criteri di confronto . . . 66
6.2 Risultati complessivi . . . 66
6.3 Effetto della strategia di recupero . . . 69
6.3.1 Correttezza del risultato . . . 69
6.3.2 Recupero e selezione delle tabelle . . . 70
6.4 Modello linguistico e modalità di interazione . . . 72
6.4.1 Confronto tra GPT-OSS e Llama . . . 72
6.4.2 Accesso agentico e contesto orchestrato . . . 72
6.5 Effetto del contesto del coder . . . 74
6.6 Risultati sul benchmark esteso in modalità agentica . . . 75
6.7 Compromesso tra efficacia ed efficienza . . . 77
6.8 Domande su una o più tabelle . . . 78
6.9 Variabilità tra esecuzioni . . . 79
6.10 Discussione e limiti della valutazione . . . 80
6.10.1 Interpretazione dei risultati . . . 80
6.10.2 Limiti del benchmark e delle misure . . . 81
6.11 Sintesi del capitolo . . . 81

7 Conclusioni **83**

7.1 Bilancio del lavoro svolto . . . 83
7.2 Significato dei risultati . . . 83
7.3 Limiti delle conclusioni . . . 84
7.4 Sviluppi futuri . . . 84
7.5 Considerazioni finali . . . 85

Ringraziamenti **86**

Bibliografia **87**

Elenco delle Figure

2.1 Confronto tra le configurazioni known-table e open-table. 19

4.1 Fasi principali della pipeline OrQA aggiornata. 48

5.1 Architettura complessiva e flusso di elaborazione di LakeGen. 54

6.1 Effetti dei fattori sperimentali 69
6.2 Interazione tra modello, modalità e recupero 73
6.3 Effetto del contesto del coder 74
6.4 Efficacia e risorse impiegate 77
6.5 Risultati per numero di tabelle 79

Elenco delle Tabelle

6.1 Fattori del disegno sperimentale 65
6.2 Risultati delle configurazioni su NYC 67
6.3 Risultati delle configurazioni su UK 68
6.4 Effetti delle configurazioni sull'exact match 70
6.5 Qualità dell'ordinamento delle tabelle 71
6.6 Costo del coder per contesto 75
6.7 Effetti sul benchmark esteso agentico 76
6.8 Riduzione dei token per modalità 78
6.9 Variabilità tra esecuzioni 80

1. Introduzione

1.1 Motivazione e contesto

Le iniziative di Open Data hanno aumentato l'accesso alle informazioni strutturate e tabellari pubblicate da governi e istituzioni. Il loro obiettivo è quello di rendere le informazioni pubbliche più trasparenti e accessibili, permettendo ai cittadini di comprendere e monitorare fenomeni di interesse collettivo. Di conseguenza, un numero elevato di dataset è accessibile attraverso portali pubblici e può essere utilizzato per condurre analisi in ambiti differenti [12].

La disponibilità degli Open Data non garantisce che le informazioni necessarie siano facili da trovare e utilizzare. I dataset pubblici sono spesso distribuiti in formati differenti, descritti da metadati incompleti o incoerenti e organizzati secondo schemi eterogenei. Un utente può quindi sapere quale informazione sta cercando senza conoscere il dataset che la contiene o la terminologia utilizzata per descriverlo. Inoltre, per ottenere una risposta può essere necessario individuare, ispezionare, pulire e combinare più tabelle. La dimensione delle raccolte rende il processo ancora più complesso, poiché il sistema deve bilanciare l'efficacia del recupero con la latenza, il consumo di risorse e il costo di indicizzazione [14, 3].

Nel problema considerato, la domanda in linguaggio naturale costituisce l'ingresso della pipeline. Molti approcci di Tabular Question Answering sono però valutati in configurazioni nelle quali la tabella pertinente è già disponibile. Nello scenario considerato in questa tesi, il sistema riceve invece soltanto la domanda e deve individuare le fonti, selezionarle, combinarle quando necessario e generare le operazioni richieste per produrre la risposta [12, 13].

Una query sintatticamente corretta può infatti produrre un risultato incompleto o non pertinente se viene applicata alle tabelle sbagliate; viceversa, recuperare una fonte valida non è sufficiente se il sistema non riesce a interpretarla o a combinarla correttamente. LakeGen affronta questi passaggi in una pipeline agentica per il Question Answering su Open Data e costituisce il sistema oggetto della valutazione svolta in questa tesi [13].

1.2 Problema affrontato

I lavori esistenti affrontano diverse parti del processo. I metodi di table discovery studiano il recupero di tabelle pertinenti e la ricerca di tabelle che possano essere combinate mediante join o union. Il Tabular Question Answering e il Text-to-SQL si concentrano invece sulla generazione di operazioni o risposte a partire da tabelle già identificate. OrQA collega i due problemi associando domande in linguaggio naturale a tabelle Open Data reali e a interrogazioni di riferimento eseguibili, mentre LakeGen integra il recupero, la selezione delle fonti e la generazione della risposta in un'unica procedura [14, 12, 13].

Questi aspetti sono spesso valutati separatamente. Una valutazione limitata al recupero non permette di stabilire se le tabelle individuate siano poi utilizzate correttamente. Allo stesso tempo, osservare soltanto il risultato del programma non consente di capire se un errore dipenda dal recupero, dalla selezione delle fonti o dalla generazione e dall'esecuzione del codice. È quindi utile osservare sia le singole fasi sia il comportamento dell'intera pipeline [3, 8].

Anche le caratteristiche degli Open Data devono essere considerate. Più dataset, o più combinazioni di dataset, possono consentire di rispondere alla stessa domanda. Se la ground truth ne rappresenta una sola, un sistema può individuare fonti valide ed essere comunque penalizzato [12]. Inoltre, le raccolte reali presentano metadati, schemi e livelli di qualità eterogenei. I risultati devono quindi essere interpretati rispetto ai dati e alle configurazioni utilizzati negli esperimenti.

1.3 Obiettivi del lavoro

La tesi valuta come alcune scelte progettuali influenzino le prestazioni di LakeGen quando le tabelle necessarie non sono note in anticipo. In particolare, esamina se le strategie di recupero lessicale, semantico e ibrido producano differenze nella qualità delle fonti individuate e nella correttezza del risultato del programma.

L'analisi considera inoltre l'effetto del modello linguistico, della modalità con cui esso accede ai candidati e delle informazioni fornite al generatore di codice. Per ciascuna scelta, la tesi confronta la qualità del risultato con la latenza e i token impiegati. Le misure del recupero, della selezione delle

tabelle, dell'esecuzione e del risultato permettono di osservare in quali fasi emergano le principali difficoltà, senza attribuire automaticamente ogni risultato errato a una singola causa.

1.4 Contributi

Il lavoro predispone un ambiente sperimentale configurabile per LakeGen e vi integra quattro strategie di recupero: BM25, recupero semantico denso, fusione ibrida pesata e Reciprocal Rank Fusion [15, 9, 2]. L'ambiente consente di confrontare anche due modelli linguistici, l'accesso agentico agli strumenti e il contesto orchestrato, oltre a tre livelli di informazione forniti al generatore di codice.

La tesi applica un protocollo di valutazione ai casi generati da OrQA per i portali di New York City e del Regno Unito. Il protocollo combina misure del recupero e della selezione con la correttezza del risultato del programma, la latenza e il consumo di token. L'analisi mette così in relazione i risultati intermedi con quelli dell'intera procedura e documenta i limiti entro cui interpretare le differenze osservate.

1.5 Struttura della tesi

Il resto della tesi è organizzato come segue. Il Capitolo 2 introduce gli Open Data e gli open data lake, il Tabular Question Answering, le tecniche di recupero e i sistemi agentici basati su modelli linguistici. Il Capitolo 3 esamina gli approcci esistenti al recupero e all'interrogazione delle tabelle, i benchmark disponibili e i limiti che motivano questo lavoro.

Il Capitolo 4 presenta OrQA: ne descrive il processo di generazione pubblicato, la pipeline aggiornata, gli artefatti impiegati negli esperimenti e i limiti del benchmark. Il Capitolo 5 illustra l'architettura di LakeGen, le fasi che portano dalla domanda alla risposta e le configurazioni confrontate nella valutazione.

Il Capitolo 6 espone il protocollo sperimentale e le metriche, quindi analizza i risultati del recupero e della pipeline end-to-end. Esamina inoltre l'effetto delle strategie di recupero, dei modelli linguistici, delle modalità di interazione e del contesto fornito al generatore di codice,

insieme alle risorse impiegate e ai limiti della valutazione. Infine, il Capitolo 7 riprende i risultati principali, ne discute i limiti e indica possibili sviluppi futuri.

2. Contesto e fondamenti

Questo capitolo introduce i concetti necessari per comprendere il sistema e le configurazioni sperimentali considerate nelle parti successive della tesi. Come discusso nel Capitolo 1, rispondere a domande in linguaggio naturale su Open Data richiede più della semplice esecuzione di una query su un database noto: i dataset pertinenti devono prima essere individuati, interpretati e, se necessario, combinati.

Il capitolo parte dai portali Open Data e dalle principali caratteristiche dei data lake tabellari. Successivamente introduce il Tabular Question Answering, distinguendo le domande single-table da quelle multi-table e le configurazioni known-table da quelle open-table. Vengono quindi presentati gli approcci di recupero lessicale, semantico e ibrido. Infine, il capitolo descrive i Large Language Model, le architetture agentiche, l'uso degli strumenti e i relativi requisiti di sicurezza.

Questi concetti permettono di stabilire una terminologia comune e di chiarire il ruolo dei principali componenti che saranno ripresi nella descrizione del sistema e nella valutazione sperimentale.

2.1 Open Data e Open Data Lake

Con Open Data si indicano i dati resi disponibili per l'accesso, il riutilizzo e la ridistribuzione con limitate restrizioni legali e tecniche. Le amministrazioni pubbliche utilizzano portali Open Data per pubblicare informazioni in diversi ambiti come demografia, trasporti, spesa pubblica, sanità, istruzione e ambiente [12, 14].

I portali Open Data organizzano generalmente i propri contenuti in dataset e nelle risorse ad essi associate. Un dataset rappresenta una raccolta di dati pubblicata ed è solitamente descritto da metadati come titolo, descrizione, editore, argomento, licenza e data di pubblicazione o aggiornamento. Le risorse permettono di accedere ai dati effettivi, sotto forma di file scaricabili, come CSV, fogli di calcolo o file JSON, oppure tramite endpoint API.

La distinzione tra dataset e risorse è importante durante la ricerca dei dati. I motori di ricerca indicizzano e ordinano generalmente i dataset attraverso i loro metadati, mentre l'informazione richiesta dall'utente può trovarsi soltanto nelle risorse associate. Di conseguenza, la presenza di

metadati pertinenti non garantisce che le risorse contengano i dati necessari per rispondere alla domanda.

I portali Open Data possono essere realizzati mediante diverse piattaforme software, come CKAN, Socrata e OpenDataSoft. Queste piattaforme forniscono interfacce per pubblicare dataset, cercare nei metadati, accedere alle risorse e, in alcuni casi, interrogare singoli record tramite API.

Sebbene le piattaforme Open Data offrano operazioni generalmente simili, i loro modelli dei dati e le loro interfacce di interrogazione possono essere differenti. Per esempio, possono utilizzare identificatori dei dataset, campi di metadati, formati delle risorse, meccanismi di paginazione e metodi di filtraggio diversi. Alcune risorse possono essere interrogate da remoto, mentre altre devono essere scaricate ed elaborate localmente. Un sistema che opera su più portali deve quindi gestire queste differenze specifiche delle piattaforme separatamente dal resto del processo di ricerca dei dataset e di risposta alle domande.

Dal punto di vista della gestione dei dati, una raccolta di tabelle Open Data può essere considerata un data lake tabellare. Un data lake contiene fonti eterogenee che non devono rispettare uno schema globale comune prima dell'acquisizione. Questo facilita la raccolta dei dati e conserva la rappresentazione adottata da ciascun fornitore, ma aumenta il lavoro di interpretazione e integrazione richiesto agli utenti e alle applicazioni [3, 14].

Tabelle che descrivono lo stesso fenomeno o la stessa entità possono utilizzare nomi degli attributi, rappresentazioni dei valori, unità o livelli di granularità differenti. Per esempio, la stessa entità geografica può essere rappresentata mediante un nome, un codice amministrativo o coordinate geografiche. Anche le date possono utilizzare formati o livelli di precisione differenti, come giorni, mesi o anni. Allo stesso modo, i valori numerici possono essere espressi mediante unità, scale o convenzioni decimali diverse. Queste differenze possono richiedere una normalizzazione prima che le tabelle correlate possano essere combinate [14, 3].

Un altro problema è che concetti equivalenti possono essere rappresentati mediante termini o nomi di attributi differenti. Per esempio, una tabella può utilizzare l'etichetta "popolazione", mentre un'altra può descrivere la stessa informazione con etichette come "residenti" o "abitanti". Di conseguenza, la sola corrispondenza lessicale esatta potrebbe non individuare tutti i dataset potenzialmente pertinenti [14].

Può verificarsi anche il problema inverso, quando etichette simili si riferiscono a concetti

diversi. Per esempio, una colonna chiamata “valore” potrebbe contenere il numero di abitanti, una percentuale o un importo monetario. Allo stesso modo, una colonna chiamata “codice” potrebbe contenere un identificatore geografico in una tabella e un identificatore organizzativo in un’altra. La somiglianza tra i nomi delle colonne può quindi essere utile, ma non è sufficiente a stabilire la compatibilità semantica.

Un’altra fonte di incertezza è la qualità degli Open Data. Le tabelle possono contenere valori mancanti, record duplicati, tipi di dati misti, righe non valide o valori usati come indicatori informali di dati mancanti. L’inferenza dello schema può quindi fallire quando, per esempio, la stessa colonna contiene sia valori testuali sia numerici. I duplicati possono inoltre alterare conteggi e aggregazioni, mentre i record incompleti possono ridurre la copertura di un join.

Anche i metadati possono essere incompleti o incoerenti. Titoli e descrizioni possono essere generici, gli attributi possono non essere documentati e i metadati possono non essere aggiornati quando cambia la risorsa corrispondente. Dataset correlati pubblicati da organizzazioni diverse possono inoltre utilizzare terminologie differenti. Poiché la ricerca tradizionale nei portali si basa generalmente sui metadati testuali, queste limitazioni possono impedire che un dataset pertinente compaia tra i candidati recuperati [14, 3].

La dimensione del repository introduce un’ulteriore difficoltà nella ricerca dei dataset. Lake-Bench valuta metodi di table discovery su raccolte che contengono più di 16 milioni di tabelle reali e fino a 100 milioni di colonne [3]. A questa scala non è pratico confrontare ogni query con ogni tabella o colonna. Sono quindi necessari meccanismi di indicizzazione e generazione dei candidati per ridurre il numero di risorse esaminate durante ogni ricerca.

Dopo aver recuperato le tabelle candidate, il sistema deve stabilire se sia necessario combinarle. Join e union rappresentano due operazioni fondamentali per l’integrazione dei dati.

Un join associa i record attraverso una o più colonne compatibili e può aggiungere attributi complementari. Una union, invece, allinea attributi compatibili e aggiunge record a una raccolta esistente. In entrambi i casi, una corretta integrazione richiede più della semplice corrispondenza tra i nomi delle colonne.

La sovrapposizione dei valori può indicare che due colonne potrebbero supportare un join. Tuttavia, i valori condivisi non garantiscono che le colonne rappresentino lo stesso concetto, poiché numeri, abbreviazioni o codici comuni possono comparire anche in domini non correlati. Al

contrario, colonne semanticamente compatibili possono avere una sovrapposizione limitata perché le rispettive tabelle coprono aree geografiche o periodi temporali differenti [3].

Anche la unionability richiede compatibilità semantica tra gli attributi. Nargesian et al. definiscono due tabelle come unionable quando contengono attributi appartenenti agli stessi domini. La loro formulazione considera evidenze derivate dalla sovrapposizione dei valori, dai domini semantici e dalle rappresentazioni in linguaggio naturale [14].

Queste osservazioni permettono di distinguere la compatibilità fisica dalla correttezza semantica. Due tabelle possono essere tecnicamente joinable o unionable e produrre comunque un risultato irrilevante o fuorviante rispetto alla domanda dell'utente. L'integrazione dei dataset deve quindi considerare sia la possibilità di eseguire un'operazione, sia la capacità della tabella risultante di conservare il significato previsto.

2.2 Tabular Question Answering

Il Tabular Question Answering, comunemente abbreviato in TQA, ha l'obiettivo di generare risposte a domande in linguaggio naturale utilizzando informazioni rappresentate in forma tabellare. Il suo scopo principale è permettere agli utenti di ottenere informazioni senza dover conoscere o scrivere SQL o codice per l'elaborazione dei dati [12, 5].

Un sistema TQA riceve in ingresso una domanda in linguaggio naturale e una o più tabelle. A seconda della configurazione, le tabelle possono essere fornite direttamente oppure individuate attraverso un componente di recupero. L'output può essere il valore di una cella, un valore booleano, un numero, una lista, una tabella o una spiegazione in linguaggio naturale.

Le operazioni necessarie per produrre una risposta possono variare notevolmente. Per esempio, alcune domande richiedono soltanto di estrarre un valore esistente, mentre altre possono richiedere di filtrare record, ordinare valori, contare righe, individuare valori minimi o massimi, calcolare aggregazioni o confrontare sottoinsiemi differenti. Le domande più complesse possono richiedere l'applicazione di più operazioni in un ordine specifico.

Queste diverse operazioni evidenziano la distinzione tra la comprensione della tabella e l'esecuzione delle operazioni richieste. Un sistema può individuare correttamente le colonne pertinenti ma non applicare l'aggregazione prevista. Al contrario, può generare un'operazione eseguibile che

utilizza i record sbagliati. La sola corretta esecuzione non è quindi sufficiente per stabilire se la risposta sia corretta.

2.2.1 Question Answering su una singola tabella

Una domanda single-table richiede informazioni contenute in una sola tabella. Nella configurazione known-table, questa tabella viene fornita insieme alla domanda. Il sistema non deve quindi cercare in un repository o decidere quale dataset utilizzare. Una volta disponibile la tabella, il suo compito principale è interpretare la domanda in linguaggio naturale, individuare le righe e le colonne pertinenti e stabilire quali operazioni siano necessarie per produrre la risposta.

Per esempio, una domanda potrebbe richiedere il valore più alto di una colonna dopo aver applicato un filtro a un'altra. Il sistema deve individuare la condizione di filtro richiesta, selezionare le righe pertinenti, trovare il valore massimo e formattare il risultato. Sebbene sia coinvolta una sola tabella, un errore in uno qualsiasi di questi passaggi può influenzare la risposta finale.

I Large Language Model possono affrontare queste domande mediante approcci differenti. Per esempio, il modello può ricevere direttamente nel proprio contesto una rappresentazione della tabella oppure generare codice eseguibile per elaborarla. DataBench valuta sia approcci in-context sia approcci basati sulla generazione di codice su dataset reali ed eterogenei [5]. La valutazione considera differenti tipi di risposta e strutture tabellari, mostrando che le prestazioni dipendono non soltanto dal modello selezionato, ma anche dalle caratteristiche della domanda e della tabella in ingresso.

Anche la quantità di contenuto della tabella fornita al modello è una scelta progettuale importante. Includere tutte le righe conserva le informazioni disponibili, ma può superare la finestra di contesto del modello o introdurre contenuti irrilevanti. Fornire soltanto lo schema, invece, riduce la quantità di contesto ma può omettere valori necessari per interpretare le colonne o formulare l'operazione corretta.

Una configurazione known-table permette di analizzare le fasi successive al recupero separatamente dalla ricerca dei dataset. Questa condizione, tuttavia, non rappresenta lo scenario Open Data considerato nella tesi.

2.2.2 Question Answering su più tabelle

Una domanda multi-table richiede informazioni distribuite tra più tabelle. Queste tabelle possono contenere attributi complementari, record compatibili o informazioni intermedie necessarie per produrre la risposta finale.

Quando le tabelle richieste sono già disponibili, il sistema deve individuare le relazioni tra di esse, selezionare colonne appropriate per i join o allineare attributi compatibili per le union e costruire una sequenza di operazioni che preservi il significato dei dati. Rispetto a una domanda single-table, gli errori possono verificarsi non soltanto durante il filtraggio o l'aggregazione, ma anche quando vengono individuate e applicate le relazioni tra le tabelle [12, 13].

Un join può essere tecnicamente eseguibile anche quando utilizza colonne semanticamente incompatibili. Una relazione molti-a-molti può inoltre duplicare righe nella tabella risultante e influenzare il risultato di un'aggregazione. Allo stesso modo, una union può combinare valori espressi con unità o livelli di granularità differenti. Anche l'ordine in cui vengono applicati filtri, join, union e aggregazioni può influenzare il risultato finale [3, 12].

Se le tabelle richieste provengono da portali o sistemi differenti, il processo di integrazione diventa ancora più complesso. Una domanda può richiedere un dataset esposto tramite API e un altro disponibile soltanto come file scaricabile. Il sistema deve ottenere entrambe le fonti, trasformarle in rappresentazioni compatibili e gestire le differenze nello schema e nella qualità dei dati.

OrQA comprende domande in linguaggio naturale sia single-table sia multi-table associate a tabelle Open Data reali e query SQL eseguibili [12]. La sua struttura mostra l'importanza di associare a ogni domanda sia i dataset richiesti sia le operazioni necessarie per ottenere la risposta.

La distinzione tra domande single-table e multi-table non è determinata dall'apparente complessità della domanda dell'utente. Una domanda apparentemente semplice può richiedere più fonti perché nessun singolo dataset contiene tutti gli attributi necessari. Al contrario, una domanda più complessa in linguaggio naturale può ricevere risposta mediante una sola tabella ben strutturata.

2.2.3 Text-to-SQL e generazione di codice

Il Text-to-SQL traduce una domanda in linguaggio naturale in una query SQL su uno schema relazionale. La query generata fornisce una rappresentazione esplicita ed eseguibile delle operazioni selezionate dal modello. È quindi possibile esaminare prima dell'esecuzione le tabelle, le colonne, i predicati, i join e le aggregazioni utilizzati.

SQL è particolarmente adatto quando i dati pertinenti sono già stati caricati in un database relazionale. La sua natura dichiarativa permette al sistema di esprimere il risultato desiderato senza implementare direttamente il processo di esecuzione. Il database può quindi validare la sintassi, ottimizzare la query e applicare i permessi di accesso.

Le risorse Open Data non sono sempre archiviate in un database relazionale condiviso. Possono invece essere distribuite tra file scaricabili e API eterogenee. La generazione di codice general-purpose può offrire maggiore flessibilità in questo contesto. Il codice Python, per esempio, può caricare file CSV, chiamare API, normalizzare valori, combinare data frame, calcolare statistiche e preparare l'output richiesto.

Questa flessibilità è un vantaggio, ma aumenta anche il numero dei possibili errori. Il codice generato può fare riferimento a file o colonne errati, utilizzare un join non appropriato, modificare involontariamente i valori o produrre un risultato nel formato sbagliato. Può inoltre tentare operazioni al di fuori dell'ambito previsto se l'ambiente di esecuzione non limita le funzionalità disponibili.

L'esecuzione del codice può anche fornire feedback utile. Errori di sintassi, colonne mancanti, incompatibilità tra tipi e operazioni fallite possono essere restituiti al componente di generazione, che può quindi tentare di correggere il codice. Tuttavia, un codice eseguito con successo può comunque tradurre in modo errato la domanda originale [13].

Per ogni domanda in linguaggio naturale, OrQA identifica le tabelle necessarie e fornisce una query SQL di riferimento eseguibile. Il risultato prodotto dal sistema può quindi essere confrontato con quello ottenuto eseguendo tale query, senza richiedere che il codice generato sia testualmente identico a quello di riferimento [12].

Infatti, un limite della valutazione exact-match è che query o programmi differenti possono realizzare le stesse operazioni e produrre lo stesso risultato. La corrispondenza testuale esatta con il

codice di riferimento è quindi meno informativa dell'equivalenza dei risultati di esecuzione e della correttezza della risposta finale.

2.2.4 Configurazioni Known-Table e Open-Table

Una distinzione fondamentale riguarda la conoscenza delle tabelle corrette prima che inizi il processo di risposta. In una configurazione known-table, il sistema può concentrarsi sull'interpretazione della domanda e sulla generazione delle operazioni richieste. In una configurazione open-table, le tabelle pertinenti devono prima essere individuate all'interno di una raccolta più ampia [12, 13].

In una configurazione open-table, il sistema deve combinare il recupero delle tabelle con la generazione della risposta. Un componente di recupero individua un insieme limitato di tabelle candidate. Questo è utilizzato per selezionare le tabelle di riferimento che saranno combinate quando necessario e interrogate. La risposta finale dipende quindi sia dalla qualità del recupero sia dalla correttezza delle successive fasi di elaborazione.

Se una tabella richiesta non è presente tra i candidati recuperati, i componenti successivi non possono ricostruire attraverso il solo ragionamento l'informazione mancante. Se la tabella viene recuperata ma non selezionata, il problema riguarda la selezione dei dataset e non il recupero. Se le tabelle corrette vengono selezionate ma combinate in modo errato, l'errore si verifica durante la generazione o l'esecuzione del codice.

Tutti questi casi possono produrre una risposta finale errata, ma richiedono interventi differenti. Una valutazione a livello del retriever è necessaria per stabilire se le tabelle richieste siano presenti tra i candidati. Una valutazione end-to-end è invece necessaria per stabilire se il sistema completo utilizzi correttamente i dati recuperati.

Questa tesi valuta una configurazione open-table. Il sistema parte da una domanda in linguaggio naturale e deve recuperare, selezionare e interrogare le risorse Open Data pertinenti. Questo scenario è il più vicino all'interazione di un utente che conosce l'informazione di cui ha bisogno, ma non sa come siano organizzati i dataset disponibili.

La Figura 2.1 sintetizza la differenza tra le due configurazioni. Nel caso known-table, la tabella corretta costituisce parte dell'input; nel caso open-table, retrieval e selezione precedono necessariamente l'elaborazione dei dati.

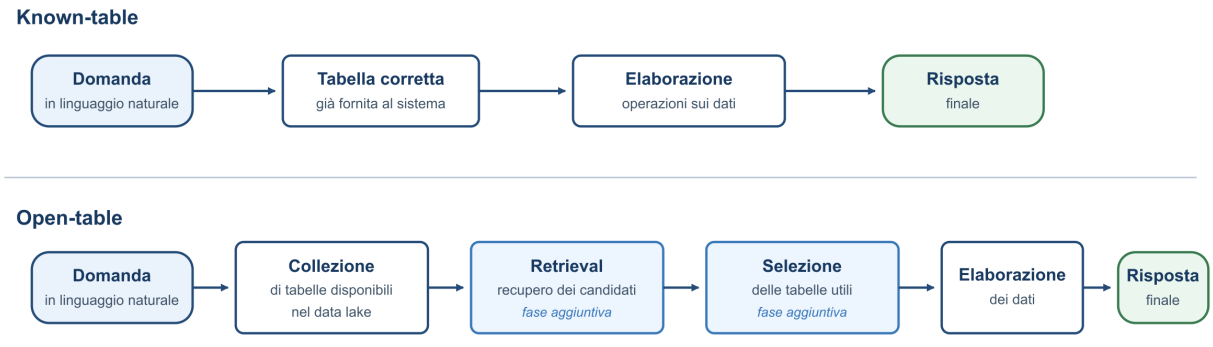


Figura 2.1: Confronto tra le configurazioni known-table e open-table.

2.3 Recupero

L'obiettivo del recupero è individuare un insieme limitato di dataset candidati da un repository che può contenere migliaia o milioni di tabelle. Il suo scopo è ridurre lo spazio di ricerca, assicurandosi che i dataset necessari per rispondere alla domanda siano inclusi tra i candidati recuperati [8, 1].

Prima di poter essere recuperato, ogni dataset deve essere convertito in una rappresentazione indicizzabile. Questa rappresentazione può includere titolo, descrizione, tag, editore, nome della risorsa, nomi delle colonne, tipi di dati inferiti e alcuni valori selezionati. La rappresentazione scelta determina quali informazioni possono essere usate dal sistema di recupero.

Utilizzare soltanto il titolo e la descrizione produce un indice compatto, ma la sua efficacia dipende dalla completezza dei metadati disponibili. Includere i nomi delle colonne può mostrare informazioni non presenti nella descrizione. Anche alcuni valori di esempio possono migliorare il recupero di dataset che contengono entità o categorie specifiche, ma aumentano la dimensione dell'indice e possono introdurre contenuti poco informativi.

I metodi di recupero possono essere suddivisi in approcci lessicali, semantici e ibridi. Ciascuna famiglia stima la pertinenza in modo diverso e può quindi recuperare un insieme differente di dataset candidati.

2.3.1 Recupero lessicale

Il recupero lessicale confronta le parole presenti nella domanda dell'utente con quelle contenute nella rappresentazione indicizzata di ogni dataset. Una semplice ricerca per parole chiave recupera i dataset i cui metadati contengono uno o più termini della query. I metodi più avanzati considerano anche la frequenza di ogni parola e quanto essa sia utile per distinguere i dataset pertinenti dal resto della raccolta.

BM25 è una funzione probabilistica di ranking ampiamente utilizzata per il recupero lessicale [15]. Una formulazione comune del punteggio assegnato a un documento D e a una query Q è:

$$\text{BM25}(D, Q) = \sum_{q_i \in Q} \text{IDF}(q_i) \frac{f(q_i, D)(k_1 + 1)}{f(q_i, D) + k_1 \left(1 - b + b \frac{|D|}{\text{avgdl}}\right)},$$

dove $f(q_i, D)$ è la frequenza del termine q_i nel documento D, $|D|$ è la lunghezza del documento e avgdl è la lunghezza media dei documenti. I parametri k_1 e b controllano la saturazione della frequenza del termine e la normalizzazione rispetto alla lunghezza del documento.

L'inverse document frequency attribuisce maggiore importanza alle parole che compaiono in un numero limitato di documenti. Le parole comuni sono meno utili per individuare un dataset pertinente e hanno quindi un'influenza minore sul suo punteggio. Le parole più rare possono invece aiutare a distinguerlo dal resto della raccolta. La saturazione della term frequency limita inoltre l'effetto delle occorrenze ripetute, evitando che un documento ottenga un punteggio alto soltanto perché la stessa parola compare molte volte.

Motori di ricerca come Elasticsearch e Solr utilizzano indici invertiti per eseguire in modo efficiente il recupero lessicale. Per ogni parola indicizzata, un indice invertito memorizza i documenti nei quali essa compare. Quando viene inviata una query, il motore di ricerca può quindi esaminare soltanto i documenti associati ai suoi termini, invece di confrontarla con ogni dataset del repository.

Il recupero lessicale è efficiente e i suoi risultati sono relativamente facili da interpretare, perché il ranking dipende dalle parole corrispondenti tra la query e ogni dataset. Tuttavia, questo approccio richiede che la query e la rappresentazione indicizzata condividano lo stesso vocabolario. Per esempio, una domanda che contiene l'espressione "numero di residenti" potrebbe non recuperare un dataset descritto soltanto dalla parola "popolazione". Le differenze dovute a sinonimi, abbreviazioni,

traduzioni o terminologia specifica del dominio possono quindi impedire il recupero di dataset pertinenti e ridurre la recall.

Queste limitazioni riguardano principalmente le query lessicali fisse. Quando un modello linguistico può riformulare iterativamente una query ed esaminare le evidenze recuperate, la ricerca lessicale può diventare parte di un processo di recupero più adattivo, come discusso nella Sezione 2.3.4.

2.3.2 Recupero semantico

Il recupero semantico rappresenta le query e le rappresentazioni indicizzate dei dataset come vettori numerici densi, comunemente chiamati embedding. Un encoder produce queste rappresentazioni in modo che testi semanticamente correlati siano collocati vicini nello spazio vettoriale.

Dato l'embedding di una query $\mathbf{q}$ e l'embedding di un dataset $\mathbf{d}$, la loro somiglianza può essere misurata mediante la similarità del coseno:

$$\text{cosine}(\mathbf{q}, \mathbf{d}) = \frac{\mathbf{q} \cdot \mathbf{d}}{\|\mathbf{q}\|\|\mathbf{d}\|}.$$

I dataset candidati vengono recuperati individuando i vettori più vicini al vettore della query. Gli indici approximate nearest-neighbour rendono questa ricerca più efficiente evitando un confronto diretto con ogni vettore della raccolta.

Un vantaggio del recupero semantico è la possibilità di collegare una domanda a un dataset anche quando utilizzano terminologie differenti, compensando alcuni limiti del recupero lessicale. Il ranking è però meno interpretabile e dipende dai dati e dall'obiettivo utilizzati per addestrare il modello di embedding.

La rappresentazione di una tabella rimane fondamentale. Utilizzare soltanto i metadati può omettere informazioni utili sullo schema. Includere i nomi delle colonne può migliorare la copertura strutturale, ma questi nomi possono essere ambigui. Anche i valori della tabella possono fornire informazioni aggiuntive, ma includerne troppi rischia di superare i limiti di input del modello e introdurre rumore. Un sistema di recupero semantico dipende quindi sia dall'encoder selezionato sia dal modo in cui viene rappresentato ogni dataset [8, 1].

2.3.3 Recupero ibrido

Il recupero lessicale e quello semantico hanno punti di forza differenti. Il recupero lessicale è utile quando la query contiene nomi esatti, codici o termini specifici del dominio che compaiono anche nella rappresentazione del dataset. Il recupero semantico, invece, può individuare dataset concettualmente pertinenti anche quando la query e il dataset utilizzano terminologie differenti. Il recupero ibrido combina i risultati dei due approcci per sfruttare queste caratteristiche complementari [8, 1].

Un modo per combinare i due approcci consiste nell'utilizzare i loro punteggi di pertinenza. Poiché il recupero lessicale e quello semantico possono produrre punteggi su scale numeriche differenti, i valori devono prima essere normalizzati. I punteggi normalizzati possono poi essere combinati mediante una somma pesata:

$$s_{\text{hybrid}}(d) = \alpha s_{\text{lexical}}(d) + (1 - \alpha) s_{\text{semantic}}(d),$$

dove α determina il contributo di ciascun metodo al punteggio finale. Un valore più alto di α attribuisce maggiore importanza al recupero lessicale, mentre un valore più basso attribuisce maggiore importanza a quello semantico. Senza normalizzazione, il metodo che produce valori numericamente più grandi potrebbe influenzare maggiormente il ranking finale anche quando i suoi risultati non sono più pertinenti.

Un altro approccio è la Reciprocal Rank Fusion, comunemente abbreviata in RRF. Invece di combinare i punteggi originali di pertinenza, RRF considera la posizione di ogni dataset nei ranking prodotti dai metodi di recupero [2]. Il suo punteggio viene calcolato come segue:

$$\text{RRF}(d) = \sum_{r \in \mathcal{R}} \frac{1}{k + \text{rank}_r(d)},$$

dove $\text{rank}_r(d)$ è la posizione del dataset d nel ranking r, mentre k è una costante che controlla l'influenza dei risultati nelle prime posizioni. Un dataset riceve un punteggio più alto quando compare vicino all'inizio di uno o più ranking. Per esempio, un dataset recuperato in una posizione elevata sia dal metodo lessicale sia da quello semantico riceve supporto da entrambi.

Poiché RRF utilizza le posizioni nei ranking, i punteggi lessicali e semantici non devono essere direttamente confrontabili. I risultati possono però dipendere dal numero di candidati inclusi in ciascun ranking e dal valore scelto per k.

Il recupero ibrido non produce sempre risultati migliori rispetto a entrambi i singoli componenti. Per esempio, uno dei metodi può introdurre dataset irrilevanti che ricevono un punteggio elevato soltanto per una corrispondenza esatta tra parole o per una somiglianza semantica fuorviante. La sua efficacia deve quindi essere valutata sulla stessa raccolta di dataset e sullo stesso compito di Question Answering utilizzati dal sistema completo [8].

2.3.4 Riformulazione agentica della query

Una domanda in linguaggio naturale potrebbe non essere una query di recupero efficace. Può contenere espressioni colloquiali, indicazioni sul formato dell'output, filtri e operazioni che sono importanti per produrre la risposta, ma non riflettono la terminologia utilizzata nei metadati dei dataset.

La riformulazione trasforma la domanda originale in una o più query di ricerca. Può estrarre entità pertinenti e concetti del dominio, rimuovere il linguaggio procedurale, generare sinonimi o produrre query alternative con livelli di precisione differenti.

Un sistema di recupero agentico può eseguire questo processo in modo iterativo. Dopo aver inviato una query iniziale, esamina i metadati dei candidati recuperati e stabilisce se siano sufficienti. In caso contrario, può generare un'altra query utilizzando termini individuati nelle descrizioni o negli schemi dei dataset più promettenti.

Questa interazione segue il principio generale di alternanza tra ragionamento, azioni e osservazioni descritto da ReAct [17]. La strategia di ricerca completa non deve essere stabilita prima della prima interazione con il portale. L'agente può invece adattare le query successive in base ai risultati restituiti dagli strumenti di recupero.

Subramanian et al. applicano questo schema al Question Answering su documenti mediante strumenti per l'esame dei metadati dei file, la ricerca con espressioni regolari e la ricerca di parole chiave specifica per i PDF [16]. L'agente esamina prima i documenti disponibili e poi genera e modifica i comandi di ricerca in base alle evidenze restituite nelle iterazioni precedenti. Il ragionamento semantico non viene svolto dallo strumento di recupero sottostante, che continua a utilizzare corrispondenze lessicali. Viene invece applicato dal modello linguistico quando seleziona le parole chiave, interpreta le corrispondenze, amplia il contesto esaminato e decide se sia necessaria

un'altra ricerca. Un meccanismo di recupero lessicale può quindi rimanere sintattico, mentre la strategia complessiva di recupero è adattiva e guidata dal modello.

La riformulazione iterativa può migliorare la recall, ma introduce anche costi aggiuntivi. Ogni tentativo aumenta la latenza, l'utilizzo del modello e il numero di operazioni esterne. La ricerca può inoltre allontanarsi dal bisogno informativo originale se il modello adotta la terminologia di un risultato irrilevante.

Un processo di recupero agentico dovrebbe quindi definire un numero massimo di tentativi, le condizioni per accettare i candidati recuperati e una regola d'arresto per i casi in cui non sia possibile trovare un dataset soddisfacente.

2.4 Large Language Model e sistemi agentici

I Large Language Model possono interpretare domande in linguaggio naturale, generare output strutturati e produrre codice eseguibile. In un sistema di Question Answering su Open Data, queste capacità possono essere utilizzate per riformulare la domanda dell'utente, selezionare i dataset pertinenti, stabilire come combinarli, generare e correggere il codice e costruire la risposta finale [13].

L'intero processo può essere assegnato a una singola invocazione del modello, ottenendo un'architettura relativamente semplice. Tuttavia, il modello deve gestire diversi compiti nella stessa invocazione: interpretare la domanda, esaminare i metadati recuperati, selezionare i dataset richiesti, determinare le operazioni di integrazione, generare codice eseguibile e rispettare i vincoli relativi all'output e alla sicurezza.

Un'architettura agentica può invece dividere il processo in una sequenza di azioni e osservazioni oppure assegnare singoli compiti a componenti specializzati. Per esempio, un componente può recuperare i dataset candidati, un altro può selezionarli e combinarli e un altro ancora può generare la risposta finale. Questa separazione rende più evidenti le decisioni intermedie e permette di verificarle prima che il sistema proceda al passaggio successivo [17, 13].

2.4.1 Capacità ed efficienza del modello

La scelta del modello linguistico introduce un compromesso tra capacità ed efficienza. I modelli con maggiori capacità di ragionamento e generazione di codice possono gestire con più affidabilità domande complesse che richiedono più passaggi, ma generalmente richiedono più tempo e comportano costi computazionali o economici maggiori. I modelli più piccoli possono essere più veloci e meno costosi, ma possono incontrare maggiori difficoltà quando il compito richiede diverse decisioni collegate.

La sola dimensione del modello non ne determina però le prestazioni. Anche i dati di addestramento, l'architettura, la lunghezza del contesto, la progettazione del prompt, il supporto agli strumenti e la quantizzazione possono influenzare i risultati.

I confronti dovrebbero quindi considerare modelli e configurazioni specifici, senza presumere che una categoria sia sempre preferibile. Per esempio, un modello più piccolo può essere sufficiente per esaminare i metadati, controllare risultati intermedi o formattare l'output finale, mentre un modello più capace può essere utilizzato per la pianificazione o la generazione di codice. L'uso di modelli differenti nelle varie fasi aumenta però la complessità dell'architettura e può rendere più difficile individuare l'origine di un errore.

2.4.2 Tool calling e operazioni deterministiche

Un Large Language Model genera output testuali o strutturati, ma non può esaminare direttamente file, interrogare un database o chiamare un'API esterna. Un sistema agentico mette a disposizione del modello degli strumenti che gli permettono di richiedere queste operazioni attraverso interfacce predefinite.

Ogni strumenti è generalmente descritto da un nome, dal suo scopo e dai parametri accettati. Quando il modello decide di utilizzare un tool, seleziona l'operazione richiesta e genera i relativi argomenti. L'applicazione controlla la richiesta, esegue l'operazione e restituisce il risultato al modello come una nuova osservazione.

Gli strumenti possono essere utilizzati per svolgere operazioni come recupero lessicale o semantico, esame dello schema, esecuzione SQL ed elaborazione di data frame. Queste operazioni sono svolte da software dedicato secondo regole esplicite. Per esempio, invece di chiedere al modello

di calcolare un'aggregazione usando tutti i valori inseriti nel contesto, il sistema può eseguire il calcolo tramite SQL o una libreria di elaborazione dei dati. Questo riduce la quantità di dati fornita al modello ed evita di chiedergli di riprodurre operazioni che il software esistente può svolgere in modo più affidabile [17, 13].

Gli strumenti di ricerca forniscono un esempio concreto di questa separazione. Nell'approccio di Subramanian et al., il modello linguistico seleziona i termini e i comandi di ricerca, mentre gli strumenti eseguono la corrispondenza effettiva [16]. Questo approccio evita di mantenere un indice vettoriale permanente e rende verificabili le singole azioni di recupero. Tuttavia, il comportamento complessivo dipende comunque dalla capacità del modello di formulare query efficaci e interpretare le evidenze restituite.

L'uso di uno strumento strutturato non garantisce però che l'azione selezionata sia corretta. Il modello può scegliere lo strumento sbagliato, generare parametri non validi, ripetere un'operazione costosa o utilizzare un risultato che non corrisponde alla domanda originale. Allo stesso modo, una chiamata dello strumento può rispettare il formato di input previsto e richiedere comunque un'operazione non appropriata o non consentita.

L'applicazione deve quindi validare le chiamate prima di eseguirle. La validazione deve verificare che l'operazione selezionata sia disponibile, che i suoi argomenti rispettino il formato previsto, che il target richiesto sia valido e che il modello disponga dei permessi necessari per accedervi. In questo modo, il modello decide quale azione richiedere, mentre l'applicazione controlla se e come eseguirla.

2.4.3 Agenti specializzati

Un agente è un componente nel quale un modello linguistico seleziona le azioni in base a un obiettivo, alle informazioni disponibili, alle osservazioni ricevute dagli strumenti e ai vincoli imposti dal sistema. Invece di assegnare l'intero processo a un singolo agente, un'architettura può dividerlo tra più componenti con responsabilità specifiche. Quando più componenti utilizzano modelli linguistici per prendere decisioni, possono formare un sistema multi-agente.

In un sistema di Question Answering su Open Data, un agente di recupero può riformulare la domanda dell'utente e individuare i dataset candidati. Un agente di integrazione può esaminare le tabelle recuperate e stabilire se debbano essere unite mediante join o union. Un altro agente

può usare queste informazioni per generare il codice SQL o Python richiesto. Altri compiti, come l'esecuzione di codice validato o la formattazione di un risultato semplice, possono invece essere assegnati a componenti deterministici quando non richiedono il ragionamento di un modello linguistico.

La specializzazione può rendere più semplice e definito il compito assegnato a ogni agente. Ciascun agente può ricevere soltanto il contesto e gli strumenti necessari al proprio ruolo. Per esempio, un agente di recupero necessita dell'accesso ai tool di ricerca dei dataset, ma non deve necessariamente poter eseguire codice. La separazione delle responsabilità rende inoltre più semplici da esaminare le decisioni intermedie, come i dataset selezionati o il piano di integrazione proposto.

La divisione del processo tra più componenti introduce un'ulteriore complessità. Le informazioni possono andare perse o essere interpretate in modo errato durante il passaggio da un componente all'altro. Un errore commesso durante il recupero può influenzare tutte le fasi successive, mentre un piano di integrazione incompleto può produrre codice errato. Ulteriori invocazioni dei modelli aumentano inoltre il tempo di esecuzione e il costo.

Un'architettura multi-agente non è necessariamente migliore di una basata su un singolo agente. La suddivisione del processo tra agenti specializzati può rendere più chiare le responsabilità, consentire verifiche intermedie e limitare i tool accessibili a ciascun agente. Questi vantaggi devono tuttavia compensare la maggiore complessità richiesta per coordinare gli agenti e gestire la comunicazione tra loro.

2.4.4 Revisione, feedback degli errori e intervento umano

Un sistema agentico può utilizzare diversi meccanismi per controllare e correggere i risultati intermedi. Nel pattern generator–reviewer, un componente produce un output candidato, come una selezione di dataset, un piano di integrazione o un frammento di codice, mentre un altro componente valuta se sia coerente con la domanda originale e con i dati disponibili. Un approccio simile è il pattern propose–critique, nel quale una proposta viene modificata in base ai problemi individuati da un altro componente [13].

Un reviewer può verificare se i dataset selezionati rispondano a tutte le parti della domanda, se le chiavi di join proposte siano plausibili e se le operazioni pianificate siano compatibili con gli schemi disponibili. La revisione, tuttavia, non garantisce la correttezza. Componenti basati

su modelli o prompt simili possono ripetere le stesse assunzioni e una proposta plausibile può comunque essere errata. Le proprietà verificabili tramite regole esplicite, come l'esistenza di una colonna, dovrebbero quindi essere controllate mediante una validazione deterministica.

Anche gli errori di esecuzione possono fornire informazioni utili per correggere un'operazione generata. Per esempio, se una query SQL fa riferimento a una colonna mancante o il codice Python incontra un tipo di dati incompatibile, il sistema può restituire una descrizione dell'errore e richiedere un'operazione corretta. Il numero di tentativi deve essere limitato, perché gli errori ripetuti aumentano il tempo di esecuzione e il costo computazionale. Inoltre, un errore causato da un'operazione non corretta può essere corretto, mentre una richiesta non autorizzata deve essere rifiutata e non ripetuta [13].

L'intervento umano rappresenta un'altra possibile forma di controllo. Un utente può confermare i dataset selezionati, risolvere un'interpretazione ambigua o esaminare un piano di integrazione prima dell'esecuzione. Questo può essere utile quando esistono più interpretazioni possibili o quando un'operazione richiede un'ulteriore conferma, ma riduce l'autonomia del sistema e aumenta il tempo necessario per completare il processo.

2.4.5 Requisiti di sicurezza dei sistemi agentici

Collegare un modello linguistico a sistemi di recupero, API esterne, database e strumenti per l'esecuzione di codice introduce rischi di sicurezza, perché gli output del modello possono causare operazioni nell'ambiente circostante. Inoltre, i metadati e i contenuti delle tabelle recuperati da fonti esterne possono influenzare le decisioni del modello. I vincoli di sicurezza devono quindi essere applicati dall'architettura del sistema e non delegati esclusivamente al modello [4, 6].

Nei sistemi di Question Answering su Open Data, i principali rischi riguardano l'accesso non autorizzato a strumenti o dati, l'esecuzione di codice generato non sicuro e la prompt injection indiretta. Quest'ultima può verificarsi quando istruzioni contenute nei metadati di un dataset o nei valori di una tabella vengono interpretate dal modello come comandi invece che come dati esterni [4].

Questi rischi possono essere ridotti trattando i contenuti recuperati come dati non attendibili, limitando ogni componente ai permessi richiesti dal proprio compito, validando le operazioni generate ed eseguendo il codice in un ambiente controllato. Anche l'accesso a tempo, memoria,

rete e filesystem dovrebbe essere limitato. Questi controlli non eliminano ogni possibile errore, ma riducono l'impatto di un comportamento errato o manipolato del modello.

2.5 Riepilogo del capitolo

Questo capitolo ha introdotto i principali concetti per descrivere il Question Answering su raccolte eterogenee di Open Data. I portali Open Data pubblicano dataset e risorse utilizzando modelli di metadati, formati e API differenti. Una raccolta sufficientemente ampia delle loro tabelle può essere considerata un data lake tabellare. Questo contesto facilita l'accesso a dati forniti da soggetti diversi, ma introduce anche differenze negli schemi, nei valori, nei metadati e nelle relazioni tra le tabelle.

Il Tabular Question Answering permette agli utenti di richiedere informazioni senza scrivere direttamente SQL o codice per l'elaborazione dei dati. Una domanda single-table richiede informazioni da una sola tabella, mentre una domanda multi-table richiede la combinazione di più tabelle. In una configurazione known-table, le tabelle necessarie sono già fornite. In una configurazione open-table, invece, il sistema deve prima recuperarle e selezionarle da una raccolta più ampia. Una volta disponibili le tabelle appropriate, il Text-to-SQL o la generazione di codice general-purpose possono essere utilizzati per produrre le operazioni necessarie a rispondere alla domanda.

Il recupero riduce una raccolta ampia a un insieme limitato di dataset candidati. I metodi lessicali confrontano le parole utilizzate nella domanda con quelle contenute nelle rappresentazioni indicizzate dei dataset. I metodi semantici utilizzano embedding per individuare dataset con significati correlati, anche quando viene usata una terminologia differente. I metodi ibridi combinano i risultati dei due approcci. La riformulazione agentica della query può inoltre adattare la ricerca in base ai candidati recuperati nei tentativi precedenti.

Infine, gli agenti basati su LLM possono dividere il processo di risposta in più passaggi o assegnare responsabilità specifiche a componenti differenti. Questo permette di esaminare le decisioni intermedie, correggere alcuni errori e controllare l'utilizzo degli strumenti esterni. Poiché questi sistemi possono accedere a risorse esterne ed eseguire operazioni generate, le loro azioni devono comunque essere limitate e validate.

Questi concetti costituiscono la base per analizzare gli approcci esistenti nel capitolo successivo e per descrivere l'architettura del sistema e le configurazioni sperimentali presentate nelle parti successive della tesi.

3. Stato dell'arte e lavori correlati

Questo capitolo analizza le aree di ricerca maggiormente collegate al sistema e alle domande sperimentali considerate in questa tesi. Il capitolo precedente ha introdotto i concetti necessari per descrivere gli Open Data lake, il recupero dell'informazione, il Tabular Question Answering e i sistemi agentici. Questo capitolo confronta invece approcci rappresentativi, le rispettive configurazioni di valutazione e i loro limiti.

I lavori esaminati affrontano fasi diverse, ma collegate, del processo. Il table retrieval e la dataset discovery riguardano l'individuazione delle fonti pertinenti all'interno di raccolte ampie. La Retrieval-Augmented Generation utilizza le informazioni recuperate per supportare un successivo compito generativo, mentre la scoperta di join e union studia come combinare tabelle differenti. Il Text-to-SQL e il Tabular Question Answering si concentrano invece sulla produzione di operazioni eseguibili o di una risposta a partire da una domanda in linguaggio naturale. I sistemi agentici possono infine integrare più fasi, affidando a componenti distinti la ricerca, l'ispezione delle fonti, l'esecuzione delle operazioni e la revisione dei risultati.

Il confronto valuta quindi quali fasi, dal recupero delle fonti alla generazione della risposta, siano coperte dai benchmark esistenti.

3.1 Table Retrieval e Dataset Discovery

Il table retrieval e la dataset discovery mirano a identificare tabelle utili all'interno di collezioni di dati vaste ed eterogenee. A differenza del recupero tradizionale di documenti, queste attività non possono basarsi soltanto sulla similarità testuale. La rilevanza di una tabella può dipendere anche dal suo schema, dai domini delle colonne, dai valori e dalle relazioni con altre tabelle. Inoltre, molti approcci partono da una tabella o da una colonna esistente, anziché da una domanda in linguaggio naturale. Questa distinzione è importante perché determina quali informazioni siano già disponibili al sistema di retrieval.

Nei portali Open Data, la modalità di accesso più comune rimane invece la ricerca per parole chiave applicata ai metadati. La qualità e la completezza di tali descrizioni possono variare tra dataset

e soggetti che li pubblicano, rendendo difficile individuare una fonte quando la terminologia della domanda non coincide con quella utilizzata nel catalogo. Nargesian et al. illustrano questo limite attraverso uno scenario nel quale la ricerca di progetti di ricerca finanziati con fondi pubblici negli Stati Uniti e in Canada restituisce 20.184 dataset candidati [14]. Il recupero di un insieme ampio di risultati lessicalmente compatibili trasferisce quindi all'utente, o a un componente successivo, il compito di verificarne pertinenza e compatibilità.

I metodi esistenti per la table discovery si concentrano comunemente su relazioni specifiche tra una query table e i candidati disponibili. Tra queste rientrano la joinability, che può aggiungere attributi complementari, e la unionability, che può aggiungere record compatibili [14, 19, 10]. Questi metodi utilizzano diverse combinazioni di sovrapposizione dei valori, domini semantici e contesto relazionale. Le loro differenze tecniche sono discusse nella Sezione 3.3.

Valentine studia il problema correlato dello schema matching e fornisce un framework per confrontare tecniche di matching in presenza di differenti trasformazioni dei dati [11]. I risultati mostrano che l'efficacia di un matcher dipende dalle caratteristiche dei dataset e dal tipo di variazione introdotta tra i loro schemi. Un metodo efficace quando i nomi delle colonne sono simili può non rimanere tale quando cambiano valori, formati o strutture. Lo schema matching non è quindi un semplice passaggio preliminare, ma influenza direttamente le fasi successive. Un'associazione errata tra le colonne può infatti portare a combinare tabelle incompatibili o a selezionare fonti non pertinenti.

Anche le caratteristiche della collezione influenzano le prestazioni riportate dai metodi di discovery. GitTables fornisce un ampio corpus di tabelle relazionali estratte da repository GitHub [7]. Il corpus include il contenuto delle tabelle e informazioni contestuali che possono supportare la ricerca sulla rappresentazione tabellare, l'inferenza dello schema e il retrieval. L'elevato numero di tabelle contenute in GitTables consente di valutare i metodi su una collezione più ampia rispetto ai dataset costruiti manualmente. Tuttavia, le tabelle provenienti da repository software possono differire da quelle dei portali Open Data governativi per metadati, terminologia e modalità di pubblicazione. Di conseguenza, i risultati ottenuti su GitTables potrebbero non essere validi anche per questi portali.

LakeBench affronta la necessità di una valutazione più ampia e sistematica della table discovery [3]. Contiene quattro data lake con oltre 16 milioni di tabelle reali, fino a 100 milioni di colonne e

oltre 10.000 query sintetiche e reali. La costruzione della ground truth ha richiesto più di 7.500 ore di annotazione umana. Il benchmark valuta l'efficacia del retrieval insieme alla latenza delle query, al costo di indicizzazione, al consumo di memoria e alla scalabilità.

LakeBench riporta inoltre che le query reali sono generalmente più difficili di quelle sintetiche, perché i candidati rilevanti sono più sparsi e meno simili alla query table. Circa l'80% delle query reali considerate nelle raccolte WebTable e OpenData presenta non più di dieci tabelle collegabili mediante join o union, mentre le query sintetiche dispongono in media di un numero maggiore di risultati compatibili [3]. Una valutazione basata soltanto su query sintetiche può quindi sovrastimare l'efficacia di un metodo di discovery.

Nel complesso, la letteratura mostra che il table retrieval non può essere ridotto a una singola misura di similarità. Una discovery efficace può richiedere metadati testuali, informazioni sullo schema, sovrapposizione dei valori, domini semantici e relazioni tra colonne. L'importanza di ciascun segnale dipende dalla query e dalle caratteristiche del data lake.

Tuttavia, la maggior parte degli approcci discussi in questa sezione presuppone che il sistema disponga già di una query table o di una colonna. Lo scenario considerato in questa tesi parte invece da una domanda in linguaggio naturale. Il sistema deve prima determinare quali concetti, entità e vincoli debbano essere cercati e soltanto in seguito valutare se le tabelle recuperate possano essere unite tramite join o union. Le tecniche esistenti di table discovery possono supportare questo processo, ma non traducono autonomamente un'esigenza informativa espressa in linguaggio naturale in un insieme sufficiente di fonti.

3.2 Retrieval-Augmented Generation su tabelle

Come discusso nella Sezione 2.3, il retrieval lessicale e quello semantico forniscono segnali complementari, mentre il retrieval ibrido combina i rispettivi ranking o punteggi. BM25, DPR e Reciprocal Rank Fusion costituiscono basi rappresentative per queste strategie [15, 9, 2]. La questione rilevante in questa sezione non riguarda il loro funzionamento generale, ma il modo in cui vengono adattati alle rappresentazioni tabellari e come il ranking risultante influenzi la generazione successiva.

Un retriever tabellare può indicizzare diversi tipi di informazione, tra cui titoli, descrizioni, schemi, righe campionate o tabelle complete. Ogni scelta introduce un diverso compromesso.

I metadati sono economici da indicizzare, ma possono omettere evidenze contenute nei valori. Rappresentazioni più ricche possono migliorare la copertura, ma aumentano i costi di indicizzazione e possono introdurre contenuti irrilevanti nelle fasi successive.

Pneuma applica queste idee direttamente alla table discovery [1]. Rappresenta le tabelle utilizzando sia il loro contenuto sia il contesto esterno, combina ricerca full-text e vettoriale e usa un LLM come giudice per filtrare i candidati. Una descrizione dello schema e delle righe selezionate generata da un LLM fornisce una rappresentazione semantica compatta. Pneuma utilizza inizialmente metodi di retrieval relativamente economici per identificare un insieme ampio di candidati e applica poi la valutazione più costosa basata sul modello soltanto ai risultati nelle posizioni più alte. In questo modo evita di indicizzare ogni cella o di fornire tabelle complete al modello linguistico.

TARGET valuta il table retrieval come parte di compiti generativi, tra cui Question Answering, fact verification e Text-to-SQL [8]. Confronta rappresentazioni sparse e dense costruite a partire da metadati, tabelle complete e singole righe. I risultati mostrano che l'efficacia dipende sia dal compito sia dalle informazioni utilizzate per rappresentare ciascuna tabella. Titoli e descrizioni possono essere determinanti, mentre le rappresentazioni a livello di riga possono migliorare la copertura a un costo maggiore.

TARGET riporta inoltre che le prestazioni a valle diminuiscono quando la tabella corretta compare più tardi nel prompt del modello [8]. Recuperare la tabella corretta non è quindi sufficiente se questa viene posizionata dopo diversi candidati irrilevanti. Aumentare la dimensione del contesto non corregge un ranking inadeguato e può introdurre ulteriore rumore.

Questo confronto evidenzia due decisioni ricorrenti: quali parti di una tabella debbano essere indicizzate e quanti candidati debbano essere conservati. Pneuma affronta la prima attraverso descrizioni compatte generate e righe selezionate, mentre TARGET valuta direttamente differenti granularità di rappresentazione. Entrambi i lavori indicano che conservare più contenuto tabellare o più candidati non è necessariamente vantaggioso, poiché la copertura del retrieval deve essere bilanciata con il costo di indicizzazione e il rumore nelle fasi successive.

3.2.1 Ricerca agentica per parole chiave come alternativa al retrieval vettoriale

Subramanian et al. studiano una diversa progettazione del retrieval confrontando una pipeline RAG convenzionale basata su un database vettoriale con un agente che cerca nei documenti mediante ispezione dei metadati, espressioni regolari e tool specifici per la ricerca di parole chiave nei PDF [16]. Anziché recuperare passaggi attraverso un indice di embedding precalcolato, l'agente genera iterativamente comandi di ricerca, ne esamina i risultati, amplia il contesto attorno alle corrispondenze rilevanti e modifica la propria strategia quando le evidenze disponibili non sono sufficienti.

Sui dataset di Question Answering documentale considerati dagli autori, l'agente raggiunge punteggi medi relativi al baseline RAG pari al 94,52 percento per la faithfulness, all'88,05 percento per il context recall e al 91,48 percento per la correttezza della risposta [16]. Questi risultati non indicano che la ricerca per parole chiave sia sempre preferibile al retrieval semantico: l'agente rimane generalmente al di sotto del baseline e le sue prestazioni relative variano tra i dataset. Mostrano tuttavia che una politica di retrieval adattiva può avvicinarsi alle prestazioni di un sistema vettoriale senza mantenere un database vettoriale permanente.

Il confronto evidenzia un compromesso architetturale. Il retrieval vettoriale trasferisce parte del processo di matching semantico negli embedding e in un indice precalcolato. La ricerca agentica per parole chiave delega invece al modello linguistico la formulazione delle query, il raffinamento della ricerca e le decisioni di arresto, mantenendo tool lessicali deterministici per l'esecuzione. Quest'ultima soluzione può ridurre i requisiti di indicizzazione e manutenzione della base di conoscenza, in particolare quando la collezione di documenti cambia frequentemente. La sua efficacia dipende tuttavia dalla presenza di indizi testuali ricercabili e può diminuire in presenza di query ambigue, documenti di grandi dimensioni, contenuti multimediali o informazioni espresse con una terminologia non prevista dall'agente [16].

L'applicabilità di questi risultati alla discovery di tabelle Open Data rimane limitata. Lo studio valuta il Question Answering documentale anziché il retrieval di tabelle eterogenee e i suoi tool cercano nei contenuti di documenti disponibili localmente. In un portale Open Data, il sistema può inizialmente avere accesso soltanto a metadati incompleti, mentre le evidenze rilevanti

possono trovarsi all'interno di risorse tabellari non ancora scaricate o indicizzate. Nel caso delle tabelle, la rilevanza non dipende soltanto dal contenuto di ciascuna fonte. Una tabella può risultare utile perché presenta uno schema o valori compatibili con quelli di altre tabelle e contribuisce a costruire l'insieme di dati necessario per rispondere alla domanda. Queste proprietà non possono generalmente essere determinate soltanto mediante matching di parole chiave a livello documentale.

Il lavoro rimane comunque rilevante per questa tesi perché evidenzia la necessità di distinguere due fattori spesso confusi: il metodo utilizzato per il retrieval e la strategia con cui il sistema interagisce con esso. Anche a parità di motore lessicale, un agente che formula e riformula iterativamente le query sulla base dei risultati ottenuti può comportarsi diversamente da una pipeline che esegue un'unica query BM25.

3.3 Scoperta di join e union

La scoperta di join e union determina come possano essere combinate informazioni distribuite tra tabelle differenti. La join discovery cerca generalmente colonne i cui valori identificano entità correlate, mentre la union discovery cerca domini di attributi compatibili e un significato relazionale coerente.

Quando due colonne condividono alcuni valori, tali valori possono essere utilizzati per associare le rispettive righe mediante un equi-join. JOSIE ricerca le colonne che presentano la maggiore sovrapposizione con i valori distinti di una colonna usata come query [19]. Il problema viene formulato come una ricerca per similarità tra insiemi e risolto mediante un indice invertito. Un modello di costo stabilisce, durante la ricerca, se sia più conveniente continuare a consultare l'indice oppure verificare direttamente una colonna candidata. In questo modo, JOSIE individua esattamente le prime k colonne per sovrapposizione senza confrontare la colonna di query con tutte quelle presenti nel data lake.

SLOTH affronta il problema correlato di determinare la maggiore sovrapposizione tra due tabelle evitando un confronto esaustivo di ogni coppia di colonne [18]. Entrambi i metodi sono utili quando i valori sono normalizzati e ripetuti tra fonti differenti. La loro efficacia diminuisce quando gli identificatori utilizzano codifiche, convenzioni di denominazione o formati incompatibili. Inoltre,

valori comuni possono comparire in colonne non correlate e non stabiliscono autonomamente la correttezza semantica di un join.

La union discovery richiede una nozione di compatibilità più ampia. Nargesian et al. formalizzano la table union search come il recupero di tabelle in grado di fornire record aggiuntivi a una query table [14]. Il loro metodo combina sovrapposizione dei valori, tipi semantici e rappresentazioni in linguaggio naturale per stimare se due attributi appartengano allo stesso dominio. Determina quindi come debbano essere allineate le colonne della query table e della tabella candidata.

Questo approccio mostra che la unionability non può essere determinata soltanto dai nomi delle colonne o dai valori esatti. Due colonne possono descrivere lo stesso concetto utilizzando etichette o rappresentazioni differenti. Al contrario, etichette simili non garantiscono che le colonne abbiano lo stesso significato. Il metodo può operare su collezioni contenenti oltre un milione di attributi Open Data, ma valuta principalmente la compatibilità di coppie di colonne e non rappresenta esplicitamente le relazioni tra tutte le colonne di una tabella [14].

SANTOS affronta questo limite considerando sia la semantica delle colonne sia le relazioni tra esse [10]. Due tabelle possono contenere domini di attributi compatibili pur rappresentando relazioni differenti. Ad esempio, una città può indicare il luogo di nascita in una tabella e la sede di un'organizzazione in un'altra. Il solo matching dei domini delle colonne potrebbe quindi produrre una union non corretta.

Per distinguere questi casi, SANTOS rappresenta ciascuna tabella come un grafo semantico. Il grafo può essere costruito utilizzando una knowledge base esterna oppure una knowledge base sintetizzata direttamente dal data lake. La seconda opzione è utile quando le risorse esterne non rappresentano adeguatamente la terminologia o le relazioni di una specifica collezione. I risultati mostrano che il contesto relazionale può escludere candidati che appaiono compatibili soltanto quando le loro colonne vengono valutate in modo indipendente [10].

Questi approcci rappresentano differenti livelli di compatibilità tra tabelle. JOSIE e SLOTH utilizzano la sovrapposizione dei valori per identificare possibili relazioni di join. Table Union Search combina segnali basati sui valori, semantici e linguistici per stimare la compatibilità dei domini. SANTOS considera inoltre le relazioni tra le colonne per rappresentare il significato dell'intera tabella. Nessuno di questi livelli è sufficiente in ogni situazione: il segnale appropriato dipende dai valori, dagli schemi e dalla semantica delle fonti confrontate.

LakeBench valuta queste differenze su categorie separate di query di join e union [3]. I risultati mostrano che nessun metodo valutato ottiene prestazioni costantemente migliori in tutte le categorie di query di join. Gli approcci che utilizzano una semantica contestuale o a livello di tabella ottengono generalmente risultati migliori per la union discovery, ma non riescono comunque a recuperare parte dei risultati attesi. Starmie ottiene nel complesso i risultati migliori tra i metodi di union discovery confrontati, ma il relativo Recall@k rimane inferiore all'80% nelle configurazioni considerate [3]. Il benchmark evidenzia quindi un compromesso tra metodi sintattici economici e rappresentazioni contestuali più ricche. Mostra inoltre che l'ordine delle tabelle recuperate cambia in base al criterio utilizzato per valutarne la rilevanza, come la condivisione dei valori, il contenimento, la compatibilità semantica o l'utilità prevista.

Per il Question Answering, la scoperta di una tabella joinable o unionable costituisce soltanto un passaggio intermedio. Il sistema deve anche determinare se la combinazione supporti gli attributi, i filtri e la granularità richiesti. Un join eseguibile può comunque essere semanticamente errato quando codici simili rappresentano entità differenti. Una union può essere strutturalmente valida pur combinando misurazioni espresse con unità o risoluzioni temporali differenti. I segnali di discovery dovrebbero quindi supportare, ma non sostituire, la validazione durante la generazione e l'esecuzione del codice.

Nel workflow OrQA, BLEND viene utilizzato per individuare coppie candidate di tabelle combinabili mediante join o union, successivamente sottoposte a una valutazione basata su LLM [12]. Il Capitolo 4 descrive il ruolo di questa discovery nella costruzione dei casi di test.

3.4 Valutazione del Text-to-SQL e del Tabular Question Answering

Le Sezioni 2.2.4 e 2.2.3 hanno già introdotto la distinzione tra gli scenari known-table e open-table e tra la generazione di query SQL e quella di codice general-purpose. In questa sezione, i lavori correlati vengono confrontati in base alle fasi della pipeline incluse nella valutazione. Alcuni valutano soltanto la generazione della risposta a partire da tabelle già fornite, mentre altri richiedono al sistema anche di individuare e selezionare le fonti necessarie.

DataBench valuta la capacità dei modelli linguistici di rispondere a domande su tabelle già fornite, senza includere la ricerca delle fonti [5]. I risultati variano in base al modello, al tipo di domanda e alla strategia di prompting adottata. OrQA utilizza invece 1.000 domande in linguaggio naturale generate sinteticamente e ancorate a tabelle reali raccolte da portali Open Data del Regno Unito e del Canada. Per ciascuna domanda indica le tabelle necessarie e una query SQL di riferimento, includendo casi single-table e multi-table con differenti livelli di difficoltà [12]. Consente quindi di valutare separatamente se il sistema abbia individuato le tabelle corrette e se le operazioni eseguite abbiano prodotto il risultato atteso.

Questa impostazione introduce tuttavia un limite nella valutazione del retrieval. Più tabelle, o più combinazioni di tabelle, possono contenere le informazioni necessarie per rispondere correttamente alla stessa domanda. Se la ground truth ne annota una sola, un sistema può recuperare fonti alternative valide ed essere comunque penalizzato [12]. La correttezza dell'insieme recuperato non dovrebbe pertanto essere interpretata soltanto come coincidenza esatta con gli identificatori annotati, ma anche in relazione alla sufficienza delle evidenze e al risultato delle operazioni eseguite.

Nel valutare una query SQL, è più utile confrontare il risultato ottenuto con quello atteso che confrontare il testo della query con una soluzione di riferimento. Query scritte in modo diverso possono infatti produrre lo stesso risultato corretto. Tuttavia, ottenere il risultato atteso sui dati disponibili non garantisce che la query sia logicamente corretta: un join o un filtro errato potrebbe produrre casualmente lo stesso output. Per questo motivo, l'execution accuracy dovrebbe essere accompagnata da un'analisi degli errori commessi dal sistema.

LakeGen copre l'intera pipeline operativa: il sistema deve individuare e integrare le fonti necessarie e generare codice Python o SQL eseguibile [13]. Questa impostazione introduce diversi punti di fallimento: un errore nella risposta finale può derivare dal retrieval, dalla selezione delle fonti, dalla loro integrazione o dalla generazione del programma. Per individuarne la causa, la valutazione deve quindi esaminare separatamente le diverse fasi della pipeline, oltre al risultato finale.

3.5 Approcci agentici al Tabular Question Answering

Partendo dai pattern agentici introdotti nella Sezione 2.4, ReAct fornisce un riferimento generale per alternare ragionamento del modello, azioni tramite tool e osservazioni [17]. La sua rilevanza per questo confronto risiede nella possibilità di rivedere le decisioni di retrieval e integrazione utilizzando le evidenze ottenute durante l'esecuzione, anziché assumere tutte le decisioni in un singolo passaggio di generazione.

LakeGen applica un'organizzazione multi-agente alla discovery e al Question Answering su Open Data lake [13]. La pipeline separa retrieval, integrazione e answering. Gli agenti di retrieval cercano in un portale e riordinano i candidati; i componenti di integrazione propongono e validano come debbano essere combinati i dataset; i componenti di answering generano codice Python o SQL eseguibile e formulano la risposta finale. Questa separazione rende osservabili le decisioni intermedie e consente di utilizzare prompt e tool specializzati per responsabilità differenti.

La scomposizione crea tuttavia anche interfacce nelle quali le evidenze possono essere omesse o distorte e richiede chiamate aggiuntive ai modelli. La specializzazione non garantisce quindi, da sola, risposte più accurate.

Nell'ambito di questa tesi, il confronto riguarda la modalità di accesso agli strumenti e le informazioni trasferite al coder. L'accesso agentico permette al modello di guidare la ricerca e l'ispezione, mentre il contesto orchestrato gli fornisce candidati già recuperati. Questo confronto valuta il controllo del recupero all'interno di LakeGen, mantenendo distinta tale scelta dal numero di agenti che compongono la pipeline.

3.6 Benchmark esistenti e pratiche di valutazione

I benchmark esistenti misurano segmenti differenti della pipeline. LakeBench si concentra sulla discovery di tabelle joinable e unionable su larga scala [3]. TARGET collega il ranking delle tabelle a compiti generativi [8]. DataBench valuta il Question Answering su tabelle fornite al sistema [5]. OrQA mette a disposizione domande, fonti Open Data necessarie e query eseguibili per casi single-table e multi-table [12]. Queste risorse sono complementari, ma i loro punteggi non sono intercambiabili perché presuppongono input e criteri di successo differenti.

Anche la scala e la costruzione delle query rendono non direttamente confrontabili i risultati. La valutazione originale di Table Union Search impiega 5.000 tabelle di Open Data e 1.000 query sintetiche ottenute mediante proiezioni e selezioni di tabelle sorgente [14]. LakeBench estende questa impostazione a raccolte più ampie e comprende anche query reali, come discusso nella Sezione 3.1 [3]. La scala della raccolta e la densità dei candidati pertinenti influenzano sia il costo computazionale sia la difficoltà del recupero, limitando la trasferibilità dei risultati ottenuti su collezioni ridotte.

I benchmark di retrieval riportano comunemente Precision@k, Recall@k, Mean Reciprocal Rank, Mean Average Precision o normalized Discounted Cumulative Gain. La recall è particolarmente importante quando una tabella necessaria non recuperata non può essere reintegrata dalle fasi successive. La precision assume maggiore importanza quando i candidati vengono passati a un reranker o inclusi nel contesto del modello. Le metriche sensibili al ranking riflettono il fatto che i primi candidati hanno maggiori probabilità di essere selezionati e risultano più utili nelle fasi successive [8].

I compiti multi-tabella richiedono una valutazione a livello di insieme. Un risultato può contenere alcune, ma non tutte, le fonti necessarie, mentre una bridge table irrilevante se considerata singolarmente può essere essenziale per collegare i dataset. Precision e recall per singola tabella dovrebbero quindi essere accompagnate dalla complete-set recall e da un'analisi di sufficienza e ridondanza. Occorre inoltre considerare le fonti alternative non annotate nella ground truth. La sola accuratezza end-to-end della risposta non consente di distinguere questi casi, ma il solo confronto con l'insieme annotato può a sua volta classificare come errato un percorso alternativo valido.

Deve essere valutata anche l'efficienza. LakeBench misura il tempo di query, il costo di indicizzazione e la memoria, poiché metodi accurati su piccole collezioni possono risultare impraticabili su un data lake [3]. I sistemi agentici aggiungono chiamate ai modelli e ai tool, retry, consumo di token e tempo di esecuzione. La misurazione di questi costi è necessaria per confrontare modelli linguistici differenti, modalità di accesso agli strumenti e livelli di contesto forniti al coder.

Una valutazione realistica dovrebbe riflettere l'eterogeneità e le imperfezioni degli Open Data. I metadati possono essere incompleti, gli schemi incoerenti e i risultati rilevanti sparsi. Le query sintetiche forniscono una ground truth controllata, ma possono sovrastimare il retrieval quando la loro costruzione rende le tabelle rilevanti insolitamente simili alla query [3]. Domande reali e

riferimenti eseguibili riducono questa artificialità, pur richiedendo una validazione più costosa.

3.7 Limiti dei lavori esistenti e research gap

Il confronto precedente evidenzia tre aspetti che richiedono una valutazione congiunta. Il primo riguarda il collegamento tra la ricerca delle fonti e la generazione della risposta. I benchmark di table discovery terminano generalmente con una lista ordinata di tabelle, mentre quelli di Question Answering known-table e di Text-to-SQL assumono che le fonti rilevanti siano già disponibili. Considerati separatamente, questi benchmark non permettono quindi di verificare se, a partire da una domanda in linguaggio naturale, il sistema recuperi tutte le tabelle necessarie per generare una risposta corretta. OrQA e LakeGen collegano queste fasi, fornendo rispettivamente i riferimenti per la valutazione e una pipeline end-to-end [12, 13]. Resta da analizzare, nello stesso ambiente, come le diverse configurazioni influenzino il recupero e la risposta finale.

Il secondo aspetto riguarda il confronto tra le strategie di retrieval e le modalità di accesso agli strumenti. I metodi lessicali, semantici, strutturali e ibridi vengono spesso valutati su collezioni diverse o rispetto a compiti differenti, rendendo difficile un confronto diretto. Occorre inoltre distinguere l'effetto del metodo di recupero da quello del controllo affidato all'agente, che può ispezionare i candidati e riformulare la ricerca. Per misurare l'effetto di ciascuna scelta sono pertanto necessari confronti controllati nello stesso ambiente sperimentale.

I risultati ottenuti nel Question Answering documentale mostrano inoltre che una ricerca agentica per parole chiave può avvicinarsi alle prestazioni della RAG vettoriale [16]. Ciò suggerisce che il metodo di retrieval e la strategia con cui viene utilizzato debbano essere considerati fattori sperimentali distinti. Non è tuttavia ancora chiaro se questi risultati siano validi anche per la ricerca di tabelle eterogenee nei portali Open Data.

Il terzo aspetto riguarda il rapporto tra efficacia ed efficienza. Il modello linguistico, la quantità di contesto fornita al coder e l'uso iterativo dei tool possono influenzare l'accuratezza, la latenza e il costo computazionale. I risultati dei lavori esaminati non permettono di stabilire quale combinazione di questi fattori offra il miglior compromesso nello scenario Open Data considerato dalla tesi.

Il research gap affrontato riguarda quindi il confronto controllato, nello stesso ambiente Open Data, tra strategie di retrieval, accesso agentico e contesto orchestrato, livelli di contesto forniti al

coder e modelli linguistici differenti. La valutazione considera gli effetti sulle singole fasi della pipeline, sul risultato finale e sui costi di esecuzione.

Questa tesi affronta tale lacuna utilizzando OrQA per costruire i casi di test e LakeGen come base della pipeline valutata [12, 13].

3.8 Sintesi del capitolo

I lavori esaminati utilizzano segnali lessicali, semantici e strutturali per recuperare e combinare le tabelle, valutandoli con input e criteri di successo differenti. Il confronto evidenzia la necessità di collegare la qualità delle fonti recuperate alla correttezza della risposta e ai costi dell'elaborazione. OrQA e LakeGen forniscono la base per il confronto sperimentale descritto nei capitoli successivi. Il prossimo capitolo presenta il benchmark e i riferimenti che mette a disposizione per osservare le diverse fasi della pipeline.

4. OrQA: generazione di benchmark basati su Open Data

Gli esperimenti di questa tesi richiedono un benchmark che valuti non solo la capacità di rispondere a una domanda quando la tabella corretta è già disponibile, ma anche quella di individuare i dati necessari. Ogni caso di test deve quindi collegare una domanda in linguaggio naturale alle risorse Open Data utili e a una query eseguibile di riferimento. OrQA crea questi collegamenti generando domande e query SQL a partire da tabelle reali provenienti da portali Open Data pubblici [12].

Questo capitolo descrive OrQA e il processo di generazione del benchmark introdotto nel relativo articolo. In primo luogo vengono individuati i requisiti che motivano il workflow; successivamente ne vengono presentate le fasi di raccolta, discovery, valutazione e generazione. Il capitolo discute inoltre il dataset pubblicato e le principali modifiche introdotte nella pipeline più aggiornata.

4.1 Requisiti del benchmark

Un benchmark Open Data end-to-end deve soddisfare requisiti che non sono presenti nelle tradizionali raccolte TQA a tabella nota e Text-to-SQL. In primo luogo, la domanda deve poter essere risolta mediante risorse che siano anche individuabili attraverso le informazioni esposte da un portale. Fornire in input la tabella corretta eliminerebbe il problema di retrieval che questa tesi intende misurare. In secondo luogo, la ground truth deve identificare sia le risorse rilevanti sia le colonne utilizzate dalla computazione di riferimento. Infine, la correttezza deve poter essere verificata attraverso l'esecuzione, anziché mediante il confronto testuale del codice generato. Programmi equivalenti possono infatti utilizzare operatori differenti e produrre comunque la stessa risposta.

Il carico di lavoro deve riflettere l'eterogeneità degli Open Data. Le tabelle possono presentare schemi incoerenti, titoli ambigui, descrizioni poco informative, metadati multilingue, valori sentinella per i dati mancanti e colonne il cui tipo calcolato non corrisponde al tipo semantico. Il benchmark deve infine includere domande sia su singola tabella sia multi-tabella. Queste ultime

sono particolarmente importanti poiché richiedono di individuare e validare relazioni quali join e union, centrali nell'analisi realistica dei data lake.

Da questi requisiti derivano tre distinti livelli di ground truth. La ground truth di retrieval identifica la tabella o l'insieme di tabelle richiesto dalla domanda. La ground truth di selezione identifica le colonne rilevanti e le relazioni previste tra tali tabelle. La ground truth di esecuzione consiste nel programma di riferimento e nel relativo risultato. Mantenere separati questi livelli consente di localizzare diversi errori. Un sistema può recuperare la tabella corretta ma selezionare colonne errate, oppure selezionare i dati corretti ma generare codice non eseguibile. Un unico punteggio end-to-end sulla risposta nasconderebbe questa distinzione. Nella valutazione di questa tesi, colonne e relazioni non sono misurate con metriche dedicate.

4.2 Workflow OrQA pubblicato

Il sistema OrQA è un workflow agentico per la costruzione di dataset TQA basati su Open Data [12]. È composto da quattro fasi principali: raccolta dei dati, ricerca delle tabelle candidate, valutazione dei candidati e generazione delle domande in linguaggio naturale con le corrispondenti query eseguibili.

4.2.1 Acquisizione degli Open Data

A partire da un endpoint CKAN, OrQA scarica le risorse tabellari insieme ai metadati esposti dal portale. L'articolo descrive un'applicazione ai portali governativi del Regno Unito e del Canada. Le raccolte risultanti contengono 24.404 tabelle britanniche e 31.437 tabelle canadesi, con una notevole variabilità nel numero di righe e colonne [12]. Tale varietà aumenta il realismo del benchmark, ma introduce anche file malformati, colonne sparse, codifiche incoerenti e descrizioni delle risorse di qualità disomogenea.

La fase di acquisizione definisce quindi sia il data lake sia la superficie su cui viene eseguito il retrieval. Il contenuto delle tabelle permette di eseguire le query, mentre titoli, descrizioni, editori, tag e informazioni di schema supportano la successiva discovery. Questa distinzione è importante: durante la ricerca delle tabelle, il sistema deve utilizzare soltanto la domanda in linguaggio naturale

e non può accedere alla query SQL di riferimento, perché quest'ultimo contiene informazioni che faciliterebbero artificialmente il retrieval.

4.2.2 Discovery e valutazione dei candidati

Nel workflow originale, BLEND indicizza il data lake e individua coppie di tabelle che possono essere unite mediante join o union. La ricerca parte dalle colonne considerate idonee e individua altre tabelle contenenti colonne compatibili. Per limitare le corrispondenze casuali, vengono escluse le colonne con meno di dieci valori distinti, appartenenti a tabelle con meno di trenta righe, oppure con più dell'80% di valori mancanti [12].

La discovery dei candidati privilegia intenzionalmente la recall: una sovrapposizione può indicare una possibile operazione, ma non dimostra che due risorse siano correlate in modo significativo. OrQA applica quindi un dibattito basato su LLM per assegnare a ogni coppia un punteggio da zero a dieci usando metadati, informazioni di schema e righe di esempio. Soltanto le coppie che raggiungono una soglia configurabile accedono alla generazione. Nell'esperimento pubblicato la soglia è pari a otto. Su un campione di 100 coppie britanniche valutate manualmente, la differenza media riportata tra i punteggi umani e quelli degli agenti è 0,43, con deviazione standard pari a 1,45 [12]. Il risultato supporta l'impiego del filtro, ma non rende la ground truth generata equivalente a un'annotazione umana esaustiva.

4.2.3 Generazione delle domande e delle query di riferimento

Le coppie accettate alimentano un secondo processo agentico. Un agente coder produce una query eseguibile e ne controlla la sintassi mediante uno strumento di esecuzione; un agente dedicato genera quindi una domanda in linguaggio naturale che esprime l'intento della query. Agenti reviewer esaminano entrambi i prodotti e richiedono correzioni quando i vincoli non sono soddisfatti. La configurazione pubblicata limita l'interazione a tre cicli di revisione e restringe il contesto iniziale a venti colonne, aggiungendo quelle ulteriori quando necessario [12].

Il generatore produce casi a tabella singola e multi-tabella con livelli di difficoltà semplice, moderato e avanzato. La difficoltà è definita attraverso i costrutti SQL attesi: i casi semplici utilizzano principalmente proiezioni e filtri; quelli moderati possono richiedere cast, elaborazione di

stringhe, raggruppamenti o aggregazioni; quelli avanzati possono introdurre sottoquery o funzioni finestra. Le domande generate contengono inoltre termini che rendono individuabili le risorse rilevanti, come parole chiave del dominio o l'organizzazione responsabile della pubblicazione.

4.2.4 Dataset pubblicato

La raccolta pubblicata contiene 1.000 triple domanda–tabella–SQL provenienti dai portali britannico e canadese. Di queste, 716 riguardano una singola tabella e 284 più tabelle. La distribuzione per difficoltà comprende 312 domande semplici, 369 moderate e 319 avanzate [12]. Insieme a ogni domanda e query di riferimento, OrQA registra statistiche di generazione quali il numero di revisioni, il tempo impiegato, i token scambiati, l'output finale del reviewer e gli errori del database in caso di fallimento.

Durante la generazione può essere necessario anche pulire i dati. Nei dataset Open Data, infatti, le colonne numeriche contengono spesso testi usati per indicare valori mancanti. Gli agenti possono quindi convertire i valori nel formato corretto, riconoscere quelli validi e gestire i casi mancanti. La query di riferimento descrive così sia l'analisi da svolgere sia le operazioni di pulizia necessarie per ottenere la risposta.

4.3 La pipeline OrQA aggiornata

L'implementazione attuale mantiene le quattro fasi concettuali descritte nell'articolo, ma le riorganizza in una pipeline configurabile ed estende sia la varietà dei portali sia il processo di validazione. La Figura 4.1 sintetizza la strategia semantica di discovery e le successive fasi di generazione della pipeline aggiornata. La prima fase raccoglie le tabelle, ne rappresenta i metadati mediante embedding e costruisce un grafo delle relazioni tra le risorse candidate. La seconda percorre tale grafo per definire i task analitici, pianificare le query, generare il relativo codice SQL o Pandas e validarne l'esecuzione.

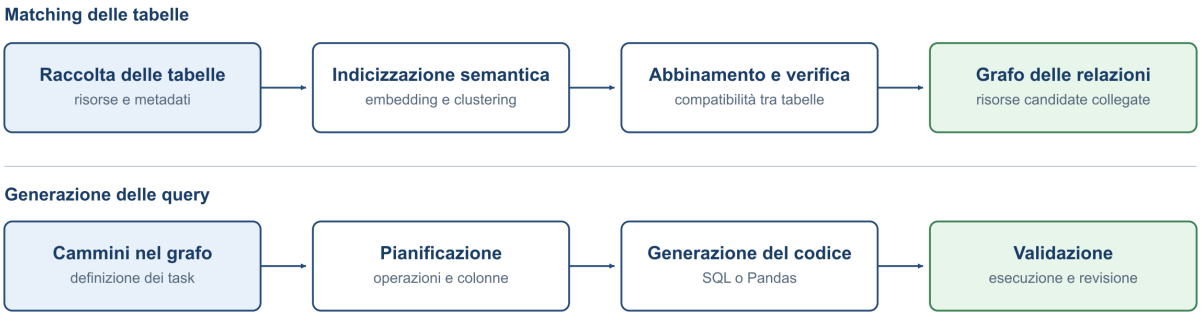


Figura 4.1: Fasi principali della pipeline OrQA aggiornata.

4.3.1 Pipeline multi-portale configurabile

Il workflow originale era organizzato principalmente intorno alle quattro fasi descritte nella Sezione 4.2. L'implementazione aggiornata le suddivide in operazioni più piccole dedicate a crawling, pulizia, normalizzazione dei metadati, indicizzazione, discovery dei candidati, generazione dei candidati per le query, generazione del codice eseguibile e risoluzione del benchmark. Questa suddivisione non modifica l'obiettivo generale di OrQA, ma rende disponibili per un'analisi separata le decisioni e gli artefatti intermedi.

File YAML specifici per ciascun portale definiscono parametri quali limiti di download, valori usati per indicare dati mancanti, soglie di discovery, lingue di destinazione, limiti di esecuzione e seed casuali. Le relative sorgenti utilizzano CKAN, OpenDataSoft o Socrata. Il supporto a queste alternative è rilevante perché la costruzione del benchmark deve gestire differenze nelle API e nelle strutture dei metadati senza considerarle differenze nel compito analitico.

Le configurazioni separate permettono di adattare i parametri alle diverse modalità di accesso, ai metadati e alla qualità dei dati di ciascun portale, senza modificare il workflow generale. I risultati intermedi vengono conservati, così da poter controllare o riprendere l'esecuzione senza ripetere le chiamate precedenti agli LLM. Un seed fisso rende il campionamento più riproducibile, anche se i modelli e i cambiamenti nei portali possono comunque produrre risultati diversi.

4.3.2 Strategie alternative per la discovery dei candidati

La pipeline aggiornata implementa due strategie alternative. La strategia classica combina la ricerca dei candidati basata su BLEND con verifiche eseguite mediante SLOTH e Valentine. La strategia semantica utilizza gli embedding dei metadati per raggruppare le risorse e individuare quelle più simili, mentre Valentine verifica la compatibilità tra gli schemi. Questa strategia supporta anche task che richiedono di collegare due tabelle e calcolare la correlazione tra colonne numeriche.

Entrambe le strategie rappresentano le relazioni tra i candidati mediante un grafo. Attraverso cammini casuali nel grafo, un task può coinvolgere più risorse correlate e non soltanto una coppia di tabelle. La configurazione definisce il numero di dataset iniziali e collegati, la lunghezza massima dei percorsi, le soglie di compatibilità tra gli schemi e il limite di token. Questa estensione affronta due limiti dell'approccio originale: la ricerca basata soltanto sulla sovrapposizione e l'impiego di non più di due tabelle.

4.3.3 Separazione tra piano, codice e validazione

Nell'implementazione attuale, la generazione del codice eseguibile è suddivisa in più fasi: definizione del task, pianificazione della query, generazione del codice, validazione e valutazione finale. Il planner stabilisce le operazioni e le colonne necessarie, mentre un altro componente genera il codice SQL o Pandas. I validator eseguono il codice con limiti di tempo e memoria e segnalano gli eventuali errori da correggere. Infine, gli agenti judge valutano il piano e il codice generato e verificano che la domanda contenga termini utili a individuare le tabelle di origine.

Questa separazione permette di individuare con maggiore precisione la causa di un errore, che può dipendere dalla scelta delle tabelle, dal piano, dalla generazione o dall'esecuzione del codice, oppure da una domanda poco chiara. Infatti, il fatto che il codice sia eseguibile non garantisce da solo la validità del caso generato: la domanda, il piano, le tabelle selezionate, il codice e il risultato devono essere coerenti tra loro.

4.3.4 Sanificazione delle query generate

Prima dell'esecuzione, il codice generato viene sottoposto a controlli specifici per SQL e Pandas. Nel caso di SQL, il validator rifiuta comandi che modificano il database o la configurazione,

accedono a file o risorse esterne oppure caricano estensioni. Vengono inoltre rimossi i comandi `SET`, controllati i join che potrebbero produrre un prodotto cartesiano e normalizzati alcuni elementi della sintassi, come gli identificatori delimitati.

Per il codice Pandas, il controllo analizza la struttura del programma e limita gli import ai moduli previsti per l'elaborazione dei dati. Sono escluse operazioni di lettura e scrittura di file, chiamate dinamiche come `exec` ed `eval` e forme di merge che possono produrre combinazioni cartesiane o risultati eccessivamente grandi. Questi controlli riducono il rischio di eseguire operazioni non previste e intercettano alcuni errori prima della validazione del risultato.

4.3.5 Solver round-trip del benchmark

Tra le aggiunte della pipeline aggiornata rientra il solver round-trip, utilizzato come controllo automatico dei casi generati. A partire dalla sola domanda, un agente indipendente produce le parole chiave e recupera le prime k tabelle tramite un indice BM25, che attribuisce pesi diversi ai vari campi. Successivamente, Valentine verifica le relazioni tra le coppie di tabelle. L'agente seleziona quindi le tabelle e le colonne necessarie, genera il codice e lo esegue. Durante il processo non conosce né le tabelle usate per generare la domanda né il codice di riferimento.

La traccia prodotta permette di verificare il recupero delle tabelle, la selezione delle colonne e la correttezza del risultato. Le tabelle e le colonne individuate dal solver vengono confrontate con quelle associate al caso mediante precision, recall e F1. Quando il codice generato e quello di riferimento sono entrambi eseguibili, i risultati vengono confrontati considerando il tipo, la forma e i valori ottenuti. Per i risultati numerici scalari viene misurata anche l'ampiezza della differenza.

Il solver esegue ogni fase una sola volta, senza tentare automaticamente di correggere gli errori. Il test misura quindi il risultato di una singola esecuzione e fornisce un controllo automatico aggiuntivo sulla qualità dei casi generati, ma non rappresenta una verifica definitiva. Se il solver non riesce a recuperare una sorgente corretta, la causa può essere una domanda formulata male oppure un limite del solver stesso.

4.4 Artefatti generati e supporto alla valutazione

L'output finale di OrQA non si limita a una raccolta di domande in linguaggio naturale. Ogni domanda è associata alle tabelle utilizzate, al codice eseguibile di riferimento e al relativo risultato. Questi elementi permettono di valutare separatamente il recupero delle tabelle, la generazione del codice e la correttezza della risposta.

Il workflow registra inoltre informazioni sul processo di generazione. Nel dataset pubblicato esse comprendono il numero di cicli di revisione, il tempo di generazione, i token scambiati, l'output del reviewer e gli errori del database [12]. La pipeline aggiornata aggiunge piani intermedi, risultati della validazione e tracce prodotte dal solver round-trip. Questi artefatti non modificano la ground truth semantica di una domanda, ma consentono di distinguere gli errori introdotti durante discovery, pianificazione, generazione del codice o esecuzione.

La distinzione è particolarmente importante per i sistemi end-to-end. Una risposta finale errata non rivela se la tabella richiesta fosse assente dai candidati recuperati, se siano state selezionate le colonne sbagliate o se il programma generato abbia applicato un'operazione errata. OrQA espone i riferimenti intermedi necessari per analizzare separatamente questi casi. Il sottoinsieme e le metriche adottati nella tesi sono definiti successivamente come parte della metodologia sperimentale.

4.5 Limiti e minacce alla validità

Il carico di lavoro rimane in parte sintetico perché le domande e i programmi di riferimento sono generati da modelli linguistici, sebbene i dati sottostanti siano reali. La distribuzione linguistica può quindi riflettere i prompt di generazione più delle domande spontanee dei cittadini. Richiedere parole chiave utili al retrieval migliora inoltre la risolvibilità del benchmark, ma può sottorappresentare richieste vaghe o esplorative.

La discovery dei candidati vincola ciò che può essere generato. Un metodo basato sulla sovrapposizione favorisce relazioni visibili attraverso valori condivisi, mentre un metodo basato su embedding dipende dalla qualità dei metadati e dalle rappresentazioni del modello. Le soglie di schema matching e del grafo possono escludere relazioni valide ma difficili. Analogamente, il

solver round-trip non è un oracolo: i suoi fallimenti combinano possibili difetti del benchmark con i limiti di retrieval BM25, selezione delle tabelle e generazione del codice.

Le risorse dei portali sono inoltre mutevoli. I file possono essere aggiornati o rimossi e i metadati possono cambiare dopo la raccolta. La riproduzione di un'esecuzione di OrQA richiede pertanto i dati raccolti, la configurazione del workflow e la corrispondente revisione del codice. Infine, la distribuzione delle domande generate dipende dai portali selezionati, dalle lingue, dalla strategia di discovery e dai prompt di generazione. Un dataset prodotto in un ecosistema Open Data può non rappresentare le domande o i problemi di integrazione presenti in un altro.

4.6 Sintesi del capitolo

OrQA stabilisce un legame tra domande in linguaggio naturale, tabelle Open Data recuperabili e computazioni di riferimento eseguibili. Il sistema costruisce tale legame attraverso crawling, discovery dei candidati orientata alla sovrapposizione, valutazione delle coppie mediante agenti e generazione revisionata di query SQL e domande. Il dataset pubblicato di 1.000 casi proviene dal Regno Unito e dal Canada; gli esperimenti di questa tesi usano invece casi generati per New York City e Regno Unito.

La pipeline aggiornata estende il workflow a ulteriori portali e lingue, supporta discovery classica e semantica, rappresenta le relazioni multi-tabella come un grafo, separa la pianificazione dalla generazione del codice eseguibile e aggiunge un processo indipendente di risoluzione round-trip. Queste estensioni preservano l'obiettivo originale di generare associazioni tra domande, tabelle e programmi, esponendo al tempo stesso maggiori evidenze intermedie per la validazione e l'analisi degli errori. Le modalità con cui tali artefatti vengono selezionati e utilizzati negli esperimenti sono descritte nel capitolo dedicato alla valutazione.

5. Architettura di LakeGen e configurazioni valutate

LakeGen è il sistema utilizzato in questa tesi per confrontare diverse configurazioni di Question Answering su collezioni eterogenee di Open Data tabellari. Le configurazioni considerate differiscono per la strategia di recupero delle tabelle, il modello linguistico utilizzato, la modalità di accesso agli strumenti e le informazioni fornite al coder. A differenza delle configurazioni TQA a tabella nota discusse nel Capitolo 2, il sistema non riceve in ingresso la tabella corretta, ma deve individuare le risorse rilevanti, verificarne il contenuto, generare un programma di analisi ed eseguirlo sui dati. Lo scopo è determinare quali configurazioni offrano il miglior compromesso tra efficacia ed efficienza nello scenario open-table considerato. OrQA fornisce le domande e i riferimenti necessari per condurre gli esperimenti.

Questo capitolo descrive l'architettura di LakeGen e le configurazioni utilizzate negli esperimenti. Per prima cosa viene illustrato il funzionamento del sistema, dalla ricezione della domanda fino alla generazione della risposta finale. Successivamente vengono presentate le strategie di recupero delle tabelle, le modalità di accesso al recupero, il contesto fornito al coder e i modelli linguistici considerati.

5.1 Requisiti di sistema e obiettivi di progettazione

Il primo requisito riguarda la configurazione open-table: il sistema riceve la domanda, ma non conosce in anticipo le tabelle necessarie per rispondere. Le tabelle utili devono quindi essere individuate all'interno del portale selezionato. Inoltre, il recupero dei candidati deve rimanere separato dalla loro successiva selezione. Questa distinzione permette di stabilire se un errore dipenda dal sistema di recupero oppure dall'interpretazione dei candidati da parte dell'LLM.

Un secondo requisito è la confrontabilità delle configurazioni. Le strategie valutate devono ricevere la stessa domanda e utilizzare la stessa versione dei dati e dei metadati. Per questo motivo, ogni esecuzione deve essere registrata insieme alla configurazione utilizzata e ai risultati intermedi, così da permettere la ricostruzione delle diverse fasi.

Infine, il codice generato deve poter utilizzare soltanto le risorse selezionate e gli agenti non devono ricevere informazioni provenienti dalla ground truth. LakeGen controlla quindi i percorsi di caricamento riconosciuti e filtra i dati trasferiti al coder. Il codice viene inoltre verificato prima dell'esecuzione per impedire l'uso di alcune operazioni non consentite.

5.2 Architettura complessiva

La Figura 5.1 presenta una vista d'insieme della pipeline end-to-end. Oltre al percorso principale dalla domanda alla risposta, la figura evidenzia le alternative di recupero, la separazione tra risoluzione e valutazione batch e i due cicli di revisione attivati, rispettivamente, da un errore correggibile nel programma o dall'insufficienza delle tabelle selezionate.

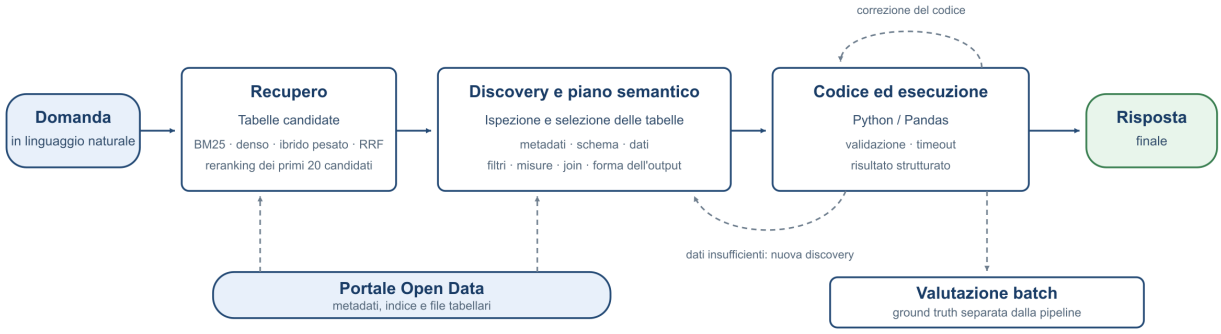


Figura 5.1: Architettura complessiva e flusso di elaborazione di LakeGen.

5.2.1 Flusso di elaborazione

L'elaborazione è suddivisa in cinque fasi. Inizialmente, il sistema interpreta la domanda e individua le parole chiave utili alla ricerca. Il componente di recupero restituisce quindi un insieme ordinato di tabelle candidate. Gli agenti ne analizzano i metadati e il contenuto, selezionano quelle necessarie e definiscono le operazioni da eseguire. Il coder genera poi un programma Python, principalmente basato su Pandas, e lo esegue sui dati selezionati. Nell'ultima fase, il risultato dell'elaborazione viene trasformato in una risposta in linguaggio naturale.

LakeGen può essere utilizzato attraverso un'interfaccia conversazionale realizzata con Chainlit, un'interfaccia a riga di comando e un'API HTTP. L'API permette di elaborare sia una singola

domanda sia un insieme di domande. Le tre modalità condividono gli stessi componenti principali e le stesse configurazioni.

5.2.2 Flusso dei dati, controllo e gestione degli errori

Il sistema di recupero restituisce, per ogni tabella candidata, i relativi metadati, il punteggio di rilevanza e la posizione nell'ordinamento. La fase di discovery collega ogni tabella candidata al corrispondente file locale e ne analizza i metadati e lo schema. Successivamente seleziona le tabelle necessarie, stabilisce il ruolo di ciascuna e, quando sono presenti più sorgenti, indica come devono essere combinate. Il coder riceve infine le informazioni necessarie per generare il programma di analisi.

La ground truth rimane separata dal processo di risoluzione. Nelle esecuzioni batch viene utilizzata soltanto dopo la produzione del risultato, per calcolare le metriche di valutazione. Le tabelle di riferimento, il programma corretto e il risultato atteso non vengono quindi inclusi nel contesto del coder. In caso contrario, la valutazione non rappresenterebbe più una configurazione open-table.

Gli errori sono classificati in base alla fase in cui si verificano. Tra i casi gestiti rientrano errori nei nomi delle colonne, nella sintassi, nei tipi e nei join, file mancanti, superamento del tempo massimo e violazioni dei vincoli di esecuzione. Quando l'errore è correggibile, il messaggio e il programma precedente vengono forniti al tentativo successivo. Se invece il coder rileva che le tabelle non contengono i dati necessari, il sistema può ripetere la fase di discovery.

Gli errori non recuperabili vengono registrati insieme allo stato dell'esecuzione. Nei job batch vengono conservati anche i risultati già prodotti, in modo da poter riconoscere le esecuzioni completate e quelle interrotte.

5.3 Recupero delle tabelle candidate

5.3.1 Elaborazione della domanda e delle parole chiave

La ricerca lessicale utilizza una lista di parole chiave ricavata dalla domanda, mentre la ricerca semantica considera la domanda completa. Se i candidati recuperati non sono adeguati, il sistema

conserva i termini già utilizzati e può formulare una nuova ricerca. Le modalità con cui l'agente accede al recupero sono descritte nella sezione dedicata all'orchestrazione.

5.3.2 Recupero lessicale

Il recupero lessicale è basato su BM25 ed è implementato mediante Apache Solr/Lucene [15]. Il sistema utilizza le parole chiave generate a partire dalla domanda per cercare le tabelle che contengono termini corrispondenti nei metadati. In particolare, la ricerca considera il titolo e la descrizione della tabella, i tag e le informazioni relative alle colonne. I termini di ciascuna ricerca sono combinati con l'operatore AND. I risultati vengono quindi ordinati in base al punteggio di rilevanza assegnato da BM25.

5.3.3 Recupero semantico denso

Il recupero semantico adotta un'architettura dual-encoder ispirata a DPR [9]. La domanda e la rappresentazione della tabella vengono codificate separatamente mediante il modello di embedding pre-addestrato `cohere.embed-v4.0`, accessibile tramite OCI. La rappresentazione della tabella comprende titolo, descrizione, tag, nomi e descrizioni delle colonne. Solr confronta quindi i vettori mediante una ricerca KNN basata sulla similarità coseno.

Il modello non viene ulteriormente addestrato sui dati del benchmark. Il recupero semantico implementato in LakeGen riprende quindi il principio generale di DPR, basato sul confronto tra la rappresentazione della domanda e quella delle tabelle, senza riprodurne l'intero processo di addestramento. Il modello di embedding e la rappresentazione delle tabelle usati in fase di ricerca devono coincidere con quelli utilizzati per costruire l'indice.

5.3.4 Recupero ibrido

LakeGen implementa le due strategie di recupero ibrido descritte nella Sezione 2.3.3. Nella fusione pesata, il recupero lessicale e quello semantico vengono eseguiti separatamente. Dopo aver eliminato le tabelle duplicate, i rispettivi punteggi vengono normalizzati e combinati mediante il parametro α. Valori più elevati di α attribuiscono maggiore importanza al recupero lessicale,

mentre valori più bassi danno più importanza a quello semantico. Se una tabella viene individuata da una sola strategia, il contributo dell'altra viene considerato pari a zero.

In alternativa alla fusione pesata, LakeGen utilizza la Reciprocal Rank Fusion (RRF) [2]. Questo metodo combina le posizioni occupate dalle tabelle negli ordinamenti lessicale e semantico, senza confrontare direttamente i relativi punteggi. Le tabelle collocate nelle prime posizioni di uno o di entrambi gli ordinamenti ricevono quindi un punteggio maggiore. RRF non richiede la normalizzazione dei punteggi e non utilizza il parametro α.

Nelle configurazioni valutate, i risultati di BM25, del recupero denso e delle due forme di recupero ibrido passano attraverso un'ulteriore fase di riordinamento. Il reranker `cohere.rerank-v4.0-fast`, accessibile tramite OCI, riordina i primi 20 candidati; gli altri conservano l'ordine prodotto dal recupero iniziale.

5.3.5 Riformulazione iterativa della query

Quando dispone dei tool di ricerca, l'agente può ispezionare le tabelle candidate e modificare i termini utilizzati se i risultati non contengono le informazioni necessarie. Il numero di tentativi è limitato e le selezioni già rifiutate vengono registrate per evitare di riproporre le stesse tabelle. Le ricerche senza risultati sono conservate in una memoria distinta per configurazione sperimentale. La modalità con contesto orchestrato, nella quale l'agente riceve invece candidati predefiniti, è descritta nella Sezione 5.6.

5.4 Individuazione e selezione dei dataset

5.4.1 Ispezione dei dataset e analisi dei metadati

Il recupero produce un insieme di candidati, ma non stabilisce quali tabelle debbano essere effettivamente utilizzate. L'agente può quindi esaminare lo schema, i tipi, alcune righe di esempio e informazioni sintetiche sulle colonne. Le ispezioni già effettuate vengono conservate, evitando di leggere più volte lo stesso file e rendendo le informazioni disponibili durante la costruzione del piano.

Titolo e descrizione permettono di formulare un'ipotesi, ma la conferma deve riferirsi a colonne presenti. Ciò è importante negli Open Data, dove il nome della risorsa può descrivere una famiglia di dataset senza precisarne periodo, granularità o misura. L'ispezione distingue anche il tipo dichiarato nei metadati dal tipo osservato nel file.

5.4.2 Selezione e combinazione delle tabelle

La selezione restituisce il più piccolo insieme di tabelle ritenuto sufficiente per rispondere. Per ogni requisito vengono indicate la tabella e le colonne che lo soddisfano; viene inoltre specificato il ruolo di ciascuna sorgente. Se un requisito essenziale non è coperto, la selezione non viene considerata completa. Qualora il coder rilevi l'assenza di colonne, periodi, categorie o chiavi necessarie, può richiedere una nuova fase di discovery.

Quando la domanda richiede più tabelle, il sistema deve stabilire come utilizzarle insieme. Le tabelle possono essere collegate attraverso colonne comuni, unite perché rappresentano parti dello stesso insieme di dati oppure analizzate separatamente per confrontarne i risultati. Questa decisione viene inclusa nel piano fornito al coder, che può così applicare l'operazione più adatta alla domanda e alle caratteristiche delle sorgenti selezionate.

5.5 Piano semantico, generazione ed esecuzione del codice

5.5.1 Piano semantico e contesto del coder

Al termine della selezione viene costruito un piano basato sulle informazioni osservate durante l'ispezione. Il piano associa alle colonne selezionate i filtri, le dimensioni, le misure, l'ordinamento, l'eventuale limite e le operazioni di join. Le operazioni supportate comprendono conteggio, conteggio distinto, somma, media, minimo, massimo, rapporto e differenza. Prima di passare al coder, il piano viene controllato per verificare la presenza delle misure, delle colonne di output e del ruolo assegnato a ciascuna tabella.

Nel percorso considerato dalla tesi il piano è prodotto all'interno della discovery e, nelle configurazioni che lo prevedono, trasferito al coder come contratto strutturato. Non costituisce quindi una fase agentica indipendente, ma il risultato della selezione e della verifica delle tabelle.

Il coder riceve la domanda, le tabelle selezionate e, secondo il contesto configurato, i relativi metadati e il piano definito nella fase precedente. Il sistema ricava inoltre dalla domanda la forma attesa dell'output, distinguendo un singolo valore da un risultato tabellare e riconoscendo eventuali richieste di ordinamento o limiti, come nei casi *top N*. Le informazioni fornite derivano dalla domanda e dalle tabelle selezionate, senza utilizzare il codice o il risultato di riferimento del benchmark.

5.5.2 Generazione ed esecuzione del programma Python

Il coder genera un programma Python, generalmente basato su Pandas. I comandi per caricare i file sono predisposti dal sistema e fanno riferimento alle sole tabelle selezionate. Prima dell'esecuzione, i nomi delle colonne presenti nel programma vengono confrontati con quelli reali. Il sistema può correggere differenze puramente formali, mentre segnala i casi ambigui o le colonne mancanti.

La presenza di SQL nel benchmark OrQA non richiede uguaglianza sintattica: il programma di riferimento e quello generato possono essere differenti purché producano risultati semanticamente equivalenti. Il confronto riguarda quindi il risultato strutturato, non il testo del codice.

Per ogni domanda viene creata una directory di lavoro distinta. Il codice viene controllato prima di essere eseguito in un processo separato, al quale è applicato un tempo massimo. Il sistema blocca alcune importazioni e chiamate non consentite e verifica i percorsi letterali usati nelle chiamate di lettura riconosciute rispetto alle tabelle selezionate. Il risultato viene restituito in un formato strutturato, separato dagli eventuali messaggi diagnostici.

5.5.3 Feedback sugli errori e nuovi tentativi

Dopo la prima esecuzione, il coder controlla che il risultato rispetti i filtri, le misure, i raggruppamenti, l'ordinamento e la forma dell'output richiesti. Se il programma non può essere eseguito, il coder riceve il messaggio di errore insieme al codice generato in precedenza. Il messaggio può riguardare, ad esempio, una colonna inesistente, un tipo di dato non valido o un errore nell'unione tra tabelle. Sulla base di queste informazioni può generare una seconda versione del programma. Se invece le tabelle non contengono i dati indispensabili, interrompe l'elaborazione e richiede una nuova selezione delle sorgenti.

5.5.4 Generazione della risposta finale

La fase finale riceve la domanda e il risultato dell'elaborazione e genera una risposta in linguaggio naturale. L'LLM non esegue nuovamente i calcoli, ma si limita a presentare il risultato prodotto dal programma. Il sistema distingue infine le risposte completate dai casi in cui l'elaborazione è stata rifiutata o conclusa con avvisi.

5.6 Configurazioni di interazione e orchestrazione

5.6.1 Accesso agentico e contesto orchestrato

Con l'accesso agentico, l'agente può richiamare direttamente gli strumenti di ricerca e ispezione. Può quindi decidere quando effettuare una ricerca, quali candidati esaminare e se modificare le parole chiave. Il numero e l'ordine delle chiamate possono però variare tra esecuzioni, influenzando il tempo necessario e il numero di token utilizzati.

Nella configurazione con contesto orchestrato, il recupero viene eseguito dall'orchestratore. I candidati vengono inseriti in una struttura che contiene la domanda, la posizione nell'ordinamento, il nome del dataset e i metadati disponibili. I punteggi prodotti dal sistema di recupero non vengono mostrati all'agente, che deve selezionare le tabelle utilizzando soltanto le informazioni ricevute e senza richiamare altri strumenti.

Il confronto riguarda quindi il controllo del recupero: nella prima configurazione è affidato all'agente, mentre nella seconda è gestito dall'orchestratore e l'agente opera su un contesto già predisposto. Per ogni esecuzione vengono registrati la configurazione utilizzata, il numero di invocazioni dell'LLM e la dimensione del contesto.

5.6.2 Intervento umano ed esecuzione autonoma

LakeGen prevede una modalità interattiva e una modalità autonoma. Nella modalità interattiva, utilizzata attraverso l'interfaccia o la CLI, l'utente può controllare le parole chiave generate e le tabelle selezionate prima che il sistema prosegua. Se la proposta non è adeguata, può rifiutarla e

fornire indicazioni per un nuovo tentativo. Questa modalità permette di correggere eventuali errori durante l'esecuzione, ma rende il risultato dipendente dall'intervento dell'utente.

Nella modalità autonoma, invece, il sistema esegue tutte le fasi senza richiedere conferme. Questa modalità viene utilizzata negli esperimenti batch, affinché tutte le configurazioni siano valutate nelle stesse condizioni e i risultati non siano influenzati da decisioni manuali.

5.7 Configurazioni del contesto del coder

Il contesto fornito al coder può assumere tre livelli:

- `minimal` include soltanto i comandi per caricare le tabelle;
- `schema_only` aggiunge i nomi e i tipi delle colonne;
- `full` comprende anche righe di esempio e ulteriori informazioni ottenute durante l'ispezione.

I tre livelli possono essere applicati alla stessa selezione di tabelle. In questo modo, il recupero e la discovery rimangono invariati e il confronto riguarda soltanto il contesto fornito al coder. Un contesto più ricco può ridurre gli errori di interpretazione, ma aumenta la dimensione del prompt. Un contesto minimo riduce invece il numero di informazioni elaborate, ma rende più difficile riconoscere tipi, valori mancanti e significato delle colonne.

5.8 Modelli linguistici

I modelli vengono confrontati mantenendo invariati la domanda, il portale, la strategia di recupero, il contesto e il numero massimo di tentativi. Il confronto considera la capacità di seguire il formato richiesto, utilizzare gli strumenti, associare i requisiti alle colonne e generare codice eseguibile. La qualità dei risultati viene analizzata insieme alla latenza e al numero di token utilizzati.

Per gli esperimenti sono stati selezionati due modelli linguistici accessibili tramite il servizio OCI Generative AI: `openai.gpt-oss-120b` e `meta.llama-3.3-70b-instruct`. Ciascun modello viene utilizzato in tutte le fasi della pipeline che richiedono un LLM, dalla discovery delle tabelle alla generazione del codice e della risposta finale.

La generazione del testo viene eseguita mediante OCI. Gli embedding sono prodotti mediante il provider configurato e indicizzati in Solr. Il modello di embedding e la rappresentazione delle tabelle devono corrispondere a quelli utilizzati durante la costruzione dell'indice.

5.9 Controlli di sicurezza e di esecuzione

La configurazione viene validata prima dell'avvio. Il coder riceve i comandi per caricare le tabelle selezionate e il sistema controlla i percorsi letterali nelle chiamate di lettura riconosciute. Prima dell'esecuzione vengono inoltre bloccati alcuni frammenti di codice, tra cui le importazioni di `os`, `sys` e `shutil`, l'uso di `subprocess` e le funzioni `eval` ed `exec`. Il programma viene eseguito in un processo separato, con un tempo massimo e un limite di memoria configurabile. Un controllo aggiuntivo segnala i join che producono un'espansione anomala del numero di righe.

Questi controlli riducono i rischi associati all'esecuzione di codice generato, ma non costituiscono un isolamento completo del processo: la verifica dei percorsi riguarda soltanto le chiamate di lettura e gli argomenti letterali riconosciuti, mentre il processo non è isolato dal filesystem o dalla rete. LakeGen viene quindi utilizzato in un ambiente locale e controllato, coerentemente con i requisiti discussi nella Sezione 2.4.5.

5.10 Configurazione e riproducibilità degli esperimenti

Le configurazioni sperimentali definiscono il portale Open Data, il modello linguistico, la strategia di recupero e il livello di contesto fornito al coder. Per ogni esecuzione vengono registrati la configurazione utilizzata, la versione del sistema, le tabelle recuperate e selezionate, il codice generato, il risultato, la durata e il numero di token impiegati. Queste informazioni permettono di confrontare le configurazioni e di ricostruire le diverse fasi dell'elaborazione.

L'utilizzo di modelli linguistici introduce una componente di non determinismo; pertanto, esecuzioni della stessa configurazione possono produrre risultati differenti anche quando i parametri rimangono invariati. Questo aspetto viene considerato nel protocollo sperimentale descritto nel Capitolo 6.

5.11 Sintesi del capitolo

LakeGen realizza una pipeline open-table che separa il recupero delle tabelle, la selezione delle sorgenti, la definizione del piano, la generazione e l'esecuzione del codice e la produzione della risposta. L'architettura permette di modificare in modo indipendente la strategia di recupero, l'organizzazione degli agenti, l'accesso agli strumenti, il contesto del coder e il modello linguistico. La registrazione dei passaggi intermedi consente inoltre di individuare la fase in cui si verifica un errore.

Il capitolo successivo usa questi assi per costruire il protocollo sperimentale e misurare efficacia, robustezza ed efficienza.

6. Valutazione sperimentale

Questo capitolo valuta LakeGen sui portali Open Data di New York City (NYC) e del Regno Unito (UK). Confronta le strategie di recupero, i modelli linguistici, le modalità di accesso agli strumenti e il contesto del coder. Misura come queste scelte incidono sulla selezione delle tabelle, sulla correttezza del risultato e sulle risorse impiegate.

Sono presentati il benchmark, le configurazioni e le metriche, seguiti dai risultati del recupero e del sistema. Il capitolo esamina anche tempo e token, domande su più tabelle, variabilità tra esecuzioni e limiti della valutazione.

6.1 Progettazione sperimentale

6.1.1 Benchmark e configurazioni

Gli esperimenti usano il benchmark generato dal processo descritto nel Capitolo 4: 381 domande per NYC e 414 per UK. Sono state considerate valide le domande con testo, codice di riferimento, risultato atteso e tabelle di riferimento presenti nel data lake. Le domande su più tabelle sono 40 per NYC e 101 per UK.

I data lake contengono circa 2.700 tabelle per NYC e 15.000 per UK. La Tabella 6.1 mostra le scelte considerate. Combinandole si ottengono 32 configurazioni. Per ogni domanda, le tre varianti del coder usano le stesse tabelle selezionate. Il recupero e l'analisi dei candidati sono quindi eseguiti una sola volta.

Tabella 6.1: Fattori del disegno sperimentale. Le varianti del coder condividono la selezione delle tabelle all'interno di ciascuna prova.

Fattore	Livelli
Portale	NYC; UK
Recupero	BM25; denso; ibrido con fusione pesata ($\alpha = 0{,}25$); RRF
Modello	GPT-OSS 120B; Llama 3.3 70B, entrambi tramite OCI
Interazione	Agentica; orchestrata
Contesto del coder	`full`; `schema_only`; `minimal`

Il costo computazionale ha motivato una valutazione in due stadi: una prova su una domanda richiede in media da tre a sei minuti e oltre 100 mila token, includendo le tre varianti del coder. Nel primo stadio, tutte le 32 configurazioni sono state valutate sulle stesse prime 100 domande per portale, per un totale di 3.200 prove. Il sottoinsieme comprende 80 domande su una tabella e 20 su più tabelle, con difficoltà distribuite proporzionalmente al rispettivo benchmark. La quota fissa di domande multi-tabella ne facilita l'analisi e il confronto tra portali, ma non rispecchia la loro frequenza nei due benchmark completi. Nel secondo stadio, soltanto le configurazioni agentiche sono state valutate su un insieme più ampio di domande per ciascun portale. Le prime 100 domande sono incluse anche in questo insieme, che non costituisce quindi una validazione indipendente.

6.1.2 Metriche di valutazione

La selezione esatta (*exact selection*) richiede che il sistema scelga tutte le tabelle di riferimento e nessun'altra. La *selection hit* richiede almeno una tabella di riferimento; la *selection recall* ne misura la quota selezionata. Queste metriche valutano il selettore. Recall@10, Hit@5 e Mean Reciprocal Rank (MRR) valutano invece l'ordinamento dei candidati e sono riportate nel riepilogo aggregato.

La correttezza del risultato è valutata con tre criteri. L'*exact result match*, detto in seguito exact match, richiede una corrispondenza rigorosa con il riferimento. Il *lenient result match* ammette nomi di colonna, maiuscole e percentuali diversi, oltre a colonne aggiuntive, ma richiede le stesse righe. La misura *supported correct* considera un risultato corretto o equivalente e sostenuto dai

dati, secondo la valutazione automatica di un modello linguistico. Infine, *execution success* indica che il programma termina senza errori, anche se il risultato può essere sbagliato.

Se il sistema si arresta prima di generare il codice, la domanda è considerata un insuccesso nelle metriche di esecuzione e di correttezza del risultato. I confronti principali usano il contesto `full`. Le tre varianti del coder partono dalle stesse tabelle selezionate: per ogni domanda e configurazione, le metriche di selezione sono quindi calcolate una sola volta. La valutazione considera il risultato del programma, non la risposta finale in linguaggio naturale.

Per ogni domanda e configurazione, tempo e token misurano l'intera prova, che comprende le tre varianti del coder. Le medie per ciascun contesto riguardano invece il solo coder e le domande in cui è stato eseguito. I token sono usati come indicatore delle risorse impiegate.

6.1.3 Criteri di confronto

L'exact match è la misura principale del risultato finale. La selezione delle tabelle, il tempo e i token aiutano a interpretarlo, ma non sostituiscono la correttezza della risposta. I confronti tra alternative sono svolti separatamente per NYC e UK sulle stesse 100 domande.

Per ogni domanda si confrontano le alternative nelle diverse configurazioni e si calcola una differenza media. L'effetto del fattore è la media di queste differenze tra le domande. Gli intervalli di confidenza al 95% sono stimati con 2.000 ricampionamenti delle domande (bootstrap), tenendo insieme i risultati della stessa domanda. Una differenza è considerata significativa se l'intervallo non comprende lo zero. Poiché i confronti sono numerosi e gli intervalli descrivono la variabilità tra domande, ma non quella tra esecuzioni, i vantaggi piccoli vanno letti con cautela.

6.2 Risultati complessivi

Le Tabelle 6.2 e 6.3 riportano le 32 configurazioni con coder `full`. La colonna *Accesso* distingue la modalità agentica (A), in cui il modello usa gli strumenti di ricerca, da quella orchestrata (O), in cui riceve candidati preparati dal sistema. Il massimo exact match osservato è del 35% su NYC, con GPT-OSS, recupero ibrido e modalità orchestrata. Su UK è del 25%, raggiunto da Llama orchestrato con recupero denso, ibrido e RRF. Questi massimi non stabiliscono quale configurazione

sia migliore: su NYC, per esempio, il 35% del recupero ibrido e il 33% di quello denso con GPT-OSS orchestrato differiscono di soli due punti.

Tabella 6.2: Risultati su NYC con coder `full`. Selezione esatta (Sel.), exact match (EM), lenient match (LM) e supported correct (SC) in percentuale. GPT indica GPT-OSS 120B; A e O indicano modalità agentica e orchestrata. Tempo e token si riferiscono alla prova con tutte e tre le varianti del coder; token in migliaia.

Recupero	Modello	Modo	Sel.	EM	LM	SC	Sec.	kToken
BM25	GPT	A	40	16	23	16	220	143,6
BM25	GPT	O	51	29	36	29	108	71,6
BM25	Llama	A	37	14	20	15	273	142,1
BM25	Llama	O	45	20	29	21	183	109,1
Denso	GPT	A	60	27	37	29	162	100,4
Denso	GPT	O	70	33	42	33	141	85,1
Denso	Llama	A	25	21	26	21	340	195,5
Denso	Llama	O	57	23	31	23	225	138,5
Ibrido	GPT	A	67	24	36	25	175	102,9
Ibrido	GPT	O	73	35	44	35	143	85,7
Ibrido	Llama	A	21	16	24	17	328	171,9
Ibrido	Llama	O	63	28	37	28	238	140,4
RRF	GPT	A	56	25	36	26	184	113,3
RRF	GPT	O	74	31	46	31	139	88,3
RRF	Llama	A	22	18	29	18	335	170,8
RRF	Llama	O	63	27	36	27	253	152,7

Tabella 6.3: Risultati su UK con coder `full`. Selezione esatta (Sel.), exact match (EM), lenient match (LM) e supported correct (SC) in percentuale. GPT indica GPT-OSS 120B; A e O indicano modalità agentica e orchestrata. Tempo e token si riferiscono alla prova con tutte e tre le varianti del coder; token in migliaia.

Recupero	Modello	Modo	Sel.	EM	LM	SC	Sec.	kToken
BM25	GPT	A	46	20	35	26	217	134,5
BM25	GPT	O	32	11	25	15	96	64,3
BM25	Llama	A	38	19	26	21	277	121,6
BM25	Llama	O	22	12	19	12	190	93,9
Denso	GPT	A	35	17	30	19	210	127,9
Denso	GPT	O	50	19	36	22	137	78,8
Denso	Llama	A	14	13	21	14	340	155,5
Denso	Llama	O	42	25	30	25	262	136,4
Ibrido	GPT	A	40	17	30	19	212	126,8
Ibrido	GPT	O	56	21	37	28	148	86,0
Ibrido	Llama	A	13	17	22	18	338	151,1
Ibrido	Llama	O	46	25	31	25	323	162,7
RRF	GPT	A	41	18	33	20	208	124,0
RRF	GPT	O	53	23	40	29	143	89,4
RRF	Llama	A	17	13	21	14	347	157,7
RRF	Llama	O	45	25	32	25	250	126,4

Selezione delle tabelle e correttezza del risultato sono distinte. Su NYC, GPT-OSS orchestrato con RRF seleziona esattamente le tabelle nel 74% delle domande, ma raggiunge il 31% di exact match. Su UK, con recupero ibrido, i valori sono 56% e 21%. Anche dopo aver trovato le tabelle, il sistema deve interpretarle e svolgere le operazioni richieste. La differenza tra le due percentuali non misura però direttamente gli errori del coder: anche tabelle alternative possono contenere i dati necessari.

La metrica scelta cambia il quadro. Con GPT-OSS orchestrato e RRF, su NYC l'exact match è del 31% e il lenient match del 46%; su UK sono del 23% e del 40%. Alcuni risultati soddisfano

quindi il criterio più tollerante, ma non quello esatto.

La Figura 6.1 riassume gli effetti dei fattori sulla selezione e sull'exact match. La Tabella 6.4 riporta le differenze e i relativi intervalli di confidenza. Per i confronti presenti in entrambe, figura e tabella usano gli stessi intervalli.

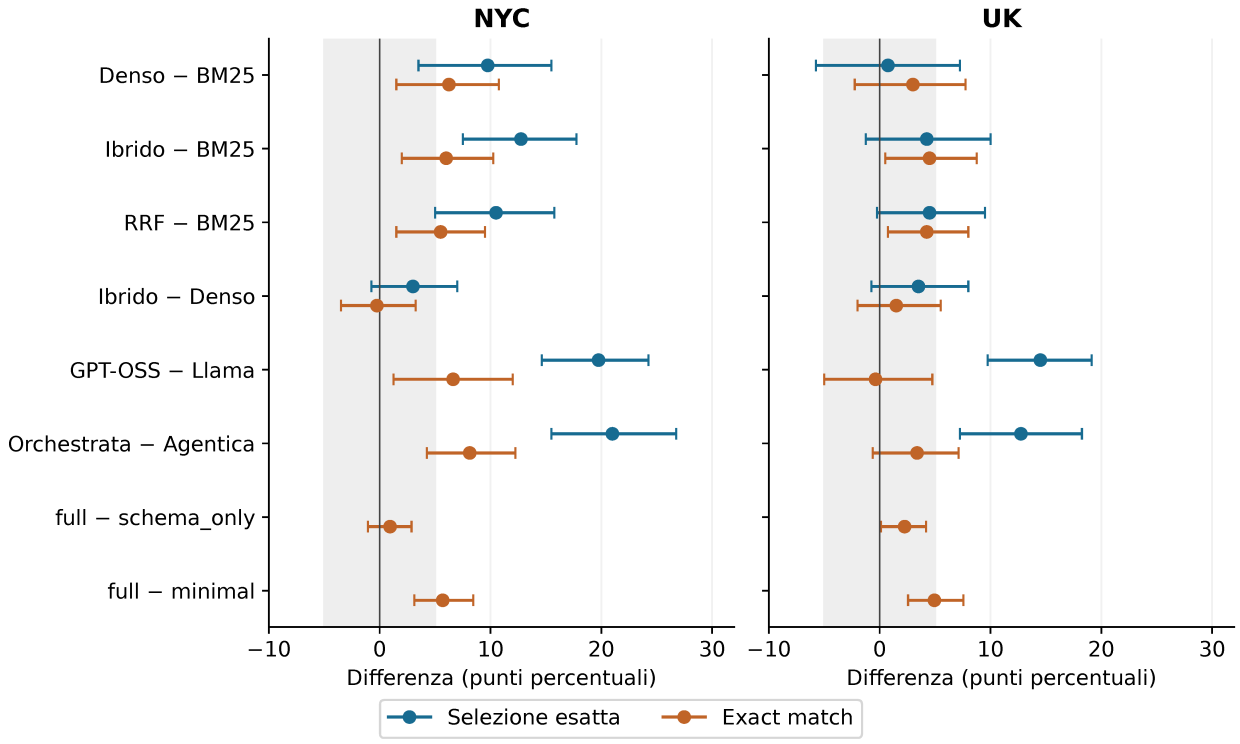


Figura 6.1: Differenze medie tra risultati ottenuti sulla stessa domanda e intervalli di confidenza bootstrap al 95%. Un valore positivo favorisce la prima alternativa indicata. Per il contesto è mostrato solo l'exact match, poiché la selezione delle tabelle è condivisa. La fascia grigia di ±5 punti indica la variabilità osservata nelle ripetizioni.

6.3 Effetto della strategia di recupero

6.3.1 Correttezza del risultato

Su NYC, l'exact match medio è del 19,75% con BM25 e di circa il 26% con ciascuna delle altre tre strategie. Il vantaggio di denso, ibrido e RRF su BM25 è di circa sei punti e i relativi intervalli non comprendono lo zero. Tra queste tre strategie, invece, non emerge una differenza chiara.

Tabella 6.4: Differenze medie nell'exact match e intervalli di confidenza al 95%. Valori in punti percentuali. Gli effetti piccoli richiedono cautela alla luce delle ripetizioni.

Portale	Confronto	Differenza	IC 95%
NYC	Agentica − Orchestrata	-8,12	[-12,25; -4,25]
NYC	GPT-OSS − Llama	6,62	[1,25; 12,00]
NYC	full − schema_only	0,94	[-1,06; 2,88]
NYC	full − minimal	5,69	[3,12; 8,44]
UK	Agentica − Orchestrata	-3,38	[-7,12; 0,62]
UK	GPT-OSS − Llama	-0,38	[-5,00; 4,75]
UK	full − schema_only	2,25	[0,12; 4,19]
UK	full − minimal	4,94	[2,56; 7,56]

Su UK, l'exact match medio va dal 15,50% di BM25 al 20,00% dell'ibrido. Ibrido e RRF superano BM25 di poco più di quattro punti, ma il vantaggio è vicino alle oscillazioni osservate tra esecuzioni. La media risente inoltre dei risultati di BM25 orchestrato, discussi più avanti. Tra denso, ibrido e RRF non emerge una differenza chiara.

6.3.2 Recupero e selezione delle tabelle

La Tabella 6.5 mostra la qualità dell'ordinamento, mediata sui due modelli. Le configurazioni possono formulare ricerche diverse: il confronto riguarda quindi l'intero processo, non solo gli algoritmi applicati alla stessa ricerca.

Tabella 6.5: Metriche di ranking: media dei due modelli, calcolata dai valori arrotondati di `summary.md`. Recall@10, Hit@5 e MRR sono espressi in percentuale. A: agentica; O: orchestrata. Non sono disponibili IC per queste misure.

Portale	Recupero	Modo	Recall@10	Hit@5	MRR
NYC	BM25	A	59,4	64,0	57,5
NYC	BM25	O	59,1	60,5	55,5
NYC	Denso	A	95,5	96,0	81,8
NYC	Denso	O	94,5	95,0	80,8
NYC	Ibrido	A	98,5	99,0	84,3
NYC	Ibrido	O	97,5	98,0	83,0
NYC	RRF	A	98,5	99,0	83,9
NYC	RRF	O	96,5	96,5	82,5
UK	BM25	A	70,2	76,0	59,7
UK	BM25	O	40,8	42,5	34,2
UK	Denso	A	82,2	80,0	57,9
UK	Denso	O	82,2	80,0	57,9
UK	Ibrido	A	88,2	84,0	61,6
UK	Ibrido	O	87,8	83,5	60,8
UK	RRF	A	87,3	84,5	62,7
UK	RRF	O	83,8	82,0	61,1

Anche il recupero e la selezione favoriscono le strategie semantiche su NYC. Con GPT-OSS agentico, Recall@10 è del 58,7% con BM25, contro il 95,5% con denso e il 98,5% con ibrido e RRF. Nella selezione esatta, mediata su modello e modalità, BM25 raggiunge il 43,25%, contro valori dal 53% al 56% delle altre strategie. Le differenze sono riportate nella Figura 6.1.

Su UK, la selezione esatta media varia poco: 34,50% con BM25, 35,25% con denso, 38,75% con ibrido e 39,00% con RRF. Il migliore ordinamento osservato non produce quindi una differenza significativa nella selezione.

6.4 Modello linguistico e modalità di interazione

6.4.1 Confronto tra GPT-OSS e Llama

Nell'exact match GPT-OSS supera Llama di circa 6,6 punti su NYC. Su UK i due modelli ottengono risultati simili. Il vantaggio di GPT-OSS è più marcato nella selezione esatta: circa 19,8 punti su NYC e 14,5 su UK. Le differenze di exact match e i relativi intervalli sono riportati nella Tabella 6.4.

Su UK, GPT-OSS supera Llama di 8,00 punti nel lenient match [3,00; 13,375], mentre la differenza nel supported correct è di 3,00 punti [-1,50; 7,75] e non è significativa. Il vantaggio nella selezione non garantisce quindi un risultato più corretto. Poiché il modello è usato in più fasi, non si può attribuire la differenza al solo selettore o al solo coder.

6.4.2 Accesso agentico e contesto orchestrato

La modalità orchestrata supera quella agentica nell'exact match di circa 8,1 punti su NYC e 3,4 su UK. Solo su NYC l'intervallo della differenza esclude lo zero. La selezione esatta migliora invece di 21 punti su NYC e 12,8 su UK. Su UK, dunque, la migliore selezione non produce un vantaggio altrettanto chiaro nel risultato finale.

La Figura 6.2 distingue i risultati per modello e strategia di recupero. Su NYC, passando dalla modalità agentica a quella orchestrata, la selezione di Llama sale dal 21% al 63% con recupero ibrido, dal 25% al 57% con denso e dal 22% al 63% con RRF. Per GPT-OSS gli aumenti sono più contenuti. Llama sembra quindi trarre maggiore beneficio dai candidati preparati dal sistema.

BM25 su UK costituisce un'eccezione. Con GPT-OSS, Recall@10 scende dal 68,7% in modalità agentica al 40,5% in quella orchestrata; con Llama, dal 71,7% al 41,1%. Anche la selezione esatta diminuisce: dal 46% al 32% con GPT-OSS e dal 38% al 22% con Llama. La ricerca lessicale con condizioni molto restrittive può non trovare candidati. Nella modalità orchestrata il modello non può riformulare la ricerca, e questo può spiegare il peggioramento osservato. Per le misure dell'ordinamento, prive di intervalli di confidenza, il confronto resta descrittivo.

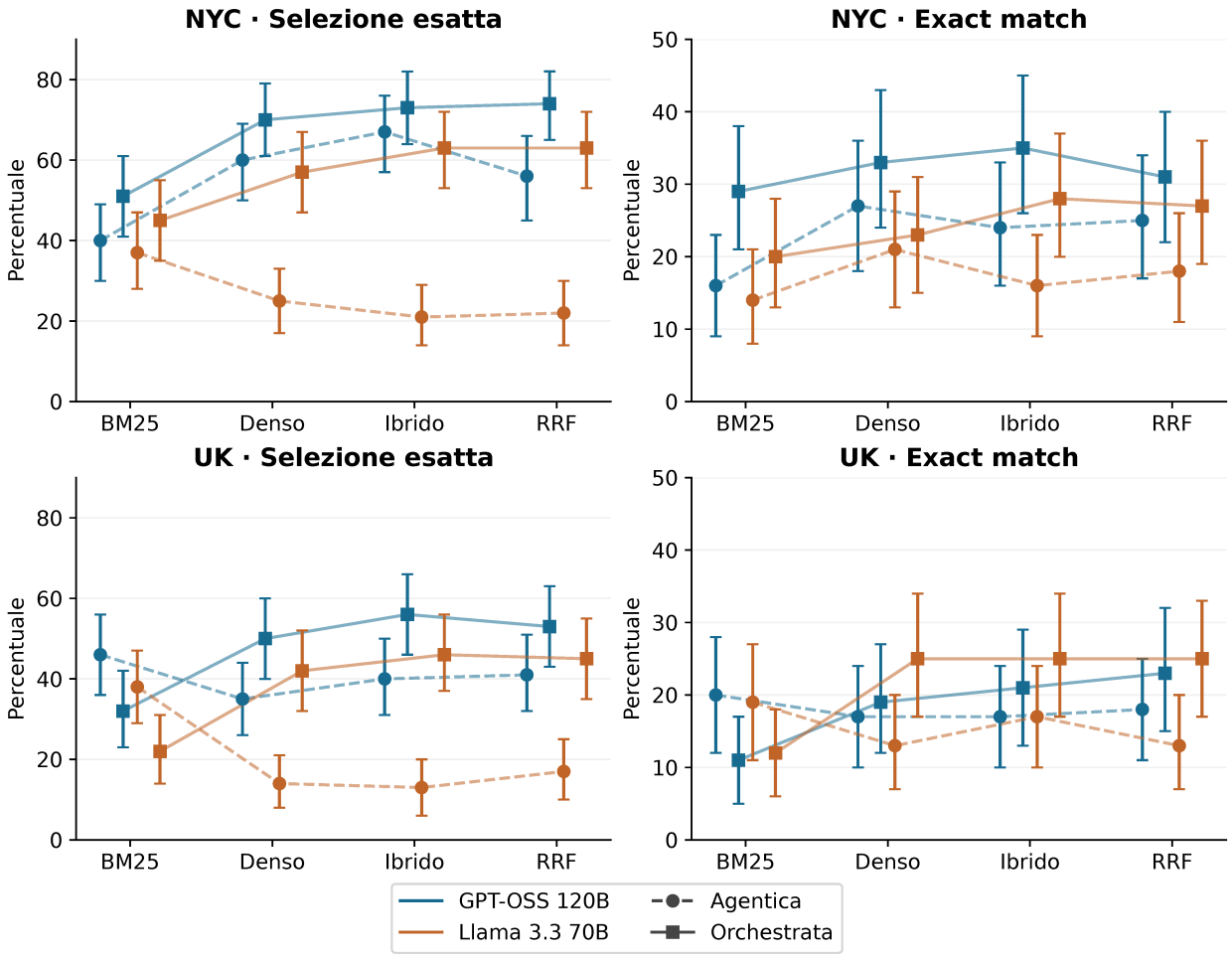


Figura 6.2: Selezione esatta ed exact match per portale, modello, modalità e strategia di recupero, con coder `full`. I punti indicano le medie su 100 domande e le barre gli intervalli di confidenza bootstrap al 95%. Le linee aiutano a leggere il grafico. Per confrontare le modalità si calcolano le differenze sulla stessa domanda, con lo stesso modello e la stessa strategia di recupero, e si considerano i relativi intervalli di confidenza.

6.5 Effetto del contesto del coder

I tre contesti del coder usano le stesse tabelle selezionate. `full` include schema e righe di esempio; `schema_only` fornisce lo schema; `minimal` contiene solo il comando di caricamento. Il confronto misura quindi l'effetto delle informazioni fornite al coder.

La Figura 6.3 mostra l'exact match per modello e contesto, mediato su recupero e modalità di accesso. Rispetto a `schema_only`, `full` guadagna circa un punto su NYC e 2,3 su UK. Il vantaggio su UK è piccolo rispetto alla variabilità tra esecuzioni: non basta a stabilire che il contesto completo sia sempre preferibile.

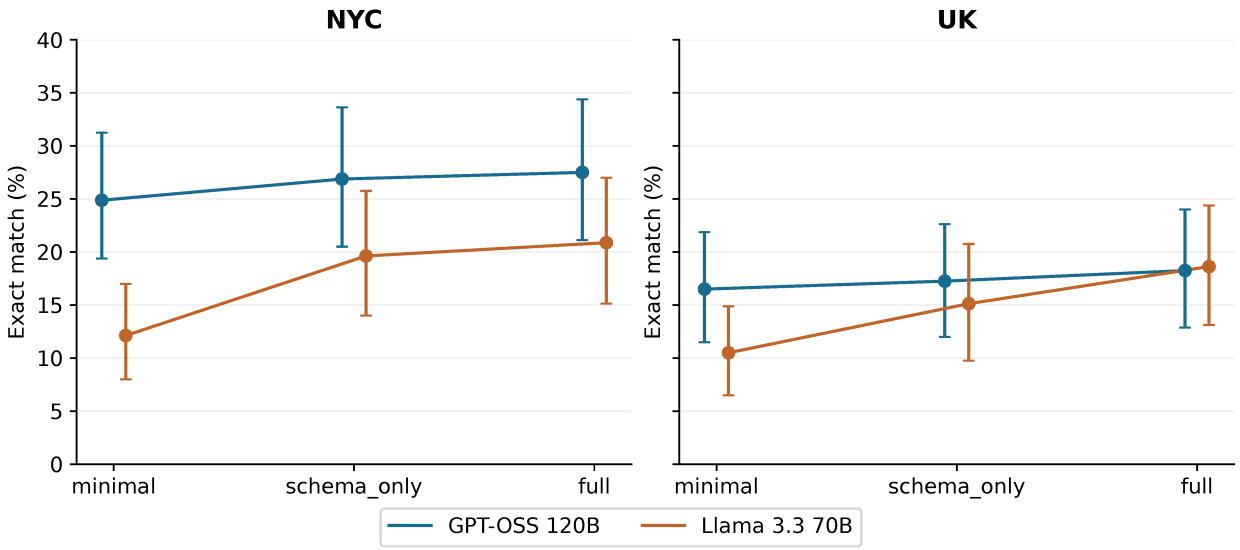


Figura 6.3: Exact match per contesto del coder e modello, mediato sulle strategie di recupero e sulle modalità di accesso. Le barre indicano gli intervalli di confidenza bootstrap al 95% sulle domande. Le tre varianti condividono la selezione delle tabelle.

La distanza tra `full` e `minimal` è maggiore: circa 5,7 punti su NYC e 4,9 su UK. Il calo con il contesto minimo riguarda soprattutto Llama, per cui la differenza supera otto punti in entrambi i portali. Per GPT-OSS è inferiore a tre punti.

Con Llama, su NYC l'exact match è del 12,1% con `minimal`, del 19,6% con `schema_only` e del 20,9% con `full`; su UK i valori sono 10,5%, 15,1% e 18,6%. Gran parte del miglioramento osservato è quindi associata allo schema; le righe di esempio aggiungono un beneficio più piccolo. Il confronto delle risorse impiegate dai singoli contesti è riportato nella Tabella 6.6.

Tabella 6.6: Tempo medio (s) e migliaia di token (kToken) del solo coder, per contesto. Le medie sono ponderate per il numero di domande in cui il coder è stato eseguito in ciascuna configurazione.

			`full`		`schema_only`		`minimal`	
Campione	Portale	Modello	s	kToken	s	kToken	s	kToken
100 dom.	NYC	GPT-OSS	51,6	29,6	43,8	25,4	44,4	25,1
100 dom.	NYC	Llama	101,8	57,5	88,4	45,1	86,9	41,0
100 dom.	UK	GPT-OSS	56,5	31,4	46,1	27,5	48,2	28,4
100 dom.	UK	Llama	107,8	52,3	93,5	40,3	100,4	40,7
Esteso	NYC	GPT-OSS	50,6	31,5	45,4	28,3	43,9	27,0
Esteso	NYC	Llama	101,0	57,4	83,4	44,6	87,8	42,7
Esteso	UK	GPT-OSS	56,8	33,6	48,8	30,2	48,9	32,4
Esteso	UK	Llama	105,4	51,2	91,5	41,7	96,8	43,5

Il contesto `schema_only` richiede meno tempo e token di `full` in tutti i gruppi della tabella. Nel campione agentico esteso, per esempio, con Llama su NYC il tempo medio del coder passa da 101,0 a 83,4 secondi e i token da 57,4 a 44,6 mila. Il contesto `minimal` non offre sempre un ulteriore risparmio rispetto a `schema_only`: su UK, nel campione esteso, usa più token con entrambi i modelli. Il suo minore exact match, discusso sopra, suggerisce quindi `schema_only` come compromesso nei dati osservati. Queste medie escludono le domande fermate prima del coder e non comprendono recupero e selezione; non vanno confrontate direttamente con i tempi e i token dell'intera prova. Per i costi dei singoli contesti non sono disponibili intervalli di confidenza: le differenze sono descrittive.

6.6 Risultati sul benchmark esteso in modalità agentica

Il benchmark esteso comprende 16 configurazioni agentiche. Le otto configurazioni NYC coprono tutte le 381 domande e le quattro UK con Llama tutte le 414. I confronti appaiati usano solo le domande comuni. Le misure di ranking UK con GPT-OSS riguardano le prime 100 domande.

La Tabella 6.7 riporta i principali confronti di exact match; per recupero e modello usa il coder `full`. Su NYC, BM25 resta inferiore di circa sei punti alle strategie con componente semantica. Tra denso, ibrido e RRF gli intervalli comprendono lo zero. Su UK nessuna strategia di recupero ha un vantaggio significativo nell'exact match. La selezione esatta favorisce invece BM25 rispetto a denso, ibrido e RRF di 16,2, 12,1 e 13,5 punti. Selezionare più spesso le tabelle di riferimento non garantisce quindi un risultato migliore.

Tabella 6.7: Contrasti appaiati di exact match sul benchmark esteso, limitatamente alla modalità agentica. I confronti di recupero e modello usano il coder `full`; gli altri confrontano i contesti del coder. Differenze e IC bootstrap al 95% in punti percentuali; *n* indica le domande del confronto.

Portale	Confronto	*n*	Differenza	IC 95%
NYC	BM25 − denso	381	-6,6	[-9,8; -3,1]
NYC	BM25 − ibrido	381	-6,4	[-9,4; -3,5]
NYC	BM25 − RRF	381	-6,3	[-9,3; -3,0]
NYC	GPT-OSS − Llama	381	6,6	[3,5; 10,0]
NYC	full − schema_only	381	1,4	[0,1; 2,8]
NYC	full − minimal	381	4,0	[2,5; 5,7]
UK	GPT-OSS − Llama	351	4,3	[1,1; 7,4]
UK	full − schema_only	414	2,4	[1,2; 3,8]
UK	full − minimal	414	4,4	[2,9; 5,8]

Nel benchmark esteso, GPT-OSS supera Llama nell'exact match di 6,6 punti su NYC e 4,3 su UK, con intervalli che escludono lo zero in entrambi i casi. Nelle prime 100 domande, invece, su UK l'intervallo comprendeva lo zero. La selezione esatta favorisce GPT-OSS di 34,0 punti su NYC [30,2; 37,8] e di 21,9 su UK [18,6; 25,4]. Il confronto fra modelli su UK usa 351 domande presenti per entrambi.

Nel campione agentico esteso, `full` supera `schema_only` nell'exact match di 1,4 punti su NYC e 2,4 su UK; gli intervalli escludono di poco lo zero. Rispetto a `minimal`, il vantaggio è di 4,0 e 4,4 punti; nel lenient match sale a 6,7 e 6,3. Le righe di esempio sono quindi associate a un beneficio piccolo, mentre il contesto minimo causa una perdita maggiore. Le ripetizioni disponibili e i molti confronti impediscono però di considerare affidabili vantaggi di uno o due punti.

6.7 Compromesso tra efficacia ed efficienza

La Figura 6.4 confronta exact match con coder `full`, tempo medio e token. Ogni punto è una configurazione. Tempo e token comprendono le tre varianti del coder anche quando il sistema si ferma prima di generare codice. Non misurano quindi il consumo della sola variante `full`.

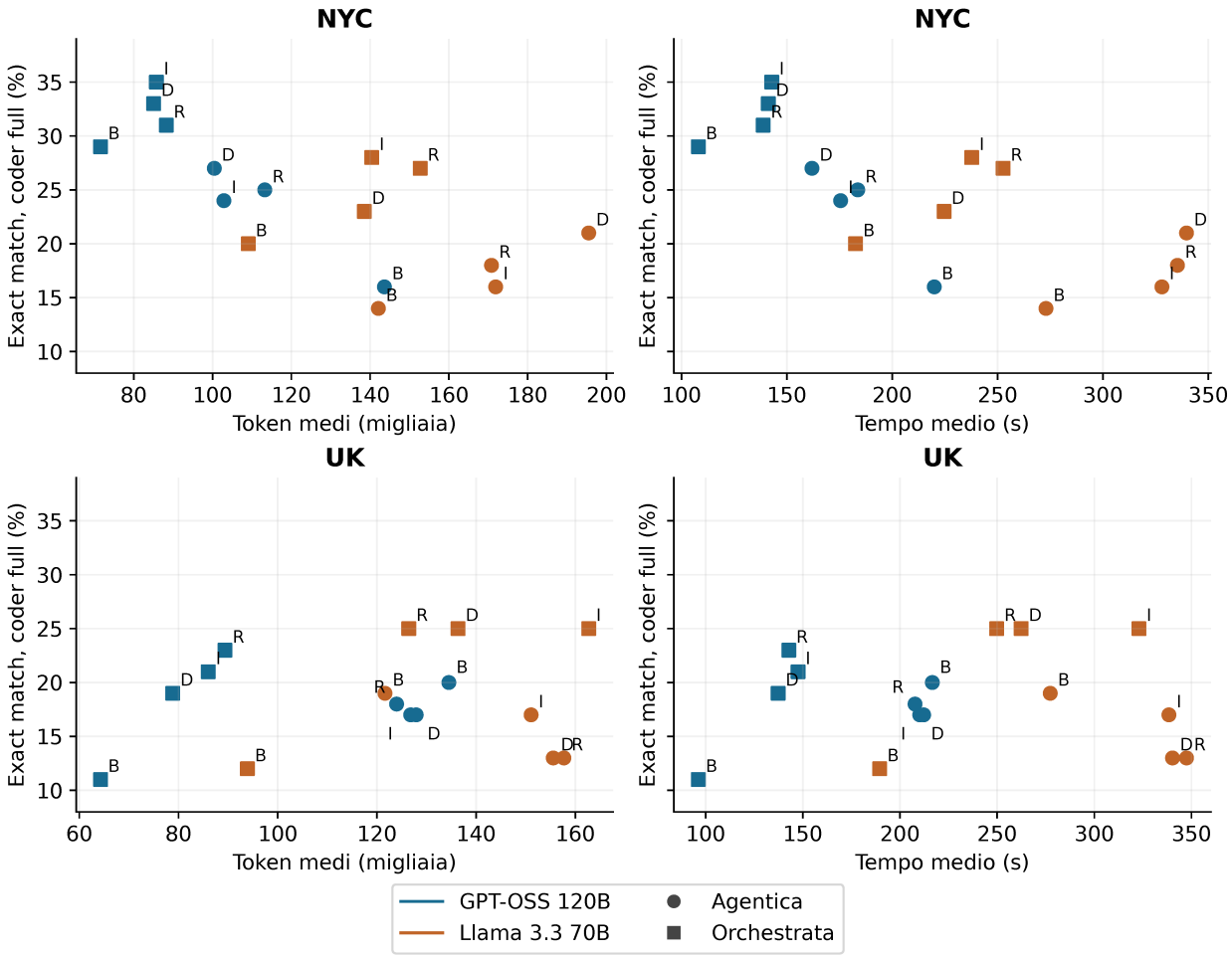


Figura 6.4: Exact match con coder `full`, tempo medio e token della prova con tre contesti. B: BM25; D: denso; I: ibrido; R: RRF. I punti mostrano le medie osservate; il grafico non indica se le differenze tra configurazioni siano statisticamente significative.

Su NYC, passando dalla modalità agentica a quella orchestrata, il tempo medio per domanda scende da 252,1 a 178,5 secondi e i token da circa 142.564 a 108.918. Le riduzioni sono del 29,2% e del 23,6%. Su UK il tempo scende da 268,8 a 193,6 secondi e i token da circa 137.394 a 104.729,

con riduzioni del 28,0% e del 23,8%. Queste medie sono calcolate sui modelli e sulle strategie di recupero considerati.

La Tabella 6.8 riporta gli intervalli di confidenza delle riduzioni percentuali dei token.

Tabella 6.8: Token medi per domanda nell'intera prova e riduzione passando dalla modalità agentica a quella orchestrata. Gli intervalli di confidenza al 95% della riduzione sono ottenuti con 2.000 ricampionamenti appaiati delle 100 domande di ciascun portale.

Portale	Agentica	Orchestrata	Riduzione	IC 95%
NYC	142.564	108.918	23,6%	[16,1%; 30,1%]
UK	137.394	104.729	23,8%	[16,4%; 30,4%]

Il vantaggio non è uguale per tutte le configurazioni. Con GPT-OSS e BM25 su NYC, il tempo scende da circa 220 a 108 secondi e i token da 143.633 a 71.570, quasi dimezzandosi. Con Llama e recupero ibrido su UK, invece, i token aumentano da circa 151.088 a 162.704 e il tempo scende solo da 338 a 323 secondi. Il consumo dipende quindi anche dalle operazioni effettivamente svolte.

Il tempo va letto insieme alla correttezza. Su UK, GPT-OSS con BM25 orchestrato impiega in media circa 96 secondi, ma raggiunge solo l'11% di exact match: un arresto anticipato può ridurre il consumo senza migliorare il risultato. Le piccole differenze di correttezza possono inoltre cambiare tra esecuzioni.

6.8 Domande su una o più tabelle

La Figura 6.5 confronta le domande su una tabella con quelle su più tabelle. Le medie usano le configurazioni con coder `full`; gli intervalli sono calcolati separatamente sulle 80 domande del primo gruppo e sulle 20 del secondo.

Su NYC, per le domande su una e su più tabelle, l'exact match medio è rispettivamente del 27,9% e del 9,4%. La selezione esatta è del 57,2% e del 28,8%; l'esecuzione riesce nell'82,4% e nel 66,3% dei casi. Su UK, negli stessi due gruppi, l'exact match è del 21,1% e del 7,8%, la selezione esatta del 37,7% e del 33,8%, e l'esecuzione riesce nell'81,6% e nel 66,9% dei casi. Le

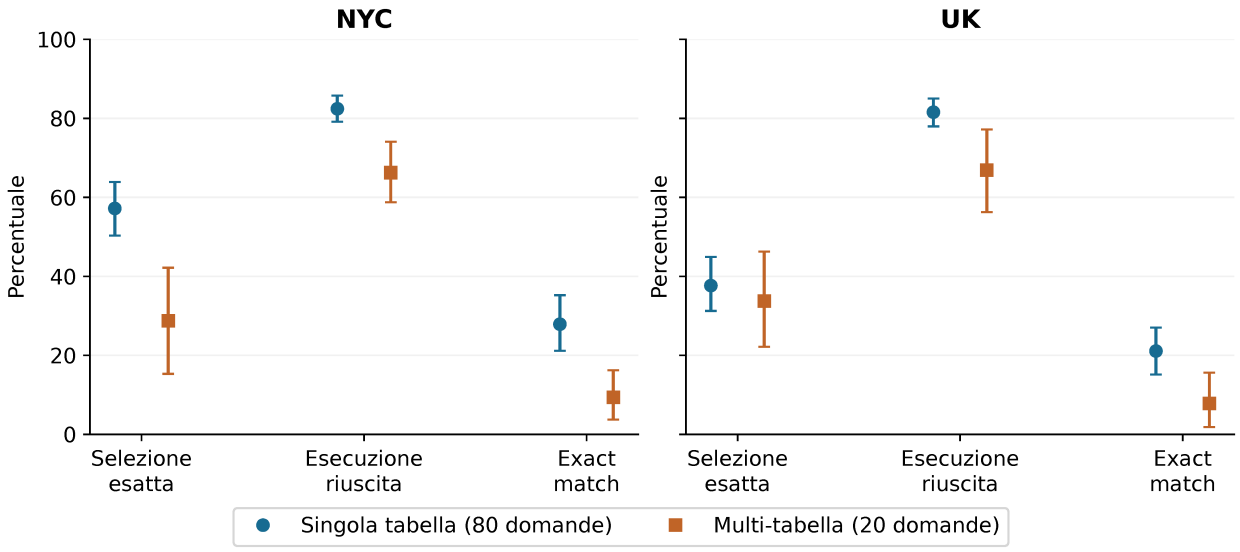


Figura 6.5: Selezione, esecuzione e correttezza per numero di tabelle di riferimento. Le medie riguardano le 16 configurazioni di ciascun portale con coder `full`. Gli intervalli di confidenza bootstrap al 95% sono calcolati sulle domande di ciascun gruppo: 80 su una tabella e 20 su più tabelle.

domande su più tabelle ottengono quindi risultati inferiori nel campione osservato. Il gruppo di 20 domande limita però la precisione delle stime.

Su UK, la selezione esatta cambia poco tra i due gruppi, mentre l'exact match diminuisce molto nelle domande su più tabelle. Le difficoltà possono riguardare anche filtri, aggregazioni e combinazioni successive alla selezione. Le misure disponibili non indicano quante volte ciascuna operazione fallisca: un programma può terminare senza errori pur svolgendo un calcolo sbagliato.

6.9 Variabilità tra esecuzioni

La Tabella 6.9 confronta selezione esatta ed exact match in quattro configurazioni agentiche eseguite due volte. Su NYC, con recupero denso, l'exact match di GPT-OSS passa dal 27% al 23% e quello di Llama dal 21% al 25%. Su UK, con BM25, GPT-OSS passa dal 20% al 25% e Llama dal 19% al 17%. Le variazioni non seguono una direzione comune.

Tabella 6.9: Ripetizioni delle quattro configurazioni agentiche: selezione esatta (Sel.) ed exact match (EM). Valori e accordo in percentuale; differenza ripetizione meno originale in punti percentuali. Ogni confronto usa le stesse 100 domande.

Portale	Recupero	Modello	Metrica	Orig.	Rip.	Δ	Accordo
NYC	Denso	GPT	Sel.	60	62	2	88
NYC	Denso	GPT	EM	27	23	-4	86
NYC	Denso	Llama	Sel.	25	23	-2	90
NYC	Denso	Llama	EM	21	25	4	96
UK	BM25	GPT	Sel.	46	47	1	85
UK	BM25	GPT	EM	20	25	5	85
UK	BM25	Llama	Sel.	38	38	0	96
UK	BM25	Llama	EM	19	17	-2	92

Considerando tutte le metriche, l'esito della stessa domanda coincide nelle due esecuzioni dall'81% al 96% dei casi. Medie simili possono dunque nascondere cambiamenti nelle domande risolte. Per esempio, nel lenient match di GPT-OSS con BM25 su UK, nove domande risultano corrette solo nella prima esecuzione e dieci solo nella seconda. La differenza totale è di un punto percentuale, ma l'esito cambia per 19 domande.

6.10 Discussione e limiti della valutazione

6.10.1 Interpretazione dei risultati

In media, la modalità orchestrata migliora la selezione delle tabelle e riduce il consumo su entrambi i portali. I candidati già recuperati possono aiutare il modello a scegliere. Con BM25 su UK, però, la modalità orchestrata peggiora i risultati: se la ricerca iniziale è troppo restrittiva, il modello non può riformularla.

Su NYC, il recupero semantico supera BM25, senza differenze affidabili tra denso, ibrido e RRF. GPT-OSS seleziona meglio le tabelle e, nel benchmark agentico esteso, ottiene più exact match su

entrambi i portali. Il contesto minimo penalizza soprattutto Llama. Il vantaggio di `full` rispetto a `schema_only` è piccolo.

Nelle condizioni studiate, i risultati suggeriscono una configurazione orchestrata con recupero denso o ibrido e almeno lo schema delle tabelle nel contesto del coder.

6.10.2 Limiti del benchmark e delle misure

Le domande e i programmi di riferimento sono generati automaticamente a partire da tabelle reali. OrQA descrive una verifica umana di un campione della raccolta pubblicata [12]; questo non garantisce la correttezza di ogni riferimento usato qui. Una risposta o una selezione diversa da quella attesa potrebbe quindi essere valida. Anche il giudizio automatico di *supported correct* non sostituisce una verifica umana. Il confronto tra le due modalità riguarda 100 domande per portale; l'analisi agentica estesa ne comprende 381 su NYC e fino a 414 su UK. Nessuno dei due insiemi può essere esteso direttamente ad altri portali o a domande formulate spontaneamente dagli utenti.

Gli intervalli di confidenza misurano la variabilità dovuta alle domande, non quella tra esecuzioni. Solo quattro configurazioni agentiche sono state ripetute. Il servizio OCI non permette di fissare il valore iniziale del generatore casuale; la temperatura è impostata a 0,1.

Le ricerche precedenti possono influenzare quelle successive. Per questo, il ricampionamento delle singole domande potrebbe non cogliere tutta l'incertezza dei risultati.

6.11 Sintesi del capitolo

Il risultato dipende sia dal recupero delle tabelle sia dal loro uso. Su NYC, le strategie con componente semantica migliorano l'exact match rispetto a BM25 anche nel benchmark agentico esteso; su UK non emerge una differenza tra strategie nel campione esteso. L'orchestrazione migliora la selezione in entrambi i portali e riduce in media tempo e token, ma nelle prime 100 domande migliora significativamente l'exact match solo su NYC. GPT-OSS seleziona meglio le tabelle e, nel campione agentico esteso, ottiene più exact match anche su UK. Il contesto minimo penalizza soprattutto Llama.

Le ripetizioni mostrano che differenze di pochi punti possono cambiare tra esecuzioni. Le conclusioni valgono quindi per le domande, i metadati e le condizioni esaminate. Il capitolo successivo presenta le conclusioni e i possibili sviluppi futuri.

7. Conclusioni

Questa tesi ha valutato LakeGen in uno scenario di Question Answering *open-table*. Data una domanda, il sistema cerca le tabelle pertinenti in un portale Open Data, genera ed esegue un programma e produce una risposta. La valutazione esamina il risultato e i passaggi intermedi, per capire dove si verificano gli errori.

Questo capitolo presenta le conclusioni del lavoro, i limiti della valutazione e i possibili sviluppi futuri.

7.1 Bilancio del lavoro svolto

La valutazione ha confrontato quattro strategie di recupero: BM25, recupero denso, fusione ibrida pesata e Reciprocal Rank Fusion. Ha considerato due modelli linguistici, due modalità di accesso agli strumenti e tre livelli di contesto per il coder. Il recupero, la selezione, il programma e il risultato sono stati valutati separatamente.

Le domande e i riferimenti derivano dal processo di costruzione del benchmark OrQA descritto nel Capitolo 4. Gli esperimenti su due portali Open Data sono presentati nel Capitolo 6.

7.2 Significato dei risultati

I risultati mostrano che migliorare il recupero delle tabelle non garantisce, da solo, un risultato corretto. Nel benchmark agentico esteso di New York City, le strategie con una componente semantica superano BM25 nell'exact match di circa sei punti percentuali; sul Regno Unito non emerge invece una differenza altrettanto chiara tra le strategie. L'effetto del recupero dipende quindi anche dalla raccolta su cui viene applicato.

Nel confronto sulle prime 100 domande di ciascun portale, la modalità orchestrata migliora la selezione esatta delle tabelle di circa 21 punti su New York City e 13 sul Regno Unito. Il vantaggio nell'exact match è però chiaro soltanto su New York City. Questo scarto indica che trovare le fonti di riferimento più spesso non basta: il programma deve anche interpretarle e applicare correttamente

le operazioni richieste. Le metriche disponibili non permettono di attribuire ogni errore a una specifica operazione del programma.

La difficoltà è particolarmente evidente nelle domande su più tabelle, che ottengono un exact match inferiore in entrambi i portali. Anche tempo e token vanno letti insieme alla correttezza: un'esecuzione che si arresta presto può consumare meno risorse senza produrre il risultato atteso. La scelta della configurazione richiede quindi di considerare congiuntamente recupero, selezione, risultato e risorse impiegate.

7.3 Limiti delle conclusioni

Il confronto tra modalità usa 100 domande per portale, di cui 20 su più tabelle. Il benchmark agentico esteso comprende 381 domande su New York City e fino a 414 sul Regno Unito. Le prime 100 domande fanno parte anche del campione esteso. Le domande sono generate automaticamente e potrebbero essere diverse da quelle poste dagli utenti. Le stime sulle domande con più tabelle sono meno precise e i dati dei portali possono cambiare nel tempo.

La valutazione esatta usa tabelle e risultati di riferimento: può quindi considerare errate fonti alternative o risposte equivalenti. Il lenient match ammette alcune differenze di formato, mentre *supported correct* usa un giudizio automatico che non sostituisce la verifica umana. La valutazione riguarda il risultato del programma, non la qualità della risposta finale in linguaggio naturale.

La maggior parte delle configurazioni è stata eseguita una sola volta. Nelle quattro configurazioni agentiche ripetute, l'exact match varia fino a cinque punti percentuali. Gli intervalli di confidenza del Capitolo 6 descrivono soprattutto l'incertezza dovuta alle domande. Piccole differenze tra configurazioni non permettono quindi di stabilire con certezza quale sia migliore.

7.4 Sviluppi futuri

LakeGen potrebbe essere valutato su altri portali, su versioni fisse dei dati e con domande raccolte dagli utenti. Ripetere più volte ogni configurazione permetterebbe di misurare meglio la variabilità. Una revisione umana dei casi controversi aiuterebbe a riconoscere fonti alternative valide e risultati equivalenti.

Il sistema potrebbe verificare se i candidati contengono tutte le colonne, i periodi e le tabelle necessari, e ripetere la ricerca quando mancano. Sarebbe inoltre utile classificare gli errori successivi alla selezione, distinguendo filtri, aggregazioni, join, union e forma del risultato. Questo permetterebbe di intervenire sulla fase responsabile.

Le domande che richiedono più di due tabelle meritano una valutazione specifica. Nell'analisi svolta, tutte le domande su più tabelle formano un unico gruppo: non viene misurata separatamente la correttezza delle relazioni individuate né delle operazioni che combinano le tabelle. Per studiare questi errori servirebbero riferimenti verificati e metriche dedicate. Per usare il sistema su raccolte meno controllate servirebbe anche isolare meglio l'esecuzione del programma.

7.5 Considerazioni finali

La valutazione non individua una configurazione migliore in ogni condizione. Mostra però dove i cambiamenti della pipeline producono benefici e dove questi si fermano prima di diventare risultati corretti. Il passaggio dalle tabelle selezionate al calcolo rimane il punto da approfondire, soprattutto quando la domanda richiede di combinare più fonti.

Le conclusioni valgono per i portali, le domande e le esecuzioni esaminate. Verifiche su altri dati, più ripetizioni e un'analisi degli errori del programma permetterebbero di stabilire quanto i risultati osservati siano generalizzabili.

Ringraziamenti

Alla mia famiglia, grazie per essermi stata accanto durante questo percorso. Il vostro sostegno e la fiducia che avete avuto in me mi hanno aiutato ad arrivare fin qui.

Ai miei amici, grazie per le risate, le chiacchierate e i momenti condivisi che hanno reso più belli questi anni.

A Laura, grazie per essermi stata accanto con pazienza e comprensione, anche quando ero molto preso da questo percorso. Hai sempre creduto in me, e poter condividere con te questo traguardo lo rende ancora più bello. Io crederò sempre in te, come tu hai sempre fatto con me.

Bibliografia

[1] Muhammad Imam Luthfi Balaka et al. «Pneuma: Leveraging LLMs for tabular data representation and retrieval in an end-to-end system». In: *Proceedings of the ACM on Management of Data* 3.3 (2025), pp. 1–28.

[2] Gordon V Cormack, Charles LA Clarke e Stefan Buettcher. «Reciprocal rank fusion outperforms condorcet and individual rank learning methods». In: *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*. 2009, pp. 758–759.

[3] Yuhao Deng et al. «LakeBench: A benchmark for discovering joinable and unionable tables in data lakes». In: *Proceedings of the VLDB Endowment* 17.8 (2024), pp. 1925–1938.

[4] Kai Greshake et al. «Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection». In: *Proceedings of the 16th ACM workshop on artificial intelligence and security*. 2023, pp. 79–90.

[5] Jorge Osés Grijalba et al. «Question answering over tabular data with DataBench: A large-scale empirical evaluation of LLMs». In: *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*. 2024, pp. 13471–13488.

[6] Keegan Hines et al. «Defending against indirect prompt injection attacks with spotlighting». In: *arXiv preprint arXiv:2403.14720* (2024).

[7] Madelon Hulsebos, Çağatay Demiralp e Paul Groth. «GitTables: A Large-Scale Corpus of Relational Tables». In: *Proceedings of the ACM on Management of Data* 1.1 (2023), pp. 1–17.

[8] Xingyu Ji et al. «TARGET: Benchmarking table retrieval for generative tasks». In: *arXiv preprint arXiv:2505.11545* (2025).

[9] Vladimir Karpukhin et al. «Dense passage retrieval for open-domain question answering». In: *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*. 2020, pp. 6769–6781.

[10] Aamod Khatiwada et al. «SANTOS: Relationship-based Semantic Table Union Search». In: *Proceedings of the ACM on Management of Data* 1.1 (2023), pp. 1–25.

[11] Christos Koutras et al. «Valentine: Evaluating Matching Techniques for Dataset Discovery». In: *2021 IEEE 37th International Conference on Data Engineering*. IEEE, 2021, pp. 468–479.

[12] Giovanni Malaguti, Angelo Mozzillo e Giovanni Simonini. «OrQA–Open Data Retrieval for Question Answering dataset generation». In: *Proceedings of the 4th Table Representation Learning Workshop*. 2025, pp. 172–181.

[13] Giovanni Malaguti et al. *LakeGen: Multi-Agent Table Discovery on Open Data Lakes*. Poster and demo paper, 23rd European Semantic Web Conference (ESWC 2026). 2026. URL: https://2026.eswc-conferences.org/program/accepted-papers/poster-demo/.

[14] Fatemeh Nargesian et al. «Table union search on open data». In: *Proceedings of the VLDB Endowment* 11.7 (2018), pp. 813–825.

[15] Stephen Robertson e Hugo Zaragoza. «The probabilistic relevance framework: BM25 and beyond». In: *Foundations and Trends in Information Retrieval* 4.1-2 (2009), pp. 1–174.

[16] Shreyas Subramanian et al. «Keyword search is all you need: Achieving RAG-Level Performance without vector databases using agentic tool use». In: *arXiv preprint arXiv:2602.23368* (2025).

[17] Shunyu Yao et al. «ReAct: Synergizing reasoning and acting in language models». In: *arXiv preprint arXiv:2210.03629* (2022).

[18] Luca Zecchini et al. «Determining the Largest Overlap between Tables». In: *Proceedings of the ACM on Management of Data* 2.1 (2024), pp. 1–26.

[19] Erkang Zhu et al. «JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes». In: *Proceedings of the 2019 International Conference on Management of Data*. ACM, 2019, pp. 847–864.