

POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Progettazione e sviluppo di un sistema multi-agente
gerarchico per il dominio turistico



Candidato:
Spanio Lorenzo

Relatore:
Monaco Simone

Tutor aziendale:
Bodojra Simonetta

luglio, 2026

Anno Accademico 2025-2026

Abstract

Il presente lavoro di tesi illustra la progettazione e l'implementazione di un sistema multi-agente (MAS) gerarchico applicato al settore turistico. L'obiettivo principale è l'integrazione di funzionalità di ricerca geospaziale di punti di interesse (POI), la generazione di itinerari personalizzati e la stima dinamica dei profili di rischio ambientale per garantire la sicurezza degli utenti. L'architettura proposta si articola su una struttura gerarchica multi-livello: un nodo orchestratore centrale scompone la richiesta dell'utente e la distribuisce ad agenti specialisti, i quali possono a loro volta delegare specifiche sotto-attività ad agenti di livello inferiore. Ciascun agente elabora autonomamente il sotto-problema assegnato e restituisce l'output al livello superiore, dove le informazioni vengono aggregate in modo incrementale fino alla formulazione della risposta finale. Dal punto di vista tecnologico, lo sviluppo si avvale del framework LangGraph per l'orchestrazione dei flussi e dell'infrastruttura serverless AWS Bedrock AgentCore per l'esecuzione degli agenti. Per garantire scalabilità ed efficienza computazionale, gli agenti sono progettati come componenti stateless; la persistenza dei dati è demandata ad Amazon S3, utilizzando dataset in formato GeoParquet interrogati direttamente tramite DuckDB. I test effettuati confermano la validità dell'approccio multi-agente per la gestione di applicazioni turistiche complesse, evidenziando la modularità e l'estendibilità dell'intera architettura.

This thesis details the design and implementation of a hierarchical Multi-Agent System (MAS) applied to the tourism sector. The primary goal is to support geospatial point-of-interest (POI) queries, personalized itinerary generation, and dynamic environmental safety assessments for users. The proposed architecture is based on a multi-level hierarchical framework: a central orchestrator decomposes the user's request and routes it to specialized agents, which can further delegate sub-tasks to lower-level components. Each agent independently processes its assigned task and returns the output to the upper level, where information is incrementally aggregated to produce the final response. On the technical side, the system is built using the LangGraph framework for orchestration and the serverless AWS Bedrock AgentCore infrastructure for agent execution. To ensure scalability and computational efficiency, the agents operate as stateless components, delegating data persistence to Amazon S3, where GeoParquet datasets are queried via DuckDB. The results validate the multi-agent approach for complex tourism applications, demonstrating the high modularity and extensibility of the architecture.

Indice

Abstract	1
Acronimi e glossario	5
1 Introduzione	6
2 Fondamenti teorici degli Agenti Intelligenti	10
2.1 Classificazione degli agenti	10
2.2 Proprietà fondamentali degli agenti	11
2.3 Architettura di un singolo agente	12
2.3.1 LLM	12
2.3.2 Memoria nell'agente	13
2.3.3 Strumenti per l'agente	15
2.4 Approccio dell'agente a fronte di problemi complessi	17
2.4.1 Decomposizione del problema per un agente	17
2.5 ReAct	18
2.6 Sistema multi-agente (MAS)	19
2.6.1 Definizione e componenti	19
2.6.2 Differenze del multi-agente con un workflow	20
2.6.3 Vantaggi del sistema Multi-Agente	20
2.6.4 Principali architetture per i Multi-Agente	22
2.6.5 Memoria per un sistema multi-agente	25
2.6.6 Human-in-the-Loop (HITL)	26
2.7 Limitazioni dei sistemi multi-agente	27
2.7.1 Allucinazioni, ambiguità e propagazione degli errori	27
2.7.2 Limiti architetturali	29
2.7.3 Limiti di osservabilità	29
2.7.4 Vulnerabilità di sicurezza	29
2.7.5 Limiti della memoria	30
2.7.6 Limiti relativi alla quantità di token	31
2.7.7 Latenza	31
2.7.8 Limiti dovuti alla supervisione umana	32
2.7.9 Costi relativi a un sistema multi-agente	32
3 Framework, infrastruttura cloud, Gestione e organizzazione dei dati	34
3.1 Framework per lo sviluppo di agenti: LangGraph	34

3.2	Infrastruttura cloud e Amazon Bedrock	35
3.3	Amazon Bedrock AgentCore	36
3.3.1	Deploy ed Esecuzione	37
3.4	Amazon S3	37
4	Multi-agente in questa tesi	39
4.1	Motivazione e scelte progettuali	39
4.1.1	La Gerarchia dei Livelli dell'Architettura	40
4.1.2	Pipeline e cicli ReAct	42
4.2	Proprietà condivise tra gli agenti	43
4.3	Human-in-the-Loop	43
4.4	Modularità degli agenti nel sistema	44
4.5	Estensione del sistema ad altri domini	44
4.6	Esempio del funzionamento di questo sistema (per un solo turno)	45
4.7	Contributi individuali e collettivi per questo multi-agente	46
5	Memoria e comunicazione nel multi-agente	48
5.0.1	Glossario dei termini della memoria	49
5.0.2	I componenti della memoria	51
5.0.3	Vantaggi dello Storage S3 rispetto al solo utilizzo del Payload	55
5.0.4	Altre scelte progettuali e precisazioni	56
5.0.5	Conversazioni multi-turno e gestione dei follow-up	57
5.0.6	Vantaggi di questo meccanismo per la memoria	57
5.0.7	Svantaggi di questo meccanismo per la memoria	57
6	L'Agente Orchestratore	59
6.0.1	La Pipeline a tre fasi	59
6.0.2	Gestione del multi-turno	62
6.0.3	Rapporto con gli altri agenti	64
7	Travel Agent	65
7.1	Introduzione	65
7.2	Strumenti a disposizione	65
7.3	Rapporto con gli altri agenti	66
8	L'Activity Agent	68
8.1	Gestione dei dati: due dataset per due scopi differenti	68
8.2	Strumenti	71
8.2.1	Ricerca Attività per Area Amministrativa	71
8.2.2	Ricerca Attività di Prossimità	71
8.2.3	Valutazione delle Condizioni Ambientali	71
8.2.4	Pianificazione Itinerari Ottimizzati	72
8.3	Esempio Itinerario	75
8.4	Rapporto con gli Altri Agenti	76
9	Hiking Agent	78
9.1	Introduzione	78

9.2	Rapporto con gli Altri Agenti	79
9.3	Costruzione del database dei percorsi	79
9.4	Valutazione condizioni ambientali	80
9.4.1	Dalla temperatura dell'aria alla temperatura radiante media	81
9.4.2	Temperatura Percepita (UTCI) e Metabolismo	81
9.4.3	Effetto del vento sulla sicurezza dell'utente	82
9.4.4	L'effetto dell'altitudine sui raggi UV	82
9.4.5	Precipitazioni	82
9.4.6	Veti per la sicurezza	83
9.5	Strumenti dell'agente	83
9.5.1	Filtraggio dei percorsi	83
9.5.2	Build Report e LLM come giudice	85
9.6	I vantaggi di più tools rispetto a una pipeline fissa	86
9.6.1	Adattamento alle richieste	86
9.6.2	Resistenza agli imprevisti	87
9.7	Visualizzazione del percorso migliore	87
10	Weather Agent	89
10.1	Open-Meteo	90
10.1.1	Utilizzo di ICON-EU, GFS Seamless, GFS Ensemble e ECMWF	91
10.2	Strumenti	92
10.3	Rapporto con gli altri agenti	95
11	Location Agent	97
11.1	Introduzione	97
11.2	Architettura del Database Geospaziale	97
11.3	Gestione di operazioni geospaziali	98
11.4	Strumenti a disposizione	99
11.5	Rapporto con gli Altri Agenti	99
11.5.1	Agente dedicato piuttosto che un servizio commerciale	100
12	Limitazioni, sviluppi futuri e conclusioni	101
12.1	Limitazioni	101
12.2	Sviluppi futuri	102
12.3	Conclusioni	103
	Bibliography	106
	List of Figures	107
	List of Tables	107
	Ringraziamenti	108

Acronimi e glossario

Acronimi

AI	Artificial Intelligence
ML	Machine Learning
LLM	Large Language Model (modelli linguistici di grandi dimensioni)
MAS	Multi-Agent System (sistema multi-agente)
POI	Point of interest (punto di interesse)

Glossario dei termini tecnici

Embedding	Rappresentazione vettoriale di un'entità
Pipeline	Sequenza strutturata di fasi di elaborazione
Payload	Pacchetto dati
Parse	Analizzare
Query	Richiesta utente
Context window	Finestra di contesto per un LLM
Tool	Strumento
Task	Compito
Routing	Instradamento

Capitolo 1

Introduzione

Questo lavoro di tesi nasce all'interno dell'azienda Data Reply, presso la quale è stato svolto il tirocinio che ha dato origine al progetto. Il compito affidato consisteva nel progettare e realizzare un'architettura software capace di far collaborare più agenti autonomi per la risoluzione di richieste complesse. Due erano i requisiti fondamentali. Il primo era la scalabilità: il sistema doveva adattarsi a un numero crescente di richieste senza interventi manuali. Il secondo era una manutenzione ridotta al minimo, così da poter estendere l'architettura a nuovi ambiti senza riscrivere le parti già funzionanti.

Per verificare la validità di questa architettura in un contesto reale, si è scelto di applicarla al settore turistico. La scelta non è stata casuale. Una nota azienda tour operator, cliente di Data Reply, aveva infatti espresso l'esigenza di realizzare un assistente in grado di accompagnare il viaggiatore durante la propria vacanza. Il turismo si è rivelato un ambito particolarmente adatto, poiché l'organizzazione di un viaggio, o anche di una sola giornata fuori porta, richiede di gestire contemporaneamente dati molto diversi tra loro: distanze geografiche, previsioni meteorologiche, luoghi di interesse, preferenze personali e tempo a disposizione.

Negli ultimi anni i sistemi basati su modelli linguistici di grandi dimensioni (LLM) hanno mostrato capacità notevoli, ma vengono di norma impiegati come singoli assistenti. Di fronte a un problema complesso, con molte variabili e vincoli impliciti, un singolo agente può incontrare difficoltà nel gestire grandi quantità di testo o istruzioni tra loro contrastanti. In questi casi, può generare risposte errate oppure apparentemente corrette, ma prive di fondamento (le allucinazioni).

Un sistema multi-agente (MAS) affronta il problema da un'altra prospettiva. Anziché affidare l'intera richiesta a un unico modello, la scompone in parti più semplici e ne assegna ciascuna a un agente specializzato, dotato dei suoi specifici strumenti e delle sole informazioni che gli servono. Così ogni agente ragiona su un compito ristretto e ben definito, riducendo il rischio di errori e allucinazioni, mentre il sistema

nel suo complesso conserva la capacità di interpretare richieste espresse in linguaggio naturale e di ricomporre i risultati parziali in una risposta coerente.

Prima ancora di definire i singoli agenti che compongono il sistema, è necessario stabilire in che modo essi comunichino tra loro. La soluzione più immediata sarebbe un'organizzazione piatta, in cui ogni agente può rivolgersi liberamente a qualsiasi altro. Però, questa impostazione ha poco senso, perché emergono due problemi concreti. Il primo riguarda il flusso delle informazioni: se ciascun agente può contattare tutti gli altri in qualunque momento, i messaggi si moltiplicano e diventa difficile ricostruire il percorso di una richiesta, capire chi ha prodotto una certa informazione e individuare dove eventualmente si è verificato un errore. Il secondo riguarda la memoria: non ci si può affidare a un'unica memoria condivisa e visibile a tutti, perché ogni agente ha bisogno soltanto di una piccola parte delle informazioni presenti. Un dato che per un agente è fondamentale risulta inutile per un altro, e raccogliere tutto in un solo spazio comune costringerebbe ciascun agente a leggere anche ciò che non lo riguarda, rendendo più probabili distrazioni ed errori.

Per queste ragioni il sistema è stato organizzato secondo una struttura gerarchica, rappresentabile come un grafo articolato su più livelli. In questo schema una richiesta ampia e generica viene accolta ai livelli superiori, scomposta in sotto-problemi più piccoli e affidata ai livelli inferiori, dove ogni agente lavora esclusivamente sul proprio campo di competenza e riceve solo le informazioni che gli servono. I risultati non tornano all'utente in forma grezza, ma compiono il percorso inverso: risalgono la gerarchia, vengono via via aggregati e sintetizzati dagli agenti intermedi e conducono infine alla risposta conclusiva.

Al vertice di questo sistema si trova l'Orchestratore, l'unico punto di contatto con l'utente. Il suo compito è accogliere la domanda, comprenderne l'intento, individuare i riferimenti geografici e tradurli in una zona geografica precisa, per poi indirizzare la richiesta all'agente più adatto. Al livello immediatamente inferiore operano gli agenti generalisti, che hanno una visione ampia di un intero dominio e coordinano gli specialisti: nel caso del turismo questo ruolo è ricoperto dal Travel Agent. Gli agenti specialisti, a loro volta, si occupano di compiti più mirati: l'Activity Agent individua attività di interesse ed è in grado di creare itinerari personalizzati, mentre l'Hiking Agent si occupa delle esperienze in ambiente naturale, valutando le condizioni meteo per garantire la sicurezza dell'utente. Alla base del sistema si collocano infine gli "esperti dei dati", il cui compito è puramente tecnico e di supporto: il Weather Agent raccoglie e interpreta le previsioni meteorologiche, il Location Agent esegue calcoli spaziali per individuare servizi e punti di interesse sul territorio.

Dal punto di vista tecnologico, l'orchestrazione dei flussi di lavoro è affidata al framework LangGraph, mentre l'esecuzione degli agenti si appoggia all'infrastruttura serverless AWS Bedrock AgentCore, che attiva i singoli agenti solo quando servono e rende in questo modo semplice la scalabilità. I dati geografici sono conservati in

file in formato GeoParquet e interrogati tramite DuckDB.

Il principio secondo cui ciascun agente deve avere accesso solo alle informazioni realmente pertinenti ha influenzato anche la gestione della memoria del sistema. E' stato quindi necessario definire come le informazioni vengono conservate, trasmesse e rese disponibili ai singoli agenti nel momento in cui ne hanno bisogno. La scelta adottata è rendere gli agenti stateless: nessun agente conserva traccia delle chiamate precedenti, ma riparte da zero a ogni invocazione, leggendo dal pacchetto di dati (payload) che riceve soltanto le istruzioni essenziali allo svolgimento del proprio compito. In questo modo la finestra di contesto di ciascun agente resta contenuta e ospita solo ciò che gli serve, evitando la saturazione che deriverebbe dal caricare ogni volta l'intera storia della conversazione. Il payload, tuttavia, è adatto a trasportare solo istruzioni brevi e dati leggeri. Le informazioni più voluminose, come i elementi dei dataset geospaziali e lunghi report meteorologici non transitano nel payload ma vengono depositate su Amazon S3. In questo caso il payload non contiene il dato in sé, bensì un riferimento alla sua posizione, così che ogni agente possa recuperare dallo storage il contenuto solo quando gli è effettivamente necessario. Lo stesso meccanismo è ciò che rende possibile la gestione delle conversazioni su più turni. Quando l'utente pone una domanda di approfondimento, l'Orchestratore non riavvia l'intero processo, ma recupera da S3 lo storico della conversazione e i risultati già prodotti nei turni precedenti, così da riprendere il filo del discorso e riutilizzare il lavoro svolto anziché rielaborarlo da capo.

Il progetto è stato sviluppato in collaborazione con Leonardo Roman, con cui sono state condivise le principali decisioni progettuali (la scelta dell'architettura, il numero di agenti, l'infrastruttura cloud e il meccanismo di memoria) e la realizzazione dell'Orchestratore, componente centrale del sistema. Il contributo presentato in questa tesi riguarda in particolare lo sviluppo del Travel Agent, dell'Hiking Agent, dell'Activity Agent e del Weather Agent, con i relativi strumenti, oltre alla progettazione dell'architettura di comunicazione che permette agli agenti di relazionarsi.

La presente tesi è strutturata come segue:

- Il Capitolo 2 espone i fondamenti teorici degli agenti intelligenti, la loro classificazione, il pattern ReAct e le principali limitazioni dei sistemi multi-agente.
- Il Capitolo 3 presenta i framework e l'infrastruttura cloud utilizzati, con particolare attenzione a LangGraph, Amazon Bedrock AgentCore e Amazon S3.
- Il Capitolo 4 illustra le motivazioni e le scelte progettuali alla base dell'architettura gerarchica sviluppata.
- Il Capitolo 5 approfondisce la gestione della memoria, la comunicazione tra agenti e l'organizzazione dei dati geospaziali.

- I Capitoli da 6 a 11 analizzano nel dettaglio il ruolo, gli strumenti e le interazioni di ciascun agente (rispettivamente Orchestratore, Travel Agent, Activity Agent, Hiking Agent, Weather Agent e Location Agent).
- Il Capitolo 12 raccoglie le limitazioni riscontrate, le possibili direzioni per gli sviluppi futuri e le conclusioni.

Capitolo 2

Fondamenti teorici degli Agenti Intelligenti

2.1 Classificazione degli agenti

La formalizzazione dell'architettura e dei meccanismi di funzionamento del sistema richiede una classificazione preliminare di quali sono i tipi di agenti [1, pp. 115–116]. Una prima tassonomia si basa sulla distinzione tra la natura fisica e digitale delle stesse. Da un lato troviamo gli agenti "incarnati" (embodied agents), entità dotate di un hardware fisico (come sensori, attuatori, bracci meccanici o robot completi) che permette loro di muoversi, percepire e alterare il mondo fisico reale. Dall'altro lato troviamo gli agenti puramente software (digital o disembodied agents), entità che agiscono esclusivamente all'interno di reti e computer. Questi ultimi sono gli unici trattati e approfonditi in questa tesi.

Restringendo il campo ai soli agenti digitali, è possibile operare un'ulteriore classificazione. Questa si basa sul loro livello di autonomia decisionale (agency) e sull'impatto effettivo che le loro azioni hanno sull'ambiente digitale circostante. Seguendo questa logica, gli agenti si dividono in tre categorie principali:

- **Agenti orientati all'azione (Action e Gaming agents):** Gli agenti che operano in simulazioni virtuali sicure come videogiochi o ambienti di training (sandbox). Qui possono agire autonomamente, ma entro regole rigide e confini definiti. Sono "goal-driven", mirano a completare compiti specifici, come vincere una partita o esplorare un ambiente, ottimizzando le decisioni tramite tentativi ed errori per massimizzare il reward ottenibile.
- **Agenti interattivi (Interactive agents):** La differenza rispetto agli Action e Gaming agents non sta nelle capacità del modello, ma nell'ambiente in cui

opera: anziché agire in simulazioni chiuse con regole predefinite, questi agenti interagiscono con sistemi reali come Internet, sistemi operativi locali o piattaforme cloud. L'agente può scrivere e inviare e-mail, aggiornare o cancellare record in un database, eseguire script su server remoti o chiamare API esterne. Tutte queste azioni hanno conseguenze reali e permanenti, diversamente da quanto avviene all'interno di una simulazione. Ciò che accomuna tutti gli agenti interattivi è la capacità di operare in loop autonomi, correggere il proprio comportamento in base ai risultati intermedi e portare a termine obiettivi complessi che richiedono più passi. Questi agenti supportano inoltre comunicazione in linguaggio naturale non solo con utenti umani, ma anche con altri agenti interattivi, rendendo possibile la loro organizzazione e formare sistemi multi-agente, oggetto centrale di questa tesi.

2.2 Proprietà fondamentali degli agenti

In questa sezione vengono espone in modo più dettagliato le proprietà fondamentali degli agenti. Se ne possono individuare quattro, che li distinguono da un software tradizionale:

- **Autonomia:** Rappresenta la capacità dell'agente di operare in modo indipendente, senza la necessità di un intervento o di una supervisione umana. Ricevuto un obiettivo (ad esempio, "organizza un viaggio" o "analizza questo database"), l'agente deve essere in grado di pianificare autonomamente i passaggi intermedi, sfruttare le sue capacità di ragionamento, la conoscenza implicita acquisita dai modelli linguistici (LLM) durante la loro fase di addestramento e usare gli strumenti a sua disposizione.
- **Proattività:** A differenza di un sistema puramente reattivo, un agente possiede una capacità interna al raggiungimento degli obiettivi (goal-driven behavior), che lo porta a prendere iniziativa anche in assenza di una richiesta esplicita. Se l'autonomia riguarda il modo in cui l'agente svolge un compito che gli è già stato assegnato, la proattività compare quando è l'agente stesso ad accorgersi che c'è qualcosa da fare: un'esigenza, un'anomalia, un'informazione mancante e si attiva per affrontarla senza che l'utente o un altro agente gliela segnalino.
- **Reattività:** Definisce la capacità dell'agente di accorgersi di ciò che cambia nell'ambiente in cui opera, ad esempio un errore in una chiamata API, una sorgente dati che cambia struttura o un input dell'utente diverso da quello previsto, e di adattare di conseguenza la propria strategia d'azione, senza restare bloccato. Questa flessibilità garantisce che il sistema non si fermi di fronte a un ostacolo imprevisto, ma cerchi una strada alternativa per aggirarlo.

- **Capacità comunicative:** Gli agenti devono possedere l'abilità di interagire in modo fluido e naturale. Non si tratta solo di saper parlare con gli esseri umani, ma di saper comunicare con altri sistemi intelligenti. Questa proprietà permette a diverse entità di scambiarsi informazioni complesse, delegare compiti, correggersi a vicenda e collaborare attivamente per risolvere problemi complessi.

Mentre l'autonomia e la proattività spingono l'agente ad agire con una certa indipendenza per il raggiungimento di un obiettivo prefissato, la reattività e le capacità comunicative assicurano che l'agente segua le reali esigenze dell'utente e i mutamenti imprevedibili del mondo reale, coordinandosi efficacemente con altri agenti [1, pp. 106–108].

2.3 Architettura di un singolo agente

Un agente singolo è composto da tre componenti che lavorano in stretta collaborazione: il modello linguistico (LLM), la memoria e gli strumenti (tools). Capire come questi elementi si relazionano tra loro è essenziale per comprendere come un agente sia in grado di portare a termine compiti complessi in modo autonomo.

Il modello linguistico costituisce il nucleo centrale dell'agente: è il componente che riceve l'input dell'utente, lo interpreta e decide come procedere. Non si limita a generare una risposta testuale, ma ragiona su quali azioni intraprendere, in quale ordine eseguirle e quando invece è necessario richiedere informazioni aggiuntive all'utente. Per poter operare in modo coerente nel corso di un'interazione, l'agente deve essere in grado di ricordare ciò che ha già prodotto. E' questo il ruolo della memoria, che permette all'agente di mantenere il contesto della conversazione, di tenere traccia delle azioni già eseguite e di non ripetere passaggi già completati.

Infine, gli strumenti estendono le capacità dell'agente oltre la semplice generazione di testo. Tramite i tools, l'agente può interrogare API esterne, leggere e scrivere file, eseguire codice o recuperare informazioni dal web. E' l'LLM a decidere quale strumento utilizzare e quando farlo, sulla base del contesto fornito dalla memoria e dell'obiettivo da raggiungere. Il risultato dell'azione viene poi incluso nel ragionamento dell'LLM, così che possa decidere il passo successivo.

Lo studio di un agente richiede l'analisi approfondita di queste tre componenti, cosa che avviene nelle successive sezioni.

2.3.1 LLM

Un Large Language Model (LLM) è un modello di intelligenza artificiale addestrato su grandi quantità di testo con l'obiettivo di comprendere e generare linguaggio naturale. Alla base di questi modelli c'è l'architettura transformer, che permette di

catturare le relazioni tra le parole in un testo, rendendo il modello capace di ragionare su contesti lunghi e articolati. A distinguere i moderni LLM dai loro predecessori sono le nuove capacità emergenti: oggi questi modelli sanno eseguire istruzioni precise, ragionare per gradi e risolvere problemi nuovi per i quali non sono stati esplicitamente addestrati. All'interno dell'architettura di un agente, l'LLM costituisce il centro decisionale: interpreta gli input in linguaggio naturale (originati dall'utente, dall'ambiente o da altri agenti), decide le azioni da intraprendere e formula le risposte.

Ai fini di questa tesi, l'LLM viene trattato come una black box, infatti ciò che viene approfondito in questa tesi non è tanto il funzionamento interno di questo tipo di modello, ma il ruolo che esso ricopre all'interno del singolo agente. Quindi si è interessati solo agli input che riceve, sugli output che genera e su come interagisce con la memoria e gli strumenti esterni. Detto questo, la scelta del tipo di modello rimane una decisione progettuale importante poiché influenza direttamente le prestazioni dell'agente, la latenza e i costi di utilizzo. Non esiste un modello universalmente ottimale in quanto la scelta dipende dai compiti previsti e dalle risorse disponibili. Vale la pena sottolineare che il modello scelto non deve necessariamente essere di tipo proprietario. Modelli open source come LLaMA o Mistral rappresentano oggi un'alternativa concreta, particolarmente adatta in contesti aziendali o istituzionali in cui la privacy dei dati è un vincolo stringente. Infatti, eseguire questi modelli in locale garantisce che le informazioni sensibili non lascino mai i server dell'utente.

2.3.2 Memoria nell'agente

Senza un qualche tipo di memoria l'agente opererebbe in uno stato di completa amnesia in cui ogni singola richiesta dell'utente verrebbe trattata in modo isolato, partendo da zero, senza alcun riferimento logico a ciò che è avvenuto nei passaggi o nelle interazioni precedenti. Grazie alla memoria, invece, il sistema acquisisce uno "stato", da questo l'agente può richiamare informazioni passate, usare lo storico per rispondere in modo molto più coerente e personalizzato [2, p. 26]. Per comprendere i meccanismi di gestione delle informazioni dell'agente, è possibile suddividere la memoria in quattro livelli, illustrati nella Figura 2.1 e analizzati nel dettaglio di seguito:

- **Memoria interna:** Comprende la conoscenza di base che il modello linguistico acquisisce durante la sua fase di addestramento iniziale. Questa memoria presenta un limite strutturale: essendo statica (ferma alla data di fine addestramento), non è in grado di aggiornarsi autonomamente. Per limitare i fenomeni delle "allucinazioni" (la generazione di informazioni plausibili ma errate), l'agente deve necessariamente dialogare con le memorie esterne descritte di seguito.

- **Finestra di contesto (Context Window):** Rappresenta la capienza massima del modello linguistico, ovvero il limite di testo (misurato in *token*) che l'intelligenza artificiale riesce a leggere e processare. Conseguentemente l'agente deve riuscire a farci stare la domanda dell'utente, le regole di comportamento del sistema e tutte le informazioni necessarie per poter generare una risposta. Essendo uno spazio limitato, non può contenere l'intera cronologia di una lunga conversazione o documenti troppo grandi. Questa restrizione rende necessaria la presenza di un sistema capace di selezionare cosa inserire in questo spazio, compito che spetta alla memoria a breve termine.
- **Memoria a breve termine:** Agisce come un grande contenitore temporaneo. Non si tratta di un meccanismo di elaborazione, ma di uno spazio di archiviazione dinamico in cui si accumulano continuamente la cronologia della conversazione in corso, i risultati provvisori ottenuti dagli strumenti esterni e i dati appena estratti dall'archivio permanente. Poiché questo contenitore si riempie rapidamente e il suo intero contenuto diventa presto troppo grande per entrare nella finestra di contesto, sono utilizzati dei meccanismi esterni di gestione (tecniche di Context Engineering). Questi algoritmi agiscono sul contenitore della memoria a breve termine per prendere dei dati, tagliare i messaggi più vecchi o riassumerne il contenuto, assicurandosi di estrarre da questo grande raccogliatore solo i dati strettamente essenziali da passare al modello nella finestra di contesto.
- **Memoria a lungo termine:** Costituisce il sistema di archiviazione permanente dell'agente, essenziale per superare la natura statica della memoria interna. Nella pratica, questa viene generalmente implementata tramite architetture esterne, come i database vettoriali o a grafo. L'utilizzo di queste tecnologie permette non solo di conservare enormi quantità di dati nel tempo [3], ma soprattutto di indicizzarli e recuperarli tramite una ricerca semantica. Quindi questa memoria permette all'agente di consultare le preferenze storiche dell'utente o di leggere l'intera documentazione di un'azienda.

Il funzionamento pratico di queste memorie si può riassumere in questo modo: quando l'utente fa una nuova richiesta, l'agente cerca innanzitutto nella memoria a lungo termine le informazioni passate o i documenti utili per rispondere, trasferendoli nella memoria a breve termine. All'interno di questo "contenitore temporaneo" si accumulano quindi la domanda attuale, la cronologia della conversazione e i dati appena recuperati. Poiché in generale tutte queste informazioni occupano troppo spazio, intervengono dei meccanismi di context engineering in modo da estrarre solo i concetti veramente importanti. Questo pacchetto di dati essenziali viene poi inserito nella ristretta finestra di contesto, rispettando così i limiti imposti dall'LLM in uso.

A questo punto, l'LLM legge le informazioni presenti nella finestra di contesto, le unisce alle nozioni di base che già possiede (la sua memoria interna) e formula la risposta. Come ultimo passaggio del ciclo, il nuovo messaggio generato viene salvato nella memoria a breve termine per poter proseguire fluidamente la conversazione, mentre le nuove informazioni ritenute utili per il futuro vengono poste nella memoria a lungo termine [3].

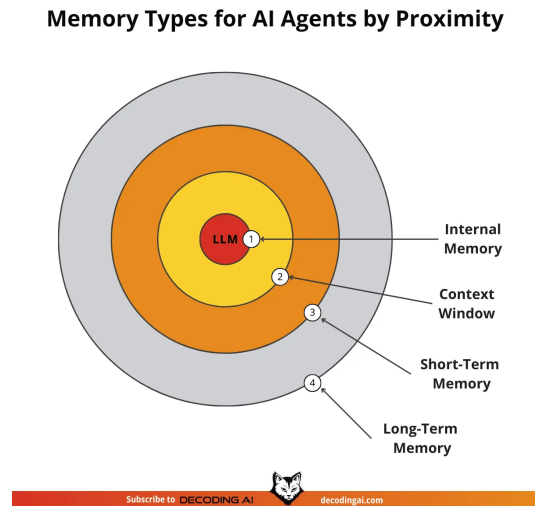


Figura 2.1: Livelli di memoria per un agente intelligente [3]

2.3.3 Strumenti per l'agente

Gli strumenti a disposizione di un agente costituiscono il mezzo pratico attraverso il quale l'intelligenza artificiale interagisce con il mondo esterno. Essi permettono di superare il limite intrinseco degli LLM, i quali, per loro natura, sono in grado esclusivamente di ricevere ed elaborare testo in un ambiente chiuso. L'integrazione dei tools conferisce all'agente la capacità di eseguire azioni specifiche e misurabili in ambienti digitali. La varietà, la qualità e la pertinenza degli strumenti implementati determinano in larga misura la capacità dell'agente di risolvere in modo efficace problemi complessi. In base alla loro funzione, questi strumenti possono essere classificati in tre categorie principali:

- **Strumenti locali:** questa categoria include tutte le azioni e le funzioni che l'agente esegue appoggiandosi esclusivamente all'ambiente in cui è in esecuzione, senza alcuna dipendenza da servizi esterni o connessioni a internet. Sono spesso basati su regole deterministiche. Alcuni esempi tipici includono: l'estrazione e

il filtraggio di informazioni da un database o da un documento salvato in locale o l'esecuzione di calcoli complessi (che un LLM faticherebbe a fare in modo preciso). Il vantaggio di questi strumenti è l'altissima velocità di esecuzione e la garanzia della privacy dei dati.

- **Strumenti basati su API:** quando l'agente ha necessità di intraprendere un'azione fuori dal proprio ambiente locale, fa ricorso agli strumenti basati su API (Application Programming Interface). Le API vengono utilizzate per ottenere informazioni in tempo reale o per compiere azioni su piattaforme di terze parti. Esempi di questi possono essere: il recupero di dati meteorologici aggiornati, la lettura di quotazioni finanziarie in borsa o l'esecuzione di codice su ambienti isolati ospitati da servizi cloud. Grazie a queste l'agente è in grado di formulare risposte contestualizzate e pertinenti alle esigenze dell'utente.
- **Strumenti di lettura e scrittura:** questa categoria comprende tutti gli strumenti che permettono all'agente di interagire con risorse di archiviazione dati in modo persistente. A differenza degli strumenti locali, che elaborano dati già presenti in memoria senza alterare nulla al di fuori del processo in esecuzione, questi strumenti producono effetti duraturi sull'ambiente. Si distinguono anche dagli strumenti basati su API perché non dipendono da servizi esterni o connessioni a internet. Gli strumenti di lettura e scrittura operano su risorse direttamente accessibili nell'ambiente in cui l'agente è presente. Le operazioni tipiche includono la lettura e la scrittura di file in vari formati (testo, PDF, CSV) e l'esecuzione di query su database relazionali o non relazionali. Il principale aspetto critico riguarda proprio la loro natura persistente: un errore dell'agente, come la sovrascrittura di un file importante, può avere conseguenze difficili o impossibili da annullare. Per questo motivo, l'adozione di questi strumenti richiede una progettazione attenta dei meccanismi di controllo e, in alcuni casi, l'introduzione di un supervisore umano che autorizzi le operazioni più critiche.

Un altro aspetto progettuale rilevante riguarda il modo in cui questi strumenti vengono eseguiti in maniera sincrona o asincrona:

- Nell'esecuzione sincrona, l'agente sospende il proprio ragionamento e rimane in attesa finché lo strumento non ha terminato, per poi riprendere con il risultato ottenuto. Questa modalità funziona bene per operazioni veloci dove l'attesa è trascurabile e la risposta può essere restituita all'utente in modo quasi immediato.
- Nell'esecuzione asincrona, invece, lo strumento viene avviato e lasciato lavorare in background in modo tale che l'agente non rimanga bloccato ad aspettarne il completamento, nel frattempo il sistema può svolgere altre attività o chiamare strumenti paralleli.

2.4 Approccio dell'agente a fronte di problemi complessi

L'approccio che un agente utilizza a fronte di problemi complessi può essere visto in due aspetti diversi, uno dal punto di vista della decomposizione del problema in molti sotto-problemi ma di minore dimensione. L'altro modo è come l'agente alterna il ragionamento logico con il processamento di nuove informazioni ottenute dall'ambiente circostante.

2.4.1 Decomposizione del problema per un agente

I problemi del mondo reale, a causa della loro natura ambigua e complessa, non possono quasi mai essere risolti da un agente attraverso un singolo passaggio logico. E' pertanto indispensabile che il sistema sia in grado di effettuare una decomposizione del problema: partendo da una strategia generale, l'agente deve derivare dei sotto-obiettivi (sub-task) operativi e di minore complessità. Per gestire l'interazione tra l'elaborazione di questo piano d'azione (planning) e la sua effettiva messa in pratica (execution), la ricerca scientifica si concentra prevalentemente su due strategie:

- **Pianificazione a priori (Decomposition-first o Plan-and-Solve):** In questo approccio, l'agente costruisce l'intero piano strategico in anticipo, prima di intraprendere qualsiasi azione: deve cioè delineare l'intera sequenza logica della soluzione, passando solo in un secondo momento all'esecuzione pratica. Questo approccio rappresenta un'evoluzione del Chain-of-Thought (CoT), mentre nel CoT tradizionale il modello analizza e risolve il problema contemporaneamente, il Plan-and-Solve impone una separazione netta tra la creazione del piano globale e la sua esecuzione. Il vantaggio principale risiede nel fornire all'agente una visione d'insieme fin dal principio, riducendo il rischio di allucinazioni durante i passaggi intermedi. Tuttavia, questa strategia presenta un difetto intrinseco, ovvero una notevole rigidità. Se il modello formula un'assunzione errata durante la pianificazione o se una risorsa esterna risulta non disponibile, l'agente smette di funzionare correttamente, in quanto incapace di adattarsi agli imprevisti.
- **Pianifica–Agisci–Osserva (Plan-act-observe):** In contrasto con l'approccio precedente, questa strategia unisce pianificazione ed esecuzione in un ciclo continuo. L'agente genera un singolo sotto-obiettivo, lo esegue, ne valuta l'esito tramite i propri strumenti e, sulla base del risultato, pianifica il passo successivo. Questa alternanza tra ragionamento e azione rende il sistema flessibile e capace di aggirare ostacoli imprevisti, con un comportamento più simile a quello umano nell'affrontare problemi complessi. Tuttavia, in presenza di problemi eccessivamente complessi o mal definiti, l'agente rischia di entrare in vicoli ciechi,

generando ragionamenti ciclici e dispendiosi che saturano la context window senza convergere verso una soluzione [1, pp. 115–116].

2.5 ReAct

La strategia Plan-act-observe, descritta nella sezione precedente, rappresenta l'idea alla base di *ReAct* (acronimo di Reasoning e Acting). L'innovazione di questo framework risiede nel continuo alternarsi di due capacità, il ragionamento (Reasoning), che guida il modello nello stabilire cosa fare e come interpretare i dati, e l'azione (Acting), che gli permette di raccogliere informazioni aggiuntive e di modificare l'ambiente. In questo modo, quanto prodotto dall'LLM viene costantemente confrontato con fatti reali, riducendo il rischio di allucinazioni. Nella pratica, il comportamento di un agente *ReAct* non è lineare ma ciclico, e si articola nelle tre fasi visibili in figura 2.2, che si ripetono in sequenza fino alla completa risoluzione della richiesta dell'utente:

- **Ragionamento (Reasoning):** l'agente analizza la richiesta dell'utente o lo stato attuale del problema, valuta le informazioni che possiede, individua quelle mancanti e pianifica il passo successivo.
- **Azione (Acting):** sulla base del ragionamento appena concluso, l'agente sceglie uno strumento a sua disposizione, può invocare una funzione matematica, interrogare un database locale, chiamare un'API esterna per una ricerca sul web o in un'architettura multi-agente, delegare un sotto-compito a un agente specializzato.
- **Osservazione (Observation):** L'agente riceve i risultati dell'azione appena eseguita, ad esempio il testo di una pagina web, una tabella di dati o la risposta di un altro agente, e li include nella propria memoria a breve termine. Questo nuovo dato diventa il contesto di partenza per il ciclo successivo.

La natura iterativa del ciclo rende inoltre il sistema adattivo: se un'azione non produce i risultati attesi, la fase di ragionamento successiva consente all'agente di rivedere la propria strategia e scegliere uno strumento diverso, senza bisogno di un intervento esterno. Infine, ReAct offre un vantaggio anche dal punto di vista dell'ingegneria del software, la trasparenza: il ragionamento, è sempre esplicitato testualmente prima di ogni azione. Ciò permette di ricostruire la sequenza delle decisioni prese dall'agente, facilitando il debugging e la validazione del comportamento nei contesti in cui l'affidabilità del sistema è un requisito critico [4].

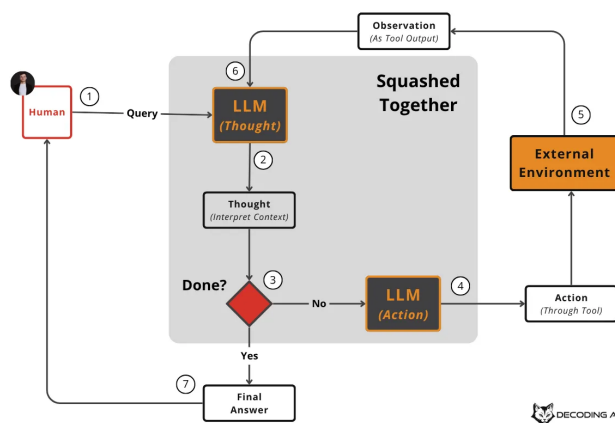


Figura 2.2: Architettura di un agente ReAct [5]

2.6 Sistema multi-agente (MAS)

2.6.1 Definizione e componenti

Un sistema multi-agente (Multi-Agent System, MAS) è un sistema composto da più agenti autonomi che interagiscono tra loro all'interno di un ambiente condiviso per il raggiungimento di un obiettivo prefissato. A differenza dei sistemi tradizionali basati su un singolo agente, in un MAS la conoscenza e le capacità decisionali sono distribuite tra i vari partecipanti. Nessun agente possiede una visione globale dell'intero sistema, ciascuno opera su un sottoinsieme di informazioni strettamente legato al proprio ruolo. Questa distribuzione parziale della conoscenza non è una limitazione, ma una precisa scelta progettuale: assegnare a ogni agente solo le informazioni rilevanti per le proprie competenze consente di specializzarlo nel proprio dominio, aumentandone l'efficacia.

Le modalità con cui gli agenti interagiscono possono variare considerevolmente. In molti scenari gli agenti collaborano, condividendo informazioni e coordinando le proprie azioni per raggiungere un obiettivo comune nel modo più efficiente possibile. In altri casi, tuttavia, gli agenti possono trovarsi in opposizione tra di loro, dove ciascun agente propone una propria soluzione e si scambiano le loro risposte fino a che non si arriva a un risultato che entrambi accettano. Questa dinamica, in apparenza conflittuale, può rivelarsi vantaggiosa in quanto la contrapposizione tra agenti con prospettive diverse favorisce l'esplorazione di un maggior numero di soluzioni e riduce il rischio di convergere verso risposte non ottimali.

Per comprendere appieno il funzionamento di un sistema multi-agente è essenziale analizzare gli elementi fondamentali che lo costituiscono. Generalmente, un MAS si fonda su quattro componenti principali:

- **Gli agenti:** Ogni agente è un'entità capace di percepire ciò che lo circonda, prendere decisioni in modo autonomo basandosi sulle proprie conoscenze e agire per raggiungere i propri scopi. Come anticipato, ogni agente opera su un insieme di informazioni parziali, in base al ruolo che è chiamato a svolgere all'interno del sistema.
- **L'ambiente:** è lo spazio all'interno del quale gli agenti operano e interagiscono. L'ambiente fornisce agli agenti le informazioni necessarie per prendere decisioni e, allo stesso tempo, il suo stato cambia in base alle azioni compiute dagli agenti stessi.
- **Le interazioni:** affinché il sistema raggiunga i propri obiettivi, all'interno del MAS gli agenti devono potersi scambiare informazioni. Questa comunicazione avviene tramite appositi linguaggi e protocolli, può assumere forme sia collaborative quanto competitive, a seconda della natura del problema e dell'architettura scelta.
- **L'Organizzazione:** stabilisce i ruoli degli agenti all'interno del MAS, le gerarchie e le relazioni di dipendenza tra loro. Definisce quindi, in modo concreto, chi fa cosa, chi ha autorità su chi e come gli agenti si coordinano per arrivare a un risultato coerente, evitando che il sistema diventi caotico o che le interazioni portino a esiti contraddittori.

2.6.2 Differenze del multi-agente con un workflow

Un workflow è una sequenza strutturata di operazioni progettata per automatizzare un processo: ogni passo è predefinito, l'ordine di esecuzione è fisso e le condizioni di transizione tra uno stadio e il successivo sono stabilite a priori. Questa rigidità lo rende uno strumento efficace per processi ripetitivi e ben delimitati, ma poco adatto ad affrontare situazioni impreviste o che richiedono adattamento. Un sistema multi-agente si distingue dal workflow proprio su questo aspetto, mentre un workflow segue un percorso predefinito, un MAS è capace di reagire a cambiamenti dell'ambiente, ridistribuire i compiti tra gli agenti e prendere decisioni senza che ogni scenario sia stato anticipato in fase di progettazione.

2.6.3 Vantaggi del sistema Multi-Agente

Le architetture multi-agente si basano sull'utilizzo di diverse entità autonome che operano congiuntamente per il raggiungimento di un obiettivo comune. A seconda

della natura del compito, tali agenti possono agire in modo indipendente, in parallelo o in modo strettamente coordinato. Questi sistemi sono particolarmente utili quando si ha a che fare con problemi complessi, dove la scomposizione in sotto-problemi risulta fondamentale. I quali consentono ai sotto-agenti specializzati di lavorare con maggiore precisione ed efficienza complessiva. L'adozione di un approccio multi-agente offre numerosi vantaggi:

- **Specializzazione:** ciascun agente può essere progettato per operare esclusivamente in un dominio specifico. Ad esempio, un agente può essere dedicato alla raccolta dei dati, un secondo alla loro elaborazione e un terzo alla gestione dell'interfaccia utente. Tale suddivisione consente di affrontare compiti diversi con un livello di competenza e accuratezza superiore rispetto a quello ottenibile da un singolo agente generalista.
- **Collaborazione:** la specializzazione dei singoli agenti trova piena espressione nella loro capacità di cooperare verso un obiettivo comune. Gli agenti possono scambiarsi informazioni, delegare sotto-compiti e integrare i rispettivi risultati.
- **Utilizzo di agenti contrastanti:** un'ulteriore tecnica utilizzata consiste nell'impiego di agenti con ruoli antagonisti. In questo schema, un agente produce una risposta o una soluzione, mentre un secondo agente ne valuta criticamente la correttezza, la coerenza o la qualità, creando un ciclo iterativo tra questi agenti.
- **Parallelismo:** il sistema è in grado di eseguire più azioni in modo simultaneo, a condizione che i compiti siano indipendenti o parallelizzabili. Tale approccio asincrono consente di ridurre significativamente i tempi di esecuzione rispetto a un'architettura puramente sequenziale, ottimizzando l'utilizzo delle risorse computazionali disponibili.
- **Scalabilità:** la natura modulare di queste architetture permette di introdurre agevolmente nuovi agenti con funzionalità aggiuntive. Il sistema risulta pertanto in grado di adattarsi a scenari di crescente complessità senza richiedere un cambio dell'infrastruttura esistente né la riscrittura dei componenti già creati.
- **Ridondanza e resilienza:** poiché le entità operano in modo parzialmente autonomo, il malfunzionamento o il blocco di un singolo modulo non comporta necessariamente un'interruzione dell'intera infrastruttura. Gli agenti rimanenti possono proseguire la propria esecuzione o ridistribuire le responsabilità dell'elemento che ha prodotto l'errore, incrementando l'affidabilità complessiva del sistema [2, pp. 32–33].

Nonostante i vantaggi illustrati, la progettazione di un sistema multi-agente non è priva di complessità. Ogni beneficio strutturale porta con sé dei rispettivi costi: la specializzazione di un agente richiede un'attenta definizione delle sue responsabilità. Il parallelismo richiede la creazione di meccanismi di sincronizzazione, la scalabilità necessita di protocolli di comunicazione robusti e la ridondanza introduce sprechi computazionali. Pertanto progettare un MAS richiede particolare attenzione su come sono organizzati gli agenti e su come strutturare i loro canali di comunicazione. Non esiste un'unica architettura ottimale in senso assoluto, a seconda della natura del problema, del vincolo relativo alla velocità di risposta e delle risorse disponibili, è possibile adottare configurazioni di diversa natura [6].

2.6.4 Principali architetture per i Multi-Agente

Le architetture principali per i multi-agente si possono vedere dalla Figura 2.3. Ora invece vengono qui presentati nello specifico le principali architetture di un MAS, analizzando i vantaggi e svantaggi di ognuna:

- **A Rete (Peer-to-Peer):** tutti gli agenti sono collegati tra loro senza alcuna gerarchia definita e comunicano in modo diretto e orizzontale. È la scelta naturale per sistemi in cui nessun agente ha un ruolo privilegiato e tutti contribuiscono con lo stesso peso al raggiungimento dell'obiettivo. Il vantaggio principale risiede nell'estrema resilienza: non esistendo un nodo centrale di controllo, il malfunzionamento di un singolo agente non compromette l'intero sistema. Il limite principale risiede nella complessità del coordinamento. L'assenza di un supervisore costringe gli agenti a una comunicazione continua per restare sincronizzati e questo massiccio scambio di messaggi può riempire la memoria di ciascun agente, riducendo le risorse disponibili per svolgere il compito per cui è stato progettato e rallentando l'intero sistema. Questo fenomeno diventa particolarmente critico all'aumentare del numero di agenti.
- **Gerarchica:** il sistema è organizzato su più livelli decisionali. Un agente supervisore (o manager) coordina le attività di un insieme di sotto-agenti, delegando a ciascuno di essi una porzione specifica del compito complessivo. La comunicazione avviene prevalentemente in senso verticale: il supervisore emette istruzioni verso il basso e riceve report di stato verso l'alto, riducendo la necessità di coordinamento diretto tra agenti allo stesso livello. È l'architettura ideale quando il problema può essere facilmente scomposto in sotto-task ben definiti. Tra i principali vantaggi vi sono la trasparenza del flusso decisionale e la semplicità con cui è possibile aggiungere o sostituire sotto-agenti senza alterare la struttura complessiva. Tuttavia, questa architettura introduce alcuni limiti significativi: il supervisore rappresenta un Single Point of Failure, per cui un suo

malfunzionamento o una decisione errata si propagano all'intero ramo gerarchico. Inoltre, in scenari ad alta parallelizzazione, il supervisore può diventare un collo di bottiglia, limitando la scalabilità del sistema.

- **Supervisore:** un agente supervisore coordina l'operato degli altri agenti, decidendo quale invocare in base allo stato corrente del task e all'output prodotto. Ogni sotto-agente è dotato di un proprio LLM, memoria e logica interna, ed è quindi un'entità autonoma a tutti gli effetti. Anche il supervisore stesso è un agente dotato di un proprio LLM che ragiona sull'output ricevuto, e non un semplice nodo di smistamento. Il supervisore non esegue operazioni dirette, ma agisce da orchestratore: interviene per correggere errori o fornire feedback. Conseguentemente non esiste una pianificazione a priori: il supervisore osserva l'output di ogni agente e decide il passo successivo solo dopo averlo valutato, instaurando di fatto un ciclo di controllo attivo tra invocazione e revisione. Tra i vantaggi principali si trovano: la robustezza agli errori perchè c'è una fase di controllo dal supervisore, la modularità, poiché i sotto-agenti sono indipendenti tra loro e possono essere sostituiti o estesi senza alterare la logica del supervisore. Inoltre il sistema risulta trasparente e interpretabile data l'indipendenza degli agenti. Gli svantaggi riguardano il fatto che supervisore ad ogni ciclo introduce latenza aggiuntiva, rendendo l'architettura più onerosa rispetto a soluzioni prive di supervisione esplicita. La qualità complessiva della risposta rimane dipendente dalla capacità del supervisore di rilevare errori e formulare feedback pertinenti.
- **Supervisore con strumenti (Tool-calling Supervisor):** in questa architettura l'agente centrale agisce come uno smistatore piuttosto che come un revisore attivo. Infatti, a differenza dell'architettura precedente, il supervisore non analizza nel dettaglio l'output dei sotto-agenti per fornire feedback o richiedere correzioni, ma si limita a comprendere la richiesta iniziale, selezionare il sotto-agente più adatto e passargli i parametri necessari. Ragionamento ed esecuzione sono delegati al singolo sotto-agente invocato, che elabora il task e restituisce il risultato al nodo centrale. E' importante notare che gli "strumenti" richiamati dal supervisore non sono strumenti deterministici in senso classico, ma agenti completi, ciascuno con il proprio LLM, memoria e logica interna, esposti attraverso l'interfaccia di tool-calling tipica degli LLM: il supervisore li invoca come se fossero funzioni, senza accedere o intervenire sul processo di ragionamento autonomo che avviene al loro interno. Il vantaggio principale risiede nell'efficienza: eliminando i cicli di revisione del supervisore riduce i tempi di esecuzione e il consumo di token. E' perciò adatto a processi pratici e ben definiti. Il limite maggiore è che se un sotto-agente produce un output impreciso, il supervisore tende ad accettarlo passivamente e inoltrarlo. Il sistema risulta quindi meno

robusto, poiché il successo del task dipende quasi interamente dalla capacità del singolo strumento di completare il lavoro al primo tentativo.

- **Custom:** nei casi reali particolarmente complessi, le architetture standard possono non essere sufficienti, rendendo necessaria la costruzione di un'architettura su misura. I vantaggi principali sono la flessibilità e la possibilità di ottimizzare ogni parte del sistema in modo indipendente, anziché applicare lo stesso schema di coordinamento a tutti gli agenti, si può scegliere quello più adatto a ciascun sotto-gruppo in base al compito che deve svolgere. Questo porta a prestazioni globali migliori rispetto all'uso di un'unica architettura uniforme. Tuttavia, questa architettura introduce il rischio di inconsistenze nelle comunicazioni e nelle responsabilità tra agenti, rendendo la fase di design particolarmente critica. Il tracciamento degli errori risulta complesso, poiché comportamenti anomali possono originarsi dall'interazione tra componenti progettati secondo logiche differenti. Inoltre, la manutenzione è più onerosa, poiché modifiche localizzate possono produrre effetti inattesi su altri componenti. Nonostante questi limiti, la flessibilità di questa architettura la rende una scelta frequente nello sviluppo di moderni sistemi multi-agente.

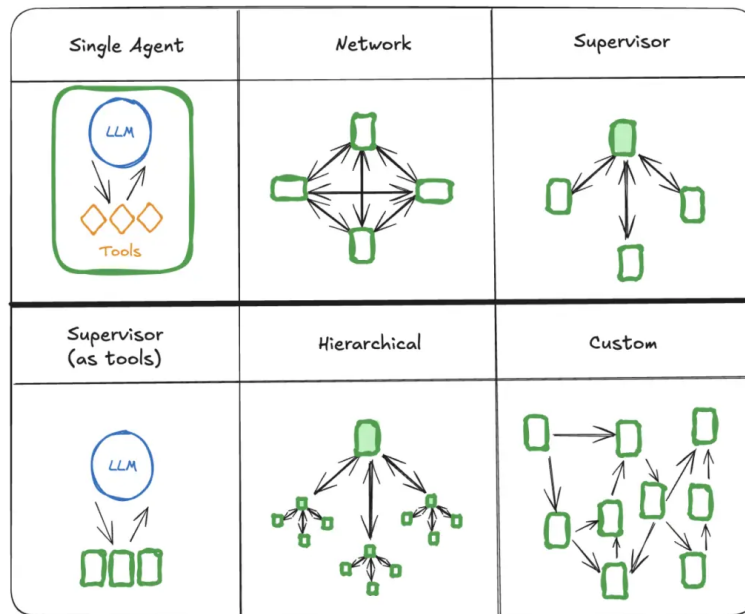


Figura 2.3: Architetture possibili per un sistema multi-agente [7].

2.6.5 Memoria per un sistema multi-agente

La gestione della memoria in un sistema multi-agente richiede un'attenzione particolare rispetto al caso del singolo agente. Infatti, in un sistema con un solo agente, la memoria riguarda esclusivamente il rapporto tra l'agente e il proprio contesto. In un MAS invece la questione è più complicata: bisogna decidere non solo cosa ricordare, ma anche chi può accedere a cosa e come le informazioni vengono trasmesse tra agenti diversi.

In generale si possono identificare principalmente quattro approcci nella gestione della memoria:

- **Memoria condivisa (Shared Memory):** gli agenti selezionati leggono e scrivono sulla stessa struttura dati, comprendendo tutti gli output degli strumenti e le risposte finali prodotte dagli agenti. Questo approccio garantisce la massima trasparenza tra gli agenti, ma introduce rischi di coerenza, ad esempio, se più agenti scrivono contemporaneamente nella memoria, possono emergere conflitti o informazioni contraddittorie. Inoltre, al crescere del numero di agenti e della durata della sessione, la struttura condivisa può diventare molto voluminosa e possibilmente può eccedere la Context Window. Anche il costo di esecuzione del sistema diventa oneroso in quanto tutti gli agenti condividono tutte le informazioni quando magari solo una piccola parte sarebbe necessaria per svolgere il task ad uno specifico agente.
- **Passaggio di messaggi (Message Passing):** ogni agente riceve soltanto le informazioni che l'agente chiamante ha scelto di trasmettergli, insieme alla propria storia interna e alla richiesta originale dell'utente. Questo approccio limita la visibilità di ciascun agente a ciò che è strettamente necessario migliorando la focalizzazione del compito da svolgere. Lo svantaggio principale è che alcune informazioni potenzialmente utili possono non essere trasmesse se il chiamante non le ritiene rilevanti.
- **Agenti senza stato (Stateless):** ogni agente non mantiene alcuno storico tra un'invocazione e l'altra e opera esclusivamente sul contesto che riceve al momento della chiamata. Questo modello è il più semplice da implementare e il più facilmente scalabile, in quanto non richiede alcuna sincronizzazione tra invocazioni successive. E' particolarmente adatto a compiti ben definiti, dove ogni richiesta è indipendente dalle precedenti.
- **Memoria esterna (External Memory):** indipendentemente dall'approccio adottato per la trasmissione del contesto tra agenti, i sistemi multi-agente possono appoggiarsi a componenti di archiviazione esterna come database vettoriali

per gestire informazioni persistenti tra sessioni diverse o volumi di dati che eccedono la disponibilità della context window. Ad ogni invocazione, l'agente recupera solo le informazioni rilevanti tramite ricerca semantica, evitando di caricare l'intera storia nel contesto. Questo approccio funziona bene su sessioni molto lunghe o con grandi volumi di dati, ma introduce una dipendenza con componenti esterni e richiede che il processo di recupero sia sufficientemente accurato da non tralasciare dettagli critici.

E' importante sottolineare che non è necessario adottare un unico approccio per l'intero sistema: all'interno dello stesso MAS possono coesistere strategie diverse, con ciascun agente (o gruppo di agenti) che adotta il meccanismo di gestione della memoria più adatto al proprio ruolo, in base al numero di agenti coinvolti, alla natura dei compiti da svolgere, ai requisiti di latenza e alla quantità di informazioni da gestire nel tempo. Una soluzione ibrida combina, ad esempio, il passaggio di messaggi per la comunicazione tra agenti specializzati, la memoria condivisa per il coordinamento dello stato globale, e la memoria esterna per la gestione della conoscenza storica persistente. Questa flessibilità è uno dei punti di forza dei sistemi multi-agente, ma richiede una progettazione accurata.

2.6.6 Human-in-the-Loop (HITL)

In scenari caratterizzati da alti livelli di rischio, lasciare al MAS un'autonomia incondizionata introduce rischi di stabilità. E' quindi necessario implementare logiche di Human-in-the-Loop (HITL), che permettano all'operatore umano di validare, sospendere o correggere le decisioni prese dal sistema al verificarsi di determinate condizioni. Il grado di partecipazione dell'umano non è fisso, ma può variare a seconda del livello di rischio del dominio in cui il MAS agisce. L'intervento umano può avvenire in momenti diversi:

- **Prima dell'esecuzione (human-in-the-loop pre-task):** l'operatore valida il piano d'azione elaborato dagli agenti prima che questo venga effettivamente eseguito. E' la forma di controllo che riduce al minimo il rischio di azioni errate, al costo di una maggiore latenza nel processo.
- **Durante l'esecuzione (human-on-the-loop):** il sistema procede autonomamente, ma l'umano monitora in tempo reale e può interrompere o correggere il comportamento degli agenti in qualsiasi momento.
- **Dopo l'esecuzione (human-in-the-loop post-task):** l'operatore esamina i risultati prodotti dagli agenti e decide se accettarli, scartarli o richiedere una nuova elaborazione. Questo approccio è adatto quando le azioni sono reversibili o quando il costo di un errore è contenuto.

La scelta del punto di intervento non è esclusiva, all'interno di sistemi più sofisticati possono combinare più modalità, ad esempio richiedere l'utilizzo di una validazione pre-task e poi una human-on-the-loop per le operazioni più semplici.

L'integrazione di un supervisore umano porta numerosi vantaggi:

- **Controllo della qualità e gestione dei casi limite:** l'operatore può verificare la coerenza e la correttezza delle risposte, intervenendo nelle situazioni inattese o ambigue.
- **Conformità normativa:** in settori come quello medico, legale o finanziario, la presenza di un supervisore umano per validare le operazioni è un requisito imposto dalla legge.

E' importante considerare anche i limiti di questo approccio. L'intervento umano introduce inevitabilmente latenza nel processo e non scala facilmente in presenza di volumi elevati di richieste. Inoltre, la qualità della supervisione dipende dalla competenza e dall'attenzione dell'operatore. Per questo motivo, la progettazione di un sistema HITL richiede di identificare quali decisioni esigono davvero l'intervento umano e quali, invece possono essere delegate in sicurezza.

2.7 Limitazioni dei sistemi multi-agente

2.7.1 Allucinazioni, ambiguità e propagazione degli errori

Poiché ogni agente si affida a un LLM come metodo di ragionamento, le limitazioni intrinseche di questi modelli influenzano inevitabilmente l'intero sistema. Tra queste, la più critica è il fenomeno delle cosiddette "allucinazioni": risposte che risultano plausibili nella forma ma fattualmente errate o che distorcono la realtà dei fatti. A ciò si aggiunge il fatto che le richieste poste dagli utenti si presentano spesso troppo vaghe, incomplete o persino contraddittorie. Per mitigare questi problemi, un'architettura ben sviluppata non si limita a fornire risposte generiche nel tentativo di indovinare cosa l'utente intendeva dire, ma adotta un approccio proattivo. Infatti, l'agente è istruito per porre domande mirate all'utente, chiarendo le ambiguità del contesto prima di procedere alla risoluzione del compito [2, p. 65]. Tuttavia è importante che questo meccanismo sia bilanciato con attenzione in quanto un agente che chiede troppe conferme rischia di risultare frustrante, mentre uno che ne chiede troppo poche rischia di procedere su informazioni incomplete. Inoltre il sistema deve sapersi dimostrare capace di ammettere i propri limiti, riconoscere i suoi fallimenti ed evitare iterazioni circolari superflue [2, p. 65].

In un MAS questi rischi sono amplificati, poiché l'output errato di un agente, che si tratti di un'allucinazione o di una distorsione può propagarsi come input agli agenti

successivi, compromettendo l'intera catena di ragionamento. Perciò vengono qui analizzate le criticità che possono condurre al fallimento di un sistema multi-agente, queste sono riconducibili a cinque macro-categorie:

- **Limiti architetturali:** ruoli definiti in modo ambiguo, l'assenza di una gerarchia strutturata e il rischio di loop infiniti possono impedire al sistema di operare in modo coordinato ed efficace.
- **Vulnerabilità di sicurezza:** il fatto che il sistema multi-agente sia interconnesso amplia la possibilità di attacco da avversari esterni, esponendo il sistema a minacce come la prompt injection o il data poisoning.
- **Limiti di osservabilità:** Quando un sistema multi-agente produce una risposta errata, non è semplice individuare il punto preciso in cui un agente ha gestito le informazioni in modo scorretto. Anche potendo osservare i ragionamenti dei singoli agenti, la collaborazione tra più componenti, distribuita su passaggi sequenziali e paralleli, introduce un'opacità che rende difficile ricostruire il percorso logico dietro la risposta finale.
- **Limiti della memoria:** la scelta dell'approccio di gestione della memoria introduce rischi diversi, tra cui la saturazione del contesto, la perdita di informazioni e il recupero impreciso dei dati.
- **Limiti relativi alla quantità di token:** la context window di ciascun agente, determinata dalla somma di token di input, reasoning e output, ha una capienza massima, oltre la quale le informazioni più datate vengono scartate o compresse senza preavviso. Nei sistemi multi-agente ciò è particolarmente dannoso, poiché un agente che perde informazioni rilevanti propaga l'errore agli agenti successivi.
- **Latenza:** la complessità della self-attention scala quadraticamente con la lunghezza dell'input [8]. Nei sistemi multi-agente, dove i token crescono rapidamente, l'esecuzione sequenziale degli agenti accumula le latenze lungo la catena, aspetto critico nelle applicazioni in tempo reale.
- **Limiti dovuti alla supervisione umana:** l'introduzione di un operatore umano aumenta l'affidabilità dei sistemi multi-agente, ma introduce possibili colli di bottiglia dovuti all'affaticamento dell'operatore e dalla dipendenza dalle sue competenze. Inoltre, la scalabilità diventa problematica con l'aumentare della complessità.
- **Costi relativi a un sistema multi-agente:** I sistemi multi-agente possono diventare costosi perché richiedono molte interazioni tra agenti e l'uso di servizi esterni, facendo crescere rapidamente le spese. Per questo è necessario bilanciare

prestazioni e costi. Sistemi più complessi sono sì più efficaci, ma anche meno sostenibili su larga scala.

Le sottosezioni seguenti analizzano ciascuna di queste categorie nel dettaglio.

2.7.2 Limiti architetturali

Una definizione ambigua delle responsabilità tra gli agenti all'interno di un sistema multi-agente impedisce al sistema di allinearsi correttamente alle intenzioni dell'utente. La progettazione dell'architettura non riguarda solo la scelta del numero di agenti o il tipo di comunicazione, ma implica decisioni fondamentali su come suddividere il problema, quali competenze assegnare a ciascun agente e come garantire che le loro azioni rimangano coerenti con l'obiettivo globale. Una comunicazione inefficiente tra gli agenti porta spesso a comportamenti incoerenti e a un peggioramento della qualità della risposta, causando anche uno spreco di risorse computazionali. Questi problemi si aggravano nei sistemi più complessi, dove il numero elevato di agenti rende la coordinazione significativamente più difficile.

2.7.3 Limiti di osservabilità

In generale, un sistema multi-agente produce ragionamenti interni complessi, tra più agenti che operano sia in modo sequenziale che in parallelo, passando attraverso numerosi passaggi intermedi. La collaborazione di molti agenti porta a risposte più precise, ma introduce un livello di opacità che rende difficile, anche per lo sviluppatore, garantire una piena tracciabilità delle decisioni prese: quando il risultato finale è prodotto dalla collaborazione di numerosi agenti, non è immediato ricostruire quale percorso logico abbia portato a quella risposta.

2.7.4 Vulnerabilità di sicurezza

La natura interconnessa del MAS fa sì che la compromissione di un singolo agente non rappresenti un problema isolato, un comportamento malevolo può propagarsi rapidamente agli altri agenti, amplificando i danni in modo difficilmente prevedibile. Rispetto ai sistemi basati su un singolo modello, ogni agente, canale di comunicazione e strumento esterno possono essere i bersagli da attacchi esterni. Una delle minacce più frequenti è la prompt injection, ovvero l'inserimento di istruzioni malevole all'interno degli input elaborati dagli agenti. L'attacco è di natura indiretta in quanto essa non proviene direttamente dall'utente, ma da fonti esterne che l'agente consulta durante la propria attività, come pagine web, documenti o risultati di ricerche [9]. In un sistema multi-agente questo tipo di attacco è particolarmente insidioso, poiché le istruzioni malevole possono essere trasmesse da un agente all'altro

attraverso i normali canali di comunicazione, senza che nessuno di essi sia in grado di riconoscere la minaccia. Il data poisoning è un'altra minaccia, che agisce però a un livello più profondo rispetto al prompt injection: invece di manipolare le istruzioni che l'agente riceve, altera direttamente i dati su cui l'agente si basa per ragionare, come le basi di conoscenza esterne o i sistemi di recupero delle informazioni. In questo modo, un attaccante può corrompere i dati per orientare il comportamento del sistema verso i risultati che desidera, senza dover interagire direttamente con gli agenti. Anche in questo caso nei MAS questo problema si aggrava per via della natura distribuita del sistema. Cioè gli stessi dati corrotti possono influenzare più agenti contemporaneamente, rendendo l'intero sistema inaffidabile.

2.7.5 Limiti della memoria

La scelta di come gestire la memoria in un sistema multi-agente introduce difficoltà diverse a seconda dell'approccio adottato. Nel caso della memoria condivisa, il contesto cresce continuamente: ogni messaggio dell'utente, ogni risposta generata dagli agenti e ogni risultato prodotto dagli strumenti esterni viene accumulato nella struttura condivisa. Sebbene esista la possibilità di utilizzare un orchestratore che gestisca questa struttura e possa in teoria filtrare o adattare le informazioni prima di consegnarle a ciascun agente, nella pratica questo non avviene quasi mai e nella maggior parte dei casi si finisce a passare a ogni agente l'intero contesto disponibile. In un sistema con molti agenti, questo accumulo avviene rapidamente, in quanto la quantità di informazioni trasportate a ogni scambio cresce sia con il numero di agenti che con la lunghezza della conversazione. Questo ha un impatto significativo su costi, latenza e qualità dell'output.

Quando gli agenti comunicano scambiandosi singoli messaggi, anziché l'intero contenuto della propria finestra di contesto, il rischio principale è la perdita di informazioni nel loro scambio. Poiché questi comunicano trasmettendosi le informazioni in modo selettivo, se uno di essi omette un dettaglio fondamentale, l'agente successivo lavorerà con dati incompleti. Questo problema tende ad amplificarsi nel corso del processo, in quanto ogni agente dispone esclusivamente delle informazioni che gli sono state trasmesse direttamente, senza alcuna visione globale dello stato del sistema.

Nel caso di utilizzo di una memoria esterna, l'approccio riduce significativamente il volume di dati trasportati a ogni scambio, ma introduce una dipendenza da componenti esterni e il rischio che le informazioni rilevanti non vengano recuperate correttamente. Questo può avvenire a causa di una formulazione imprecisa della richiesta oppure di un indice del database vettoriale non correttamente strutturato, con il risultato che l'agente opera su un contesto incompleto pur non essendone consapevole.

Utilizzando agenti privi di memoria persistente (stateless) si elimina il problema di accumulo, ma rendono impossibile qualsiasi ragionamento che richieda di fare riferimento a passi precedenti. L'unica alternativa è ricostruire esplicitamente la storia rilevante a ogni invocazione, ricadendo di fatto nel problema dell'eccessivo volume di informazioni già discusso per la memoria condivisa.

2.7.6 Limiti relativi alla quantità di token

Come illustrato in Figura 2.4, i token elaborati da un LLM si suddividono in tre categorie. I token di input rappresentano il testo fornito al modello come contesto, inclusi i messaggi precedenti, le istruzioni di sistema e i dati ricevuti. I reasoning token sono prodotti internamente dal modello durante il processo di ragionamento, non compaiono nell'output finale ma occupano comunque spazio nella finestra di contesto. Gli output token costituiscono infine il testo effettivamente generato come risposta. Il limite della quantità di token all'interno della context window si applica alla somma di tutte e tre le categorie.

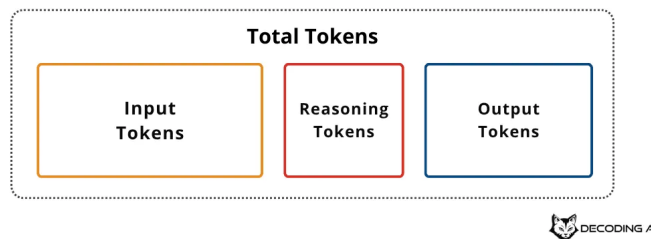


Figura 2.4: Suddivisione dei token: input, reasoning e output [5].

Indipendentemente da come viene gestita la memoria, la context window di ciascun agente ha una capienza massima. Quando viene raggiunta, il sistema che gestisce il contesto deve necessariamente scartare o comprimere parte delle informazioni (tipicamente quelle più datate) senza che l'utente ne sia informato. Questo comportamento è particolarmente dannoso nei sistemi multi-agente in quanto un agente che perde informazioni rilevanti tenderà a produrre output parziali o errati che verrà trasmesso all'agente successivo, propagando l'errore.

2.7.7 Latenza

Un ulteriore limite riguarda la latenza. La complessità computazionale del meccanismo di self-attention scala quadraticamente con la lunghezza della sequenza di input, ovvero $\mathcal{O}(n^2)$ [8]. Questo significa che raddoppiare la lunghezza dell'input

non raddoppia il tempo di elaborazione, ma lo quadruplica. Come si è visto nella sottosezione dedicata ai limiti della memoria, in un sistema multi-agente la quantità di token può crescere rapidamente, soprattutto se quest'ultima non viene gestita correttamente. Se il sistema prevede inoltre l'esecuzione sequenziale di più agenti, le latenze si accumulano: il ritardo introdotto da ciascun agente si somma a quello dei precedenti, allungando progressivamente i tempi di risposta lungo l'intera catena. Questo aspetto diventa particolarmente critico nelle applicazioni che richiedono risposte in tempo reale, dove una latenza elevata può compromettere il suo utilizzo indipendentemente dalla qualità della risposta prodotta.

2.7.8 Limiti dovuti alla supervisione umana

Come si è visto in 2.6.6 l'introduzione di un supervisore umano comporta vantaggi sul punto di vista di affidabilità dell'intero sistema. Tuttavia nell'introdurre questo meccanismo di sicurezza sorgono anche alcune criticità:

- **Collo di bottiglia:** se il sistema richiede approvazione umana per ogni passo, la velocità complessiva del sistema multi-agente è fortemente limitata.
- **Affaticamento:** quando un umano supervisiona un sistema complesso per lunghi periodi questo può diventare mentalmente faticoso. Con il tempo, l'operatore tende ad approvare le azioni in modo acritico, riducendo l'efficacia del controllo.
- **Dipendenza dalle competenze dell'operatore:** la qualità della supervisione dipende dalla preparazione dell'umano. Un operatore non esperto potrebbe non accorgersi di errori di piccola dimensione nelle risposte degli agenti.
- **Scalabilità:** all'aumentare del numero di agenti e della complessità dei task, la quantità di decisioni da supervisionare cresce rapidamente, rendendo difficile mantenere un controllo efficace senza aumentare il numero di operatori.

La scelta del livello di intervento umano rappresenta un compromesso tra sicurezza e autonomia: sistemi più controllati sono più affidabili ma meno efficienti, mentre sistemi più autonomi scalano meglio ma riducono la possibilità di intercettare errori.

2.7.9 Costi relativi a un sistema multi-agente

I sistemi multi-agente che adottano modelli commerciali sono soggetti a un modello di fatturazione pay-as-you-go, in cui il costo è calcolato in base al numero di token elaborati da ciascuna chiamata all'API del modello linguistico. A differenza di una singola interazione, in un'architettura multi-agente la risoluzione di un task complesso richiede che più agenti comunichino tra di loro, invocino strumenti esterni

e in caso di errore ritentino l'operazione. Una gestione inefficiente del sistema ha una doppia conseguenza: può degradare la qualità delle risposte, come discusso nella sottosezione 2.7.6 ma anche genera costi evitabili poiché la generazione di token non necessari vengono comunque elaborati e fatturati. A questi costi si aggiunge quello dovuto alle API esterne, come servizi meteorologici o motori di ricerca, il cui contributo complessivo può diventare rilevante in un sistema quando il bacino di utenti diventa ampio. L'insieme di questi fattori impone di considerare attentamente il trade-off tra prestazioni e budget già in fase di progettazione, poiché il controllo dei costi rappresenta uno dei principali ostacoli alla scalabilità dei sistemi multi-agente.

Capitolo 3

Framework, infrastruttura cloud, Gestione e organizzazione dei dati

Una volta stabilito come gli agenti collaborano tra loro, resta da decidere come costruire davvero il sistema. Oltre alla comunicazione tra agenti, serve un'infrastruttura che gestisca la memoria, regga un numero crescente di richieste e conservi i dati nel tempo. All'interno di questa tesi sono state adottate le seguenti tre tecnologie:

- **LangGraph**: il framework di orchestrazione utilizzato per modellare i flussi di lavoro, gestire i cicli di ragionamento e coordinare lo stato condiviso tra i diversi agenti del sistema.
- **Amazon Bedrock e AgentCore**: l'infrastruttura cloud che esegue gli agenti e dà un accesso ai modelli linguistici, con la sicurezza e la scalabilità che servono a un'applicazione reale.
- **Amazon S3**: il sistema di archiviazione a oggetti dove gli agenti salvano le informazioni che non entrano nella finestra di contesto, come documenti e risultati prodotti durante l'elaborazione.

3.1 Framework per lo sviluppo di agenti: LangGraph

Per tradurre un'architettura multi-agente dalla teoria alla pratica è necessario adottare un framework di orchestrazione capace controllare più agenti in modo rapido e intuitivo. Tra le soluzioni più adottate spicca LangGraph, il framework utilizzato in questa tesi. LangGraph modella il flusso di lavoro di un agente come un grafo orientato in cui nodi, archi e condizioni di transizione vengono dichiarati nella fase di progettazione.

I tre elementi fondamentali sono qui approfonditi:

- **Lo stato:** un oggetto dati condiviso passato e aggiornato di nodo in nodo. Generalmente in un sistema multi-agente diventa la memoria condivisa attraverso cui gli agenti si coordinano. Al crescere della complessità, tuttavia, uno stato globale unico rischia di diventare difficile da interpretare e viene "inquinato" da informazioni irrilevanti. Per questo nelle architetture più complesse si affiancano memorie esterne e meccanismi alternativi di comunicazione, come si è gestita questa problematica si veda il capitolo 5.
- **I nodi:** Un nodo riceve in input lo stato corrente, esegue un'operazione e restituisce un aggiornamento dello stato stesso. La complessità del compito eseguito può variare enormemente: può trattarsi di una semplice elaborazione di dati, dell'invocazione di un LLM o dell'esecuzione di uno strumento esterno. Grazie alla funzionalità dei subgraphs (sotto-grafi), un nodo può persino racchiudere al suo interno architetture molto più complesse, come un intero agente autonomo (ad esempio ReAct) o un intero sistema multi-agente annidato.
- **Gli archi condizionali:** permettono ad una LLM di passare da un nodo all'altro. Sono il meccanismo che dà autonomia al modello linguistico: leggendo lo stato corrente, è lui stesso a decidere se fermarsi e consegnare la risposta finale, oppure, ad esempio, se andare in un altro nodo in cui viene usato uno strumento esterno ,

Oltre alla struttura a grafo, LangGraph offre una serie di funzionalità che lo rendono adatto allo sviluppo di sistemi agentici robusti e scalabili. I principali punti di forza sono:

- **Grafi ciclici:** a differenza dei tradizionali flussi unidirezionali (DAG), LangGraph supporta percorsi circolari, abilitando così l'approccio ReAct.
- **Supporto asincrono e retry:** i processi asincroni consentono di lavorare in parallelo, riducendo la latenza complessiva. I meccanismi automatici di retry rendono il sistema resiliente a cadute di connessione, Rate Limits e sovraccarichi dei server.
- **Human-in-the-Loop:** Langgraph consente di sospendere l'esecuzione in attesa dell'approvazione umana, implementando il pattern Human-in-the-Loop (sezione 2.6.6).

3.2 Infrastruttura cloud e Amazon Bedrock

Realizzare un sistema multi-agente (MAS) richiede un'infrastruttura in grado di garantire disponibilità continua e capacità di calcolo scalabile. Per evitare significativi

investimenti iniziali in hardware e manutenzione, in questo progetto si è scelto di adottare i servizi cloud di Amazon Web Services (AWS), basati su un modello pay-per-use, in cui si pagano esclusivamente le risorse consumate. Per quanto riguarda l'interazione con le LLM è stato utilizzato Amazon Bedrock. Questo servizio espone un singolo endpoint API per accedere ai modelli di frontiera di diverse compagnie (come Anthropic, Meta, Mistral AI).

3.3 Amazon Bedrock AgentCore

Se Bedrock standardizza l'accesso ai modelli, l'esecuzione e l'orchestrazione degli agenti richiedono un ambiente dedicato. La logica di un MAS si basa solitamente su framework open-source, come LangGraph o CrewAI, che lasciano però allo sviluppatore il problema della gestione dell'infrastruttura sottostante e la sua scalabilità. In questo progetto si è scelto di adottare Amazon Bedrock AgentCore proprio perché permette di eseguire nativamente tutti i principali framework sul mercato, sollevando lo sviluppatore dalla complessa gestione. AgentCore funziona come un ambiente e permette di scalare le risorse in automatico e adotta una fatturazione basata esclusivamente sul tempo di calcolo effettivo, escludendo i tempi di attesa legati alle operazioni di I/O [10]. Il sistema è composto da vari moduli indipendenti. In questo lavoro vengono sfruttati i seguenti:

- **Runtime:** gestisce l'esecuzione degli agenti scalandoli automaticamente in base alla domanda.
- **Gateway:** trasforma API esistenti e funzioni AWS Lambda in strumenti pronti all'uso per gli agenti, supportando lo standard MCP (Model Context Protocol).

Gli altri moduli previsti dall'ecosistema (Browser, Observability, Policy e Identity) rappresentano estensioni naturali per sviluppi futuri, mentre per la memoria si è preferito implementare una soluzione custom basata su Amazon S3, descritta nel Capitolo 5, al posto del modulo standard di AgentCore.

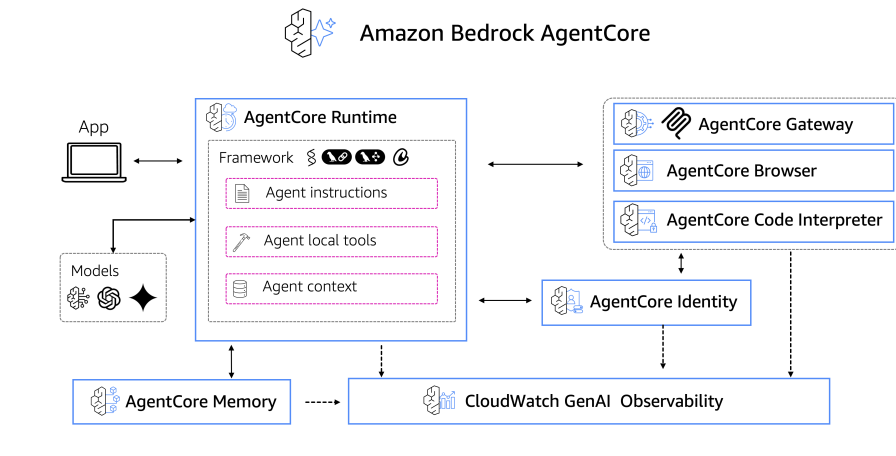


Figura 3.1: Insieme dei componenti di Amazon Bedrock AgentCore. Immagine tratta da [11] sotto licenza Apache 2.0.

3.3.1 Deploy ed Esecuzione

La messa in produzione tramite il modulo Runtime segue una pipeline definita. Il codice dell'agente e le relative dipendenze vengono pacchettizzati in un'immagine Docker compilata per architettura ARM64 e caricata sul registro di Amazon Elastic Container Registry (ECR). Alla ricezione di una richiesta, AgentCore recupera l'immagine da ECR e avvia in pochi millisecondi una microVM, un ambiente virtuale isolato in cui viene eseguito il container. A differenza di servizi serverless tradizionali come AWS Lambda, che richiedono l'allocazione preventiva della memoria, AgentCore non necessita di configurazioni preventive. Nel contesto di questo progetto, gli agenti sono stati progettati per essere completamente stateless e conseguentemente ogni microVM elabora la singola interazione in totale isolamento, delegando il salvataggio e il recupero dello stato della conversazione al servizio di storage Amazon S3 ad ogni nuova invocazione.

3.4 Amazon S3

Amazon Simple Storage Service (S3) è il servizio di archiviazione a oggetti offerto da AWS. A differenza dei tradizionali file system gerarchici, S3 organizza i dati in bucket, ovvero contenitori dalla capacità virtualmente illimitata. Al loro interno non esistono sottocartelle: ogni risorsa è salvata come singolo "oggetto" ed è identificata esclusivamente da una chiave univoca.

La chiave non è un semplice identificatore da ricercare tra tutti quelli presenti, ma corrisponde direttamente alla posizione fisica dell'oggetto, rendendo il recupero immediato indipendentemente dalla quantità di dati archiviati. Proprio grazie a questa combinazione di scalabilità e accesso rapido rende s3 una valida scelta per gestire la memoria per un'architettura multi-agente. L'adozione di questo strumento comporta i seguenti vantaggi:

- **Estensione della memoria:** la finestra di contesto di un LLM impone un limite fisso alla quantità di informazioni che l'agente può elaborare in un singolo ciclo di ragionamento. Con S3 il calcolo e la conservazione dei dati restano separati. In questo modo l'agente può lavorare con grandi quantità di informazioni, come documenti, dataset o cronologie delle interazioni e recuperare solo ciò che gli serve al momento.
- **Persistenza e disponibilità:** Su S3 l'agente ritrova le informazioni salvate nelle sessioni precedenti e conserva i propri risultati in modo stabile nel tempo.

Capitolo 4

Multi-agente in questa tesi

4.1 Motivazione e scelte progettuali

Il progetto risponde all'esigenza, manifestata da Data Reply, di produrre un'architettura multi-agente flessibile ed estendibile a domini eterogenei, anche molto distanti tra loro. Per soddisfare tale requisito, durante la fase di progettazione e dopo aver valutato vari framework, si è optato per AWS Bedrock AgentCore come piattaforma di riferimento. La sua capacità di attivare on-demand i container dei singoli agenti, infatti, permette di ottimizzare l'uso delle risorse computazionali che risultano così facili da scalare all'occorrenza. A livello strutturale, il sistema si basa su un'architettura gerarchica organizzata su quattro livelli. Si tratta tuttavia di una gerarchia personalizzata (custom), poiché i flussi possono saltare alcuni step a seconda della complessità della richiesta. Questa architettura è stata scelta proprio perché consente di suddividere un problema complesso in sotto-problemi più piccoli e isolati. In questo modo, si lascia che siano di volta in volta gli agenti più idonei a intervenire in modo mirato, anziché ricorrere a un unico modello linguistico generico che faticherebbe a gestire numerosi strumenti e a rispondere per ambiti così diversi tra loro.

La scelta di questo approccio multi-agente offre tre vantaggi fondamentali:

- **Modularità:** Ogni agente è indipendente e isolato dal punto di vista logico e infrastrutturale. Questo consente di aggiungere nuovi agenti o espandere il sistema ad altri domini applicativi apportando modifiche minime ai componenti già esistenti.
- **Scalabilità:** Il sistema è in grado di gestire in modo completamente autonomo qualsiasi picco o fluttuazione nel traffico degli utenti. Delegando interamente ad AWS il compito di scalare automaticamente le risorse di calcolo necessarie.

- **Precisione:** Gli agenti specializzati e i Data Expert hanno accesso diretto a strumenti tecnici specifici (come database GeoParquet su S3, API OpenStreet-Map e Open-Meteo), garantendo risposte accurate e basate su dati reali, cosa che un modello linguistico generico non potrebbe gestire con lo stesso livello di dettaglio.

4.1.1 La Gerarchia dei Livelli dell'Architettura

L'architettura del sistema, rappresentata nella Figura 4.1, è strutturata su quattro livelli gerarchici. Questa divisione non è casuale, ma risponde a un principio di specializzazione progressiva: scendendo dal primo all'ultimo livello, ci si sposta da una visione d'insieme (orientata all'interazione con l'utente) verso compiti e competenze sempre più specifici e legati a singoli domini. In questo modo, la richiesta iniziale dell'utente viene suddivisa in problemi sempre più piccoli, delegando la risoluzione di questi agli agenti che possiedono le competenze migliori per affrontarla.

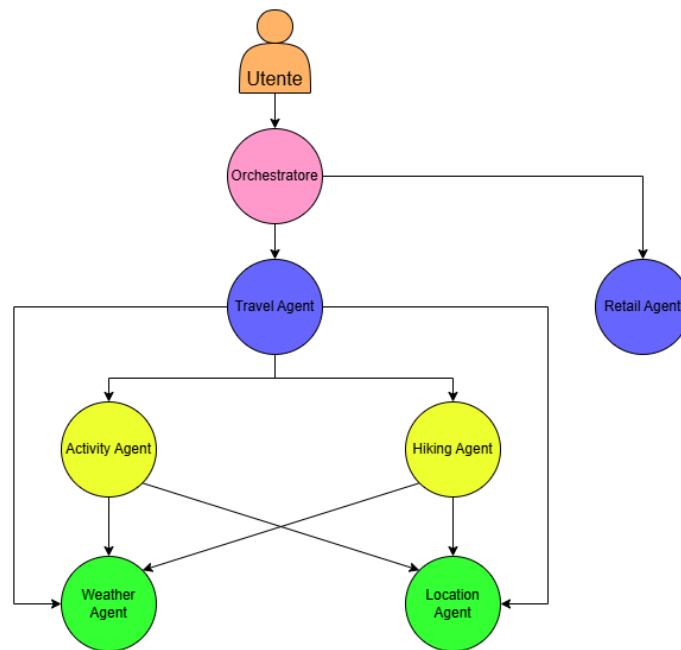


Figura 4.1: Architettura del sistema multi-agente. L'utente (in arancione) comunica con l'orchestratore (di livello 1, in rosa), indirizza la query a un generalista (di livello 2, in blu), che a sua volta invoca uno o più specialisti (livello 3, in giallo) che consultano se necessario i data expert (livello 4, in verde).

Di seguito vengono descritti i singoli livelli gerarchici e gli agenti che ne fanno parte:

Livello 1: L'Orchestratore (Rosa)

Rappresenta il punto di ingresso dell'intero sistema ed è costituito solo dall'Orchestratore. Il suo compito è prendere la richiesta dell'utente, analizzarla per comprenderne l'intento, estrarre le informazioni geografiche essenziali (traducendo i nomi delle località in geometrie reali) e una volta preparato questo contesto di base, indirizzarla verso l'agente più idoneo del livello inferiore.

Livello 2: Livello Generalista (Blu)

Gli agenti di questo livello hanno una visione ampia di un intero dominio. Il loro compito non è effettuare previsioni o elaborare dati geografici, devono invece essere in grado di saper coordinare i sotto-agenti del proprio dominio assegnando loro compiti più specifici.

A questo livello troviamo due agenti:

- **Travel Agent:** è l'agente responsabile del dominio turistico. Il suo ruolo è interpretare le esigenze dell'utente (come trovare un ristorante nelle vicinanze, attività specifiche o luoghi di interesse) tenendo conto del suo profilo (preferenze alimentari, budget o ritmo preferito), delegando poi le richieste più tecniche (come l'analisi delle condizioni ambientali o la pianificazione di un itinerario giornaliero) agli agenti specialisti del livello inferiore.
- **Retail Agent:** Rappresenta l'agente generalista per il dominio commerciale. A differenza del dominio turistico che costituisce il cuore del sistema, questo agente e le sue funzionalità non sono state approfondite o implementati nella versione attuale del progetto.

Livello 3: Livello Specialistico (Giallo)

Quando gli agenti generalisti del livello superiore hanno bisogno di maggiore precisione e dettaglio per una specifica attività invocano gli agenti specialisti, che operano su un dominio di applicazione ancora più ristretto.

- **Hiking Agent:** è lo specialista dell'escursionismo. E' in grado di consigliare percorsi naturalistici estraendo dal database esclusivamente quelli che rispettano le caratteristiche richieste dall'utente. Inoltre, valuta le condizioni meteo su più giorni e fasce orarie per verificare che siano favorevoli alle attività all'aria aperta, accertandosi che non vi siano rischi per la sicurezza.
- **Activity Agent:** è lo specialista delle attrazioni e del tempo libero. E' in grado di suggerire in base alle preferenze dell'utente i punti di interesse (POI) più adatti, che siano storici, culturali o ricreativi. Può anche utilizzare un

algoritmo di ottimizzazione per organizzare un itinerario giornaliero adatto alle richieste dell'utente,

Livello 4: Esperti Ambientali (Verde)

Alla base della gerarchia si trovano gli agenti esperti di dati fisici e territoriali. Il loro compito non è pianificare o analizzare le informazioni in una visione d'insieme, ma estrarre, raccogliere e sintetizzare i dati sul territorio esclusivamente in base ai parametri e alle caratteristiche specificate dagli agenti dei livelli superiori.

- **Weather Agent:** L'esperto del meteo. Interroga API meteorologiche in tempo reale, fornisce previsioni dettagliate, interroga database storici e li interpreta.
- **Location Agent:** L'esperto di ricerca spaziale. Il suo ruolo è unicamente quello di raccogliere dal database i punti di interesse (POI) presenti nell'area o lungo il percorso indicati dagli specialisti superiori.

E' importante notare che la comunicazione all'interno di questa gerarchia segue una percorso a "doppio senso". Mentre la fase di attivazione e delega si muove dall'alto verso il basso (o rimane in un loop iterativo all'interno di uno stesso agente), quando un agente dei livelli inferiori completa il proprio compito, il suo output risale la gerarchia in senso contrario rispetto alle frecce di attivazione. Questo processo di "risalita" fa sì che le risposte vengano via via rielaborate, aggregate e sintetizzate dagli agenti dei livelli superiori, fino a ritornare all'Orchestratore, che consegna la risposta finale all'utente.

4.1.2 Pipeline e cicli ReAct

Oltre alla gerarchia dei ruoli, il sistema si basa su una netta distinzione nel modo in cui i vari agenti elaborano le informazioni e prendono decisioni. Nel sistema coesistono due soli approcci logici:

La Pipeline Lineare (Livello 1) L'Orchestratore adotta un approccio basato su una pipeline logica sequenziale e unidirezionale. Nella primissima fase del processo, infatti, l'Orchestratore interpreta il contesto tramite un LLM per comprendere l'intento dell'utente e decidere a quale agente generalista indirizzare la query. Una volta conclusa questa fase di ragionamento, i passi successivi operano in modo rigorosamente deterministico, senza ulteriori chiamate all'LLM. Per i dettagli sulle singole fasi si rimanda al capitolo 6.

Il Ciclo ReAct (Livelli 2, 3 e 4) Tutti gli altri agenti hanno una struttura di tipo ReAct. Questo è dovuto al fatto che tutti questi agenti non possono seguire un flusso rigido, ma devono operare in un ciclo iterativo continuo per poter risolvere problemi complessi. Per maggiori dettagli sulla struttura e caratteristiche per dei sistemi ReAct.

4.2 Proprietà condivise tra gli agenti

L'architettura gerarchica definisce le differenze tra gli agenti, il ruolo che ciascuno ricopre, l'area di competenza e il livello di autonomia. Esistono, tuttavia, proprietà che accomunano tutti i componenti del sistema, indipendentemente dalla loro posizione nella gerarchia.

Tutti gli agenti seguono un modello di esecuzione stateless: ogni invocazione parte da una context window vuota, cioè senza che venga mantenuto alcuno stato interno tra una chiamata e l'altra. Questo approccio evita che gli agenti accumulino contesto irrilevante nel tempo. Per comprendere cosa questa natura stateless comporti all'interno di un sistema multi-agente, inclusi gli aspetti positivi e negativi, si rimanda al capitolo specifico della memoria. Un'altra caratteristica che accomuna gli agenti è l'utilizzo dello stesso modello linguistico: Claude Haiku 4.5 di Anthropic, eseguito tramite Amazon Bedrock. E' stato selezionato per la sua elevata rapidità e per il basso costo di esecuzione, evitando l'uso di modelli più avanzati che avrebbero rallentato il sistema multi-agente e generato costi eccessivi.

Infine, tutti gli agenti condividono la stessa struttura di deployment basata su Bedrock AgentCore, come si è visto nel capitolo precedente.

Le proprietà descritte in questa sezione sono da considerarsi valide per tutti gli agenti del sistema. Nei capitoli dedicati ai singoli agenti non verranno quindi richiamate esplicitamente, tranne nei casi in cui un agente richieda una precisazione specifica.

4.3 Human-in-the-Loop

Il sistema è progettato per operare in modo completamente autonomo. Tuttavia, data la complessità delle richieste che possono essere formulate dall'utente, è possibile integrare una modalità di supervisione umana per incrementare l'affidabilità e la solidità dei risultati.

In questo scenario il sistema multi-agente può essere eseguito tre volte in parallelo a partire dallo stesso input dell'utente. A causa della natura stocastica dei Large Language Model impiegati, si ottengono tre risposte differenti. Queste ultime vengono sottoposte a un operatore umano che ne valuta l'accuratezza e la coerenza,

selezionandone una sola, che costituirà l'output finale restituito all'utente. Questo meccanismo di revisione riduce il rischio che allucinazioni o risposte non del tutto accurate vengano consegnate all'utente finale.

L'adozione di questa modalità presenta tuttavia uno svantaggio: aumenta la stabilità del sistema grazie al controllo umano ma introduce una latenza non indifferente, poiché i tempi di risposta complessivi non dipendono più esclusivamente dall'elaborazione computazionale, ma anche dai tempi di lettura e approvazione dell'operatore.

4.4 Modularità degli agenti nel sistema

Ciascuno degli agenti presenti in questo sistema sono stati sviluppati per poter essere impiegati anche in modo indipendente. Questa scelta progettuale si può vedere nell'insieme degli strumenti messi a disposizione di ciascun agente: non tutti sono strettamente necessari al funzionamento del sistema multi-agente descritto in questa tesi, ma contribuiscono a rendere ogni agente completo nel proprio ambito di competenza. Ogni agente può quindi essere estratto dal sistema e integrato in un progetto differente senza perdere le proprie capacità essenziali, riducendo al minimo le modifiche necessarie per adattarlo a nuovi sistemi. Questa modularità porta con sé un ulteriore vantaggio: è possibile modificare o aggiornare un singolo agente senza interrompere il funzionamento dell'intero sistema. Per questa ragione, nei capitoli a seguire verranno descritti tutti gli strumenti di ogni agente, inclusi quelli non strettamente necessari per il funzionamento di questo specifico sistema, per fornire una visione completa delle capacità di ciascun componente.

4.5 Estensione del sistema ad altri domini

Sebbene questa tesi si concentri sul dominio turistico, il sistema è progettato per gestire domini applicativi anche molto differenti tra loro. A dimostrazione di ciò, è stato integrato un Retail Agent al livello 2 dell'architettura, parallelo al Travel Agent, verso cui l'orchestratore è in grado di dirigere le query pertinenti. Non è stato possibile dedicare tempo sufficiente al suo sviluppo, ma la sua implementazione risulta notevolmente facilitata ora che l'infrastruttura di base è stata già costruita.

4.6 Esempio del funzionamento di questo sistema (per un solo turno)

Per capire appieno questo esempio è necessario conoscere il significato dei termini specifici che sono stati adottati per questo sistema, che però non sono stati ancora ben definiti finora. Si è scelto comunque di inserire questa sezione in questo capitolo in quanto è l'unico che ha una prospettiva ad alto livello sul funzionamento di tutto il sistema.

Un esempio di richiesta che il sistema è in grado di gestire è il seguente: “Consigliami un itinerario giornaliero per Torino. In base alle mie preferenze e alle previsioni meteo dei prossimi tre giorni, dimmi tu il giorno migliore in cui andare.”

1. **Avvio orchestratore** L'orchestratore si attiva e cerca la cronologia dei messaggi precedenti ma poiché si tratta del primo turno della conversazione, il registro è completamente vuoto.
2. **Analisi della richiesta** L'orchestratore analizza il messaggio dell'utente per comprenderne il significato. In questa fase esso determina quale sarà l'agente più adatto a invocare (in questo caso il Travel Agent). legge la richiesta per individuare il luogo geografico richiesto dall'utente (in questo caso Torino) e crea una versione arricchita della domanda originaria.
3. **Risoluzione geografica** L'orchestratore prende il nome della destinazione (Torino) e tramite l'algoritmo deterministico, individua la sua posizione esatta e i suoi confini territoriali sotto forma di poligono geografico. Questo poligono viene salvato come file nello spazio di archiviazione online (Amazon S3) e associato a un codice identificativo unico. Da questo momento in poi, l'intero sistema farà riferimento a Torino usando solo questo codice, evitando di allegare l'intero file del poligono nei messaggi tra gli agenti.
4. **Invio del payload** L'orchestratore invia la richiesta al Travel Agent. Per farlo, crea un payload contenente la richiesta dell'utente, il codice identificativo della geometria e quello della sessione.
5. **Avvio dell'agente e pre-caricamento.** L'agente generalista invocato (il Travel Agent) prima di iniziare il suo ragionamento, usa il codice identificativo della geometria per scaricare i confini reali di Torino e le coordinate geografiche, salvandoli nella propria memoria locale in modo da averli subito pronti per le ricerche. Viene anche scaricato i dati contenenti tutte le informazioni che si hanno sull'utente.

6. **Invocazione dell'agente specialista.** Il travel agent inizia a ragionare e capisce che per pianificare l'itinerario e scegliere il giorno migliore in base al meteo deve consultare l'Activity Agent. Dunque quest'ultimo viene invocato e gli viene passato un'altro payload.
7. **Chiamata al Data Expert** L'Activity Agent capisce che per scegliere il giorno migliore ha bisogno delle previsioni del tempo per i prossimi tre giorni. Avvia quindi il Weather Agent, chiedendogli di analizzare il tempo previsto per Torino per quel periodo.
8. **Recupero dei dati meteo.** L'agente del meteo recupera le previsioni per Torino, le salva in un file nello spazio di archiviazione online generandone il codice identificativo e restituisce le informazioni sull'andamento del tempo all'Activity Agent.
9. **Scelta del giorno e creazione dell'itinerario.** L'activity agent confronta le previsioni del tempo con le preferenze dell'utente (ad esempio evitando i giorni di pioggia) e seleziona il giorno ideale per la gita. Successivamente, cerca le attrazioni migliori a Torino, pianifica la sequenza delle visite per la giornata e l'Activity agent salva le informazioni trovate all'itinerario dell'archivio file online, generando un proprio codice identificativo. Infine, manda il payload di ritorno al Travel Agent inviandogli l'itinerario e la lista dei codici dei file creati (compresi quelli del meteo).
10. **Scrittura della risposta e invio del payload.** Il travel Agent ottiene la risposta e i codici dei file ricevuti dall'Activity Agent. Elabora quindi la risposta finale in linguaggio naturale per l'utente. Successivamente, inserisce questa risposta e la mappa completa di tutti i codici dei file generati (sia da lui stesso che dagli agenti attivati precedentemente) all'interno di un pacchetto di dati di ritorno (il payload). Questo pacchetto viene inviato all'orchestratore.
11. **Aggiornamento della cronologia e chiusura.** L'orchestratore riceve la risposta finale e tutti i codici generati. Aggiorna il Registro Conversazione su S3 inserendovi la richiesta originale, la risposta finale e i codici di archiviazione dei file. Il file della sessione viene salvato online per essere riutilizzato al turno successivo, dopodiché l'orchestratore invia la risposta all'utente (insieme ad eventuali figure) e a questo punto l'orchestratore si spegne.

4.7 Contributi individuali e collettivi per questo multi-agente

Lo sviluppo di questo progetto è stato fatto grazie all'azienda Data Reply e svolta in collaborazione con Leonardo Roman. In questa sezione vengono esplicitati quali

ruoli si hanno avuti nello sviluppo di questo sistema multi-agente. E' necessario precisare fin da subito che la struttura stessa di questo progetto impone che esista una stretta collaborazione tra gli sviluppatori. Infatti, la decisione di che architettura utilizzare, quanti agenti implementare, a quale infrastruttura cloud affidarsi e come viene implementata la memoria, sono tutte scelte progettuali fatte necessariamente in collaborazione. L'unico agente singolo che è stato svolto in collaborazione è l'orchestratore, questo infatti ricopre un ruolo centrale nel sistema, è quindi stato naturale la collaborazione per la sua implementazione.

Ora invece vengono descritte le parti del progetto che sono state svolte in autonomia: Leonardo è stato responsabile dell'effettiva implementazione dell'architettura (AWS bedrock, AWS agentCore, S3 ed anche il deployment), dello sviluppo del Location Agent e della gestione dei dati. Quest'ultimo aspetto è stato fatto tramite la costruzione di apposite pipeline in grado raccogliere e creare dei dataset. Io, invece, mi sono dedicato principalmente allo sviluppo degli agenti che compongono questo sistema multi-agente: in particolare ho lavorato su Travel Agent, Hiking Agent, Activity Agent e Weather Agent, i loro strumenti e l'architettura di comunicazione tra di essi.

Capitolo 5

Memoria e comunicazione nel multi-agente

Gestire lo stato in un sistema multi-agente significa decidere quali informazioni salvare, dove memorizzarle e chi può accedervi. Poiché la memoria e la comunicazione tra gli agenti sono strettamente collegate, in questo capitolo verranno analizzati questi due aspetti. La scelta di come strutturare la memoria influenza direttamente la velocità del sistema, la coerenza dei dati e la capacità dell'infrastruttura di gestire carichi di lavoro crescenti, in particolare con lo scopo di prevenire la saturazione della finestra di contesto dei singoli agenti.

Nel sistema sviluppato non è stato possibile adottare un approccio tradizionale basato su un'unica memoria globale condivisa. Infatti, una condivisione indiscriminata delle informazioni tra gli agenti degraderebbe la qualità delle risposte che essi producono. A livello di progettazione, la soluzione adottata per ovviare a questi problemi è stata quella di imporre a tutti gli agenti del sistema la caratteristica di stateless, cioè nessun agente conserva lo stato interno tra un'invocazione e l'altra. Questo vincolo elimina possibili inconsistenze tra invocazioni parallele e favorisce la scalabilità. Occorre però definire con precisione come le informazioni sono conservate e rese disponibili tra gli agenti. Si è deciso di adottare una strategia ibrida che combina tre livelli nella gestione della memoria: uno stato locale, interno al grafo di ciascun agente, che ne governa il funzionamento. Un altro riguarda il meccanismo di comunicazione tra i nodi del grafo dell'architettura multi-agente, basato sullo scambio di messaggi strutturati (Payload). L'ultima modalità si basa non su uno stato globale condiviso ma su una memoria esterna, ospitata su Amazon S3, che funge da memoria condivisa persistente tra gli agenti e tra i turni della conversazione. E' opportuno precisare che quanto viene trattato in questo capitolo riguarda il funzionamento di tutti gli agenti del sistema, ad eccezione dell'orchestratore. Quest'ultimo, svolge un ruolo centrale nell'architettura complessiva e quindi viene approfondito nel capitolo

apposto. Pertanto, ogni riferimento ad "agenti" in questo capitolo è da intendersi come riferito a tutti gli altri.

5.0.1 Glossario dei termini della memoria

Prima di entrare nei dettagli del funzionamento della memoria, ho riassunto in una tabella i termini tecnici che sono utilizzati in questo capitolo. L'obiettivo è facilitare la comprensione delle sezioni successive.

Termine	Ruolo nella gestione della memoria
Stato interno	Struttura dati in RAM che contiene l'intera cronologia del ciclo ReAct: messaggi, ragionamenti intermedi, log degli strumenti e risultati. Quando l'agente produce la sua risposta finale questo stato viene resettato.
Archivio Interno	Sottoinsieme dello Stato Interno. Contiene esclusivamente i dati strutturati prodotti dai tool (es. liste di POI, risultati di query), senza i ragionamenti intermedi dell'LLM. Al termine dell'esecuzione dell'agente, viene salvato nel Working Storage.
Payload	Pacchetto JSON a struttura fissa scambiato tra nodi. Non è uno stato globale, trasporta esclusivamente informazioni e i metadati necessari al nodo ricevente per inizializzare il proprio stato interno.
Amazon S3	Infrastruttura di memoria esterna e condivisa. Sostituisce il passaggio diretto di dati grezzi tra agenti, rendendo accessibili le informazioni tramite ID senza saturare RAM o context window.
Working Storage	Archivio su S3 contenente gli Archivi Interni prodotti dagli agenti invocati. Cioè, al termine dell'esecuzione degli agenti essi salvano i propri Archivi Interni (non la context window dell'agente in quanto conterrebbe ragionamenti intermedi, retry e informazioni ridondanti non utili).
Storage ID o ID	Identificativo univoco restituito da Amazon S3 al momento del salvataggio di un file. Permette a qualsiasi agente che lo possiede di recuperare esattamente quel file senza dover conoscerne il contenuto a priori.
Geometry storage	Poligoni e coordinate dell'area geografica di interesse, salvati su S3 dall'orchestratore. L'ID viene passato sempre nel payload tra gli agenti, in modo tale che ogni agente possa scaricare esattamente la zona geografica interessata.
Focus Geometry	Geometria spaziale di minore dimensione rispetto alla Geometry storage, creata da un agente specializzato a partire dai propri dati di dominio. Viene salvata su S3 e referenziata tramite ID nel payload.
Turno	Corrispondente a un ciclo completo richiesta-risposta. Contiene la query, la risposta consegnata all'utente e i puntatori agli ID del working storage prodotti in quella sessione.
Registro conversazionale	File JSON su S3 gestito esclusivamente dall'orchestratore. All'inizio di ogni turno viene scaricato in memoria, aggiornato con le nuove informazioni e riscritto su S3 al termine. Funge da memoria a lungo termine del sistema, cioè conserva domande, risposte passate date dal sistema all'utente e gli ID di riferimento di ogni turno.

Tabella 5.1: Riepilogo dei termini chiave nella gestione della memoria del sistema multi-agente.

5.0.2 I componenti della memoria

Stato interno di un agente

In questo progetto, ogni nodo del grafo principale definito con LangGraph è a sua volta un agente che segue una logica di tipo ReAct. Internamente, ciascun agente ha una sua architettura, le cui componenti sono le tre fasi del ciclo ReAct: ragionamento, azione e osservazione. Durante l'esecuzione dell'agente, esso mantiene uno stato locale che viene aggiornato a ogni iterazione del ciclo.. In questo modo, lo stato accumula progressivamente sia i risultati parziali del ragionamento sia gli output delle chiamate agli strumenti, fino a quando l'agente non converge su una risposta finale. Questo stato è del tutto invisibile all'esterno del nodo e viene distrutto al termine dell'invocazione, seguendo la natura stateless del sistema.

Archivio Interno

L'Archivio Interno non raccoglie la cronologia dei messaggi o i passaggi logici intermedi del modello linguistico, ma funge da contenitore esclusivo per i dati strutturati e pesanti prodotti dai tool (come geometrie spaziali, coordinate o lunghe risposte da API). L'utilizzo di questo archivio permette al sistema di disaccoppiare la logica di ragionamento dell'LLM dalla gestione dei dati. L'archivio è organizzato come una struttura chiave-valore: la chiave consiste in una stringa testuale univoca (si precisa che questo identificativo è diverso dall'ID trattato finora: quest'ultimo riguarda l'accesso agli Archivi Interni su S3, mentre gli identificativi interni all'Archivio Interno sono usati dagli strumenti, come spiegato più avanti), mentre il valore associato corrisponde ai dati grezzi salvati.

Quando un tool genera nuovi dati, li registra all'interno di questa struttura e per far sì che l'LLM sia a conoscenza dell'esistenza di questi dati (pur ignorandone il contenuto reale), solo la chiave viene inserita all'interno della finestra di contesto. In questo modo, il modello può decidere di trasmettere le informazioni ad altri tool semplicemente passando la chiave come parametro, questi tool possono così accedere ai dati originari e manipolarli attingendo direttamente dall'Archivio Interno, senza che le informazioni grezze debbano mai transitare nella finestra di contesto dell'LLM, che ne verrebbe altrimenti inquinata. Al termine dell'esecuzione dell'agente, questo archivio viene salvato nel Working Storage su S3, consentendo a eventuali invocazioni future di riprendere e interrogare i dati già elaborati senza doverli calcolare nuovamente.

Memoria esterna come memoria condivisa

Nel sistema sviluppato, la memoria esterna coincide con quella condivisa. Dal punto di vista infrastrutturale, questa architettura utilizza il cloud storage Amazon S3 per archiviare e recuperare le informazioni, evitando così di dover passare una grande quantità di dati tramite Payload (si veda la sezione successiva per comprendere il meccanismo di comunicazione tramite Payload). L'uso di questa memoria si divide in tre aree diverse (salvati in 3 buckets diversi su S3):

- **Working Storage:** Gli agenti elaborano dati geografici e meteorologici estremamente voluminosi (risposte API molto lunghe, archivi in formato GeoParquet, lunghi elenchi di punti di interesse), che se riversati direttamente nella finestra di contesto la saturerebbero in brevissimo tempo. Il Working Storage è uno spazio di memoria su S3 in cui vengono salvati gli Archivi Interni prodotti dagli agenti, cioè, al termine della propria esecuzione, un agente non salva la propria finestra di contesto, ma solo i dati contenuti nel proprio Archivio Interno, poiché la context window può contenere informazioni irrilevanti che non conviene conservare per i turni successivi. Quindi l'agente carica il proprio Archivio Interno su S3 e restituisce all'agente chiamante i corrispondenti ID, attraverso i quali un agente può in un secondo momento scaricare i dati presenti in quel file del Working Storage e includerli nel proprio Archivio Interno. Questo meccanismo garantisce continuità anche in caso di query di follow-up da parte dell'utente: un agente, infatti, può scaricare i dati già trovati nel turno precedente e rispondere senza dover rieseguire gli stessi ragionamenti e chiamate ai tools.
- **Geometry storage:** Una volta che l'orchestratore ha localizzato l'area geografica di interesse per l'utente, salva i poligoni geografici all'interno della memoria condivisa (S3) utilizzando un formato binario (pickle) che ne mantiene intatta la struttura. Questo formato permette di salvare direttamente l'oggetto così come si trova in Python, comprese tutte le sue informazioni geografiche, senza doverlo tradurre in altri linguaggi. Questo processo "congela" gli oggetti geometrici nel loro stato in RAM, evitando le onerose conversioni in formato testo che verrebbero mandate agli altri agenti tramite payload. Inoltre, salvando in formato binario, i dati vengono scritti e letti in modo molto più rapido e senza alcuna perdita di precisione.

Ai sotto-agenti viene inviato esclusivamente un identificativo univoco (ID): questo permette loro di scaricare e "scongela" istantaneamente l'oggetto geometrico esatto. In questo modo si garantisce la coerenza spaziale in tutto il sistema. Dal punto di vista tecnico questa memoria fa parte del Working Storage ma si

è preferito separare le due cose in quanto hanno contenuti molto diversi tra di loro.

- **Registro Conversazionale gestito dall'orchestratore:** Il mantenimento della coerenza tra più interazioni è garantito da un file JSON, definito Registro Conversazionale, che funge da memoria a lungo termine per la conversazione con l'utente. La gestione di questo stato segue un rigoroso ordine di esecuzione a ogni interazione:

1. **Inizializzazione e ricostruzione del contesto:** All'arrivo di una nuova query dall'utente, l'orchestratore viene invocato e scarica dal bucket Amazon S3 l'intero registro conversazionale esistente. L'analisi di questo file permette all'orchestratore di ricostruire il proprio contesto in maniera strutturata ed efficiente. L'operazione di ripristino in memoria avviene per ogni turno precedente avvenuto (inteso come il ciclo d'interazione completo che va dalla richiesta dell'utente alla risposta del sistema). Ciascun turno contiene tre elementi fondamentali: la query, la risposta generata e consegnata all'utente e i corrispettivi puntatori univoci (ID) per l'eventuale recupero dei dati archiviati e l'ID specifico alla geometria,
2. **Esecuzione del Turno:** L'Orchestratore analizza la nuova richiesta considerando lo storico appena ripristinato e delega l'elaborazione all'agente generalista competente.
3. **Aggiornamento e salvataggio del registro:** L'Orchestratore è pronto a includere nel Registro Conversazionale un nuovo Turno a quelli già presenti. Infatti si includono i seguenti dati: la risposta appena generata, gli Storage ID dei nuovi dati prodotti nel Working Storage e l'ID della geometria spaziale trovata. Il Registro Conversazionale viene quindi aggiornato con il nuovo Turno sovrascrivendo il file precedente su S3,

Come si è appena visto l'intero Registro Conversazionale deve essere re-inviato al termine di ogni ciclo. Ma poichè questo contiene solo domande-risposta relativamente brevi e gli ID di riferimento e (anziché i pesanti dati grezzi), il file si mantiene nell'ordine di pochi Kilobyte, rendendo la sovrascrittura pressoché istantanea. Questo meccanismo garantisce all'orchestratore di disporre sempre dell'intero storico necessario, pur funzionando in un'architettura completamente stateless.

La comunicazione nel multi-agente: Payload

L'elemento che peremtte il coordinamento dell'architettura multi-agente è il meccanismo di passaggio dei dati tra i nodi. Per questo progetto non esiste uno "stato

globale” nel senso comune di LangGraph (ovvero un oggetto che percorre l’intero grafo accumulando informazioni ad ogni nodo). Adottare un simile approccio vanificherebbe i vantaggi della natura stateless del sistema, poiché ogni agente si troverebbe a ricevere e dover gestire l’intero storico delle elaborazioni precedenti.

Il sistema adotta invece un’architettura basata sullo scambio di messaggi: i nodi comunicano passandosi esclusivamente un payload JSON con una struttura fissa. Quando perciò un agente padre invoca un sotto-agente, gli trasmette questo pacchetto. L’agente ricevente estrae le informazioni e le utilizza per costruire il contesto iniziale su cui il proprio LLM ragiona.

Si noti come il payload ricevuto non coincida con lo stato interno descritto precedentemente, il payload è soltanto il mezzo per l’inizializzazione del contesto, mentre ciò che l’LLM produrrà durante l’elaborazione farà parte della memoria interna dell’agente.

Questo meccanismo permette agli agenti invocati di essere totalmente inconsapevoli della conversazione che avviene tra l’utente e l’orchestratore, questi operano esclusivamente sui dati ricevuti nel payload. Questa caratteristica evita che la context window dell’agente chiamato contenga informazioni non rilevanti per il loro dominio di competenza.

Il pacchetto di dati trasmesso nel payload dal padre al figlio ha questa struttura:

- **Identificativo della sessione:** un codice univoco associato alla singola conversazione. Non cambia tra un turno e l’altro della chat e indica in quale partizione di S3 salvare i dati.
- **Identificativo utente:** necessario per ottenere le caratteristiche dell’utente e preferenze.
- **Identificativo della geometria spaziale:** l’ID della geometria di riferimento, affinché l’agente possa scaricare da S3 solo l’esatta zona geografica di interesse.
- **Identificativo della Focus Geometry (opzionale):** alcuni agenti specializzati possono allegare un secondo ID che indirizza a una geometria più specifica rispetto a quella generale dell’orchestratore.
- **Identificativi dei turni precedenti:** a partire dal secondo turno di conversazione, l’orchestratore recupera dal Registro Conversazionale gli identificativi degli archivi generati nei turni antecedenti. Questi vengono organizzati in una struttura a coppie chiave-valore, in cui ogni chiave rappresenta il nome specifico di un agente e il rispettivo valore contiene la lista dei codici identificativi che quell’agente ha salvato su S3.

L’obiettivo fondamentale di questo meccanismo è consentire una comunicazione efficiente e persistente tra turni diversi, salvaguardando la natura stateless del

sistema. Quando un agente viene invocato in un turno successivo e deve ricollegarsi a quanto elaborato in precedenza, il suo LLM è consapevole di questi identificativi e può usare i propri tool per scaricare da S3 solo i file specifici richiesti per la nuova elaborazione. In questo modo si evita il ricalcolo di dati complessi e si riduce al minimo l'occupazione della context window dell'agente.

- **Query del nodo padre:** l'istruzione testuale che il figlio dovrà eseguire. Questa non coincide con la richiesta originale dell'utente. Il padre, prima di invocare il sotto-agente, elabora la query da mandare al sotto-agente in base alle proprie conoscenze e alle informazioni già in suo possesso. Il figlio riceve quindi una richiesta specifica e contestualizzata nel suo ambito di competenza, passando-gli anche le preferenze generali dell'utente o le elaborazioni di altri agenti se necessario.

Si noti che la richiesta formulata dal padre deve essere quanto più possibile mirata, proprio perché il padre non conosce il contenuto degli Archivi Interni del figlio.

Il Payload di ritorno

Quando un agente figlio completa la sua esecuzione, restituisce al padre un payload composto da tre elementi: la risposta testuale, i metadata di sessione e la lista degli Storage ID prodotti durante l'elaborazione. Al ricevimento del payload, il padre integra la risposta testuale del figlio nella propria context window, per proseguire il ragionamento, mentre la lista degli Storage ID confluisce nel proprio stato interno per poi essere pronti a essere inclusi nel successivo payload di ritorno.

Il payload di ritorno contiene l'ID generato dall'agente figlio, ma include anche quelli prodotti dai suoi sotto-agenti(dato che il padre non ha visibilità diretta su cosa sia avvenuto al suo interno né su quali sotto-agenti siano stati eventualmente invocati). L'utilizzo del metodo payload permette di risalire la gerarchia del sistema portando con sé sia gli ID propri che quelli ereditati dai livelli sottostante. Quindi quando l'Orchestratore è pronto a consegnare la risposta all'utente ha anche con sé la lista completa di tutti gli Storage ID prodotti nell'intero sistema (in quel turno), che viene poi salvata nel Registro Conversazionale.

5.0.3 Vantaggi dello Storage S3 rispetto al solo utilizzo del Payload

Nella comunicazione tra agenti, l'utilizzo di uno storage esterno (Amazon S3) in alternativa al passaggio diretto dei dati nel payload offre i seguenti vantaggi:

- **Gestione di dati di grande dimensione:** Gli oggetti geospaziali, come poligoni amministrativi o punti di interesse, esistono in memoria come strutture

dati binarie, cioè un formato compatto ed efficiente, ma incompatibile con il formato JSON che il payload utilizzerebbe. Per includerli nel payload sarebbe necessario serializzarli, ovvero convertirli in una lunga sequenza di testo, per poi deserializzarli all'arrivo, ricostruendo l'oggetto originale. Questo doppio passaggio introduce costo computazionale non trascurabile e appesantisce il payload. Salvando invece l'oggetto su S3 nel suo formato binario l'agente ricevente lo ripristina direttamente in memoria con una singola operazione di lettura, senza alcuna conversione intermedia.

- **Riprendere una conversazione passata:** archiviare i dati su uno storage permanente consente all'utente di poter riprendere una conversazione interrotta con il sistema. Infatti, tutti i risultati prodotti dal sistema in una sessione rimangono salvati su S3 pronti per essere ripescati e utilizzati dal sistema. Cosa non da poco se si considera che l'intera architettura ha agenti stateless.

5.0.4 Altre scelte progettuali e precisazioni

E' necessario precisare tre aspetti cruciali legati al ciclo di vita degli agenti:

- **Persistenza tra turni:** Lo stato interno di un agente rimane lo stesso per tutta la durata del suo task, cioè quando il padre invoca un sotto-agente, il padre non si spegne ma viene messo in pausa, conservando intatto il proprio stato locale in RAM. Al ritorno del payload di risposta, esso si "risveglia" per continuare il suo task.
- **Gestione dello stato:** Data la natura intrinsecamente stateless degli LLM, il framework si fa carico di mantenere lo stato della sessione. Esso aggiorna e ricostruisce la context window a ogni iterazione, gestendo anche il ciclo di interrogazione del modello e dei relativi tool fino alla generazione della risposta finale.
- **Struttura Hub-and-Spoke.** Il flusso di comunicazione tra gli agenti e i loro figli segue un modello a stella (Hub-and-Spoke), ovvero il padre ha il ruolo di nodo centrale e i suoi sotto-agenti non comunicano mai direttamente tra loro. Questo architettura è stata scelta perché semplifica il coordinamento: il padre è l'unico responsabile di aggregare i risultati e decidere i passi successivi, evitando dipendenze dirette tra sotto-agenti. Se ogni sotto-agente comunicasse con gli altri, aggiungere un nuovo agente richiederebbe di aggiornare tutti quelli con cui interagisce. L'isolamento tra i sotto-agenti garantisce che un eventuale errore in uno di essi non si propaghi agli altri. Inoltre, questa struttura permette il funzionamento di agenti in parallelo.

5.0.5 Conversazioni multi-turno e gestione dei follow-up

Affinché il sistema risulti utile e interattivo, deve poter sostenere conversazioni logicamente coerenti in cui le risposte si basano sull'intero contesto della conversazione. Il comportamento che il sistema assume dipende dal Turno in cui ci si trova. Se l'utente chiede per la prima volta al sistema si è nel primo Turno e conseguentemente il registro Conversazionale è vuoto e non esiste ancora alcuna informazione geografica salvata. Quindi l'Orchestratore deve occuparsi della localizzare geografica. Nei turni successivi, invece, il registro viene scaricato da S3 e contiene l'intera storia della conversazione (domande, risposte e metadati) insieme all'ID della geometria spaziale e agli ID del Working storage prodotti in precedenza. L'orchestratore può così ripartire da dove ci si era fermato, senza chiedere all'utente di ripetere il contesto.

Al termine di ogni turno, l'orchestratore è l'unico agente autorizzato a salvare (o sovrascrivere se è già presente) su S3 il registro conversazionale. Dopodiché anche l'orchestratore si spegne, lasciando su S3 tutto il necessario per riprendere la conversazione al turno successivo.

5.0.6 Vantaggi di questo meccanismo per la memoria

L'architettura di memoria ibrida descritta in questo capitolo offre diversi vantaggi rispetto a soluzioni più tradizionali. Il primo e più immediato riguarda l'efficienza nell'uso dei token: separando nettamente i dati grezzi salvati nell'Internal Storage e archiviati su S3 e usando solo gli ID) dal contesto testuale iniettato nei prompt, si limita drasticamente la quantità di token consumati a ogni invocazione.

Un secondo vantaggio riguarda la scalabilità: la natura stateless degli agenti consente di eseguire più invocazioni in parallelo e permette a ogni agente ricevere esclusivamente le informazioni pertinenti al proprio dominio di competenza, evitando che dati irrilevanti inquinino il ragionamento del modello linguistico. Riducendo quindi le probabilità di produrre allucinazioni o errori di ragionamento.

5.0.7 Svantaggi di questo meccanismo per la memoria

Questa architettura presenta una complessità di implementazione e manutenzione considerevole, che rappresenta il suo limite più significativo. La molteplicità di componenti coinvolti, tra stato interno, payload, working storage, registro conversazionale e la loro interazione tramite ID, rende il sistema difficile da debuggare. In caso di comportamento anomalo, individuare quale componente sia responsabile del malfunzionamento richiede un'analisi approfondita dell'intera catena di esecuzione. Questa complessità riguarda anche la manutenzione nel tempo poiché qualsiasi modifica strutturale a uno dei componenti rischia di propagare effetti inattesi sugli altri. Un secondo svantaggio riguarda la latenza introdotta dalle operazioni compiute su

S3: ogni agente deve scaricare i dati necessari all’inizio della propria esecuzione e caricare i dati al termine. Sebbene queste operazioni siano generalmente rapide, con file di grandi dimensioni possono diventare un collo di bottiglia non trascurabile. Vi è poi una dipendenza forte nei confronti di Amazon S3: un’interruzione del servizio renderebbe il sistema completamente non operativo, poiché la memoria condivisa è l’unico meccanismo che garantisce la coerenza tra gli agenti e la continuità conversazionale. Questa dipendenza implica inoltre la necessità di gestire con attenzione i costi di storage e di trasferimento dati, che in un sistema con molte sessioni attive simultaneamente potrebbero crescere in modo significativo.

Capitolo 6

L'Agente Orchestratore

L'agente orchestratore rappresenta la porta d'ingresso principale del sistema (single entry point) e ha come compito principale quello di indirizzare in modo intelligente le richieste degli utenti verso gli agenti corretti. La sua funzione non si limita però al mero smistamento dei messaggi: prima di delegare la richiesta, l'orchestratore si occupa anche di individuare e risolvere l'area geografica a cui la query fa riferimento, traducendo il riferimento testuale dell'utente in una geometria concreta utilizzabile dagli agenti a valle. Per assolvere a questo duplice compito, l'architettura dell'orchestratore si fonda su due punti fondamentali:

- **La Pipeline di Esecuzione:** Ogni richiesta segue un flusso logico sequenziale in cui l'agente scompone la query ricevuta tramite l'uso di LLM per estrarre l'intento, traduce i riferimenti testuali in specifiche zone geografiche (attraverso il modulo di *Place Resolution*) e infine indirizza la query all'agente di dominio più competente.
- **Gestione del Registro Conversazionale:** L'orchestratore mantiene una memoria persistente su Amazon S3 che permette al sistema di preservare il contesto storico e gestire una vera e propria conversazione multi-turno con l'utente.

Il funzionamento della pipeline e i dettagli tecnici del modulo di Place resolution vengono approfonditi nella sezione 6.0.1, mentre le logiche del Registro Conversazionale sono analizzate nella sottosezione 6.0.2.

6.0.1 La Pipeline a tre fasi

Il primo livello, relativo alla comprensione semantica e all'estrazione dell'intento, segue un flusso logico composto da tre parti sequenziali, ciascuna delle quali produce un output necessario per la successiva.

1. **Parse query:** Il modello linguistico analizza la richiesta dell'utente inserendola all'interno del contesto fornito dal registro conversazionale (multi-turno) e produce una query arricchita (*enriched query*). Quest'ultima è una riformulazione autonoma della richiesta originale che risolve i riferimenti contestuali impliciti, con l'obiettivo di passare esclusivamente essa agli agenti a valle. Una volta definita la *enriched query*, l'LLM genera un oggetto JSON tipizzato che specifica, oltre ai dettagli estratti, l'agente di dominio a cui indirizzare la query stessa (ad esempio, il Travel Agent).
2. **Place Resolution:** Il riferimento geografico estratto viene passato al modulo deterministico di *Place Resolution* (per maggiori dettagli sul suo funzionamento si rimanda alla sottosezione 6.0.1), che lo traduce in una geometria (poligoni amministrativi o isocrone in formato GeoJSON) e nelle relative chiavi di partizionamento (che su S3 agiscono come *directory*) per lo storage. Questa struttura viene salvata nel Working Storage su S3. Ai sotto-agenti a valle viene trasmesso esclusivamente l'identificativo univoco della geometria, consentendo loro di recuperare autonomamente i dati spaziali necessari per rispondere in modo specifico nella zona indicata. In questo modo, i sotto-agenti del sistema possono caricare la geometria nel proprio archivio locale senza iniettarla direttamente nel prompt. Tale approccio previene l'inquinamento e la saturazione della finestra di contesto dell'LLM con coordinate geografiche di scarsa utilità, lasciando che siano i tool dell'agente ad accedere direttamente ai dati spaziali in memoria, non passando mai quindi all'interno della *context window*. Inoltre, questa risoluzione geografica fatta dall'orchestratore permette a tutti gli agenti a valle di operare sulla stessa geometria, senza che essi debbano ricalcolarla ogni volta.
3. **Routing:** L'orchestratore utilizza il suo strumento di routing, che è puramente deterministico, per invocare il sotto-agente di dominio identificato nel primo step della pipeline. L'orchestratore passa a questo sotto-agente il payload contenente la query. L'esatto funzionamento di questo meccanismo viene trattato nel capitolo sulla memoria.

Sebbene questa pipeline possa apparire come una semplice sequenza lineare, l'orchestratore rimane un agente intelligente in quanto l'LLM nella prima fase agisce in modo non deterministico e ha la possibilità di scegliere come interpretare una query e verso cosa indirizzarla.

Dal punto di vista dell'orchestratore, una volta completato il Routing, esso rimane in attesa della risposta da parte del sotto-agente invocato. Quando la risposta è pronta, l'orchestratore la restituisce all'utente senza alcuna rielaborazione. Questa scelta progettuale garantisce una separazione delle responsabilità: l'orchestratore si occupa di interpretare la richiesta, risolvere il contesto geografico e indirizzarla al

dominio corretto, mentre il compito di rispondere rimane interamente a carico dei sotto-agenti. Di conseguenza il sistema multi-agente così creato può gestire domini anche molto diversi tra di loro senza che esso conosca i dettagli specifici di ciascuno di essi.

Place Resolution

All'interno dell'architettura dell'orchestratore, il modulo di *Place Resolution* funge da collegamento tra le informazioni ottenute dall'uso dell'LLM nel Parse query e la geometria necessaria per le analisi spaziali. È da precisare che questo modulo non si basa su logica probabilistica, ma opera interrogando tabelle di riferimento archiviate in formato GeoParquet su Amazon S3, tramite DuckDB.

Questo processo si articola in quattro passaggi:

1. **Pre-caricamento:** Quando il modulo di *Place Resolution* viene inizializzato, si collega a un bucket S3 nel cloud e, tramite DuckDB, carica in memoria due file di riferimento in formato GeoParquet. Il primo file contiene i nomi dei luoghi, i livelli amministrativi (nazione, regione, provincia, comune), le relazioni gerarchiche (es. Torino è all'interno del Piemonte) e i relativi ID univoci. Il secondo file (notevolmente più pesante) contiene invece le geometrie reali (i poligoni) associate a quegli ID.
2. **Validazione gerarchica:** Utilizzando esclusivamente il primo file, si verifica la coerenza della richiesta testuale dell'utente. Ci si assicura, ad esempio, che un comune appartenga effettivamente alla provincia o alla regione indicata, consultando la gerarchia predefinita. Una volta confermata, questa gerarchia amministrativa validata viene salvata per essere utilizzata come metadato al punto 4.
3. **Risoluzione geometrica:** Trovato l'ID esatto della località, viene interrogato il secondo file GeoParquet per recuperare i confini esatti sotto forma di poligono. Tale poligono è definito come una sequenza ordinata di coppie latitudine-longitudine in coordinate WGS84 (il sistema di riferimento geografico globale usato anche dal GPS), dove il primo e l'ultimo punto coincidono per chiudere la forma. Il risultato finale (geometria unita ai metadati) viene formattato in GeoJSON. Questa struttura standardizzata è l'unico payload spaziale che si sposta tra i vari agenti per garantire che tutti siano perfettamente allineati sull'area geografica in questione: nessun agente riceverà mai del semplice testo ambiguo come "Val d'Aosta", ma sempre e solo il poligono dei suoi confini. È da sottolineare che il file GeoParquet di grandi dimensioni viene scaricato solo una volta dall'orchestratore per l'intero sistema, mentre i sotto-agenti potranno recuperare il poligono con dimensione notevolmente ridotta.

4. **Come si ottengono le chiavi di partizionamento (directory):** Il poligono ottenuto nella fase precedente descrive l'area di interesse, ma non dice ancora dove si trovano fisicamente i dati corrispondenti nel Data Lake. Essendo quest'ultimo organizzato su S3 con una struttura a cartelle basata sullo schema Hive, i percorsi fisici dei dati (le directory) coincidono con le chiavi di partizionamento, definite fino al livello regionale (per esempio `country=IT/admin_level_1=toscana`). I livelli amministrativi provincia e comune, non generano cartelle separate ma restano semplici colonne all'interno dei file regionali. Il modulo di *Place Resolution* dispone di una tabella di riferimento che associa a ciascuna regione i relativi percorsi su S3. Confrontando il poligono dell'area richiesta con questa tabella, ricavando così la lista delle directory (o chiavi di partizionamento) da interrogare. In questo modo gli agenti a valle possono aprire direttamente le sole partizioni pertinenti, riducendo drasticamente latenza e costi computazionali.

L'output della *Place Resolution* è un oggetto strutturato con tre componenti distinti, ciascuno con un ruolo preciso. La geometria in formato GeoJSON è il poligono dell'area di interesse: non serve solo a identificarla, ma viene usata dagli agenti a valle per filtrare con precisione i dati al suo interno, per esempio estraendo solo i sentieri che ricadono effettivamente dentro i confini richiesti. Le chiavi di partizionamento (directory) sono le stringhe calcolate al punto precedente, per esempio `country=IT/admin_level_1=toscana`. Su S3, infatti, i dati sono distribuiti in una struttura gerarchica dove ogni cartella definisce un partizionamento geografico. Queste chiavi indicano agli agenti esattamente quali percorsi aprire, evitando di scansionare l'intero archivio. I metadati amministrativi, ricavati durante la validazione gerarchica, sono i nomi di regione, provincia o città. Si includono perché alcuni servizi esterni, come le API meteo o i servizi di ricerca punti di interesse, non accettano un poligono in input ma si aspettano riferimenti testuali. Per i dettagli su come la geometria viene salvata e trasmessa all'interno del sistema multi-agente si veda la voce "Geometry storage" nella sottosezione 5.0.2 del capitolo 5.

6.0.2 Gestione del multi-turno

L'orchestratore è progettato per supportare conversazioni multi-turno, ovvero sequenze di query correlate in cui ciascuna richiesta può fare riferimento a informazioni presentate nei turni precedenti. Questo rappresenta un problema architetturale non banale: il sistema multi-agente è progettato in modo da essere stateless. Ciò significa che ogni invocazione (sia dell'orchestratore sia degli altri agenti) è un'esecuzione indipendente e nessun componente mantiene la propria memoria.

Per superare questa limitazione, l'orchestratore delega il mantenimento dello stato a un Registro Conversazionale, un documento JSON salvato su Amazon S3. Questo

file viene scaricato all'inizio di ogni invocazione dell'orchestratore e sovrascritto da esso al termine del turno con i nuovi aggiornamenti (al primo turno il file viene inizializzato vuoto). Il contenuto del registro viene letto e iniettato nella finestra di contesto dell'LLM dell'orchestratore all'inizio di ogni nuova esecuzione. In questo modo, l'orchestratore può “leggere” la cronologia della conversazione e usarla per comprendere nuove domande dell'utente.

Questo file contiene una serie di metadati globali e un singolo array cronologico che rappresenta lo storico dei turni. Ogni elemento di questo array è un oggetto strutturato (un dizionario) che raggruppa tutte le informazioni relative a un singolo turno tra utente e sistema. Nello specifico ogni oggetto contiene le seguenti informazioni:

- **Query Originale e Query Arricchita:** Per ogni turno viene salvata la domanda originale dell'utente e la sua versione esplicita generata dall'orchestratore. Mantenere nello storico le query arricchite passate è fondamentale, infatti, se l'utente fa una domanda ambigua al Turno 2 (es. “quanto è lungo?”), l'orchestratore legge la query arricchita del Turno 1 (es. “Cerca il sentiero T123 in Piemonte”) e usa questa informazione per formulare la nuova query arricchita corretta per il Turno 2 (“Quanto è lungo il sentiero T123 in Piemonte?”).
- **ID della Geometria:** Un codice univoco calcolato dal *PlaceResolver* che identifica l'area geografica (es. una regione o un comune). Se in un turno successivo l'utente fa una domanda sulla stessa area (es. “e quali attività ci sono lì?”), l'orchestratore recupera la geometria tramite questo ID senza dover ripetere il processo di localizzazione.
- **Puntatori S3 ai Dati Strutturati (Output Storage IDs):** Un dizionario in cui si associa il nome dell'agente invocato (es. Weather Agent) alla lista degli identificativi univoci S3 che puntano ai dati da esso generati. Ogni volta che un agente termina un'elaborazione, infatti, salva il proprio stato e i risultati strutturati in un file JSON (l'Archivio Interno) su S3. Salvando esclusivamente questi codici identificativi nel Registro Conversazionale, l'orchestratore può trasmetterli come parametri agli agenti nei turni successivi. In questo modo gli agenti ricevono unicamente gli “indirizzi” per ritrovare i propri dati passati, scaricandoli da S3 solo se la nuova richiesta dell'utente (domanda di *follow-up*) richiede effettivamente la consultazione di vecchie elaborazioni, evitando così di scaricare dati inutilmente e di saturare la memoria globale.
- **Risposta Integrale all'Utente (Results Summary):** La risposta testuale finale fornita dal sistema all'utente nel rispettivo turno, rigorosamente salvata per intero e senza subire alcun processo di sintesi o riassunto (*as-is*).

Questa architettura definisce una netta separazione tra la “memoria globale” dell'orchestratore e la “memoria locale” dei sotto-agenti. È importante precisare che

le finestre di contesto vere e proprie dei sotto-agenti (ovvero lo storico dei messaggi con l'LLM, le interrogazioni ai database e i ragionamenti interni) non vengono mai salvate su S3: si tratta di dati volatili che si azzerano al termine di ogni esecuzione. Ciò che i sotto-agenti salvano stabilmente su S3 è unicamente il loro stato strutturato (il proprio Archivio Interno), ovvero i dati elaborati dai vari tools (come strutture JSON, array di coordinate o liste di attività).

Se l'orchestratore dovesse inglobare all'interno del proprio registro conversazionale l'intero peso di tutti questi dati strutturati generati a valle da ciascun agente, il file JSON diventerebbe enorme. Di conseguenza, la finestra di contesto dell'orchestratore si saturerebbe rapidamente di informazioni di dominio irrilevanti per il *routing*. Per questo motivo, vengono salvati all'interno di questo file globale solo i puntatori a questi archivi S3 (i già citati *Output Storage IDs*), insieme alla geometria, alle query e alla risposta finale. Questa scelta progettuale assicura che il registro conversazionale rimanga estremamente leggero e veloce da ricaricare ad ogni turno.

Infine, il salvataggio integrale della risposta all'utente, unito allo storico delle query, è essenziale affinché l'LLM dell'orchestratore sia capace, nei turni successivi, di ricostruire la conversazione esattamente così come l'ha letta e vissuta l'utente, permettendogli di formulare una *enriched query* coerente con il contesto reale. Nello specifico, lo storico completo viene ricostruito dal Registro Conversazionale prima di ogni invocazione dell'orchestratore iterando su tutti i turni dell'array. Ogni turno viene formattato e inserito nel prompt specificando per ognuno: la query dell'utente, l'agente che è stato chiamato a risolvere il task, l'ID della geometria, i rispettivi *Output Storage IDs* e la risposta testuale in formato integrale. A questo punto l'LLM dell'orchestratore possiede il contesto puntuale e inalterato di tutto ciò che è stato fatto e scritto precedentemente dall'intero sistema, potendo così gestire alla perfezione le complesse dinamiche e le ambiguità tipiche delle domande di *follow-up*.

6.0.3 Rapporto con gli altri agenti

L'orchestratore esercita un controllo gerarchico diretto sui generalisti (Travel Agent e Retail Agent), decidendo in autonomia quale attivare in base all'intento estratto dalla query.

Per quanto riguarda gli altri agenti del sistema, all'interno dei capitoli dedicati alla loro descrizione non sarà necessario specificare come questi ottengano il riferimento geografico, infatti, l'orchestratore salva la geometria su S3 nel Working Storagee e include nel payload l'ID corrispondente. In questo modo ogni agente possa così scaricare autonomamente l'area geografica di interesse senza che questa debba essere ricalcolata o trasferita direttamente nella context window.

Capitolo 7

Travel Agent

7.1 Introduzione

Il Travel Agent è l'agente generalista incaricato di gestire le richieste nel dominio turistico. Viene invocato direttamente dall'orchestratore e ha il compito di assistere alle richieste dell'utente e supervisionare la pianificazione logistica, affidandosi ai suoi sotto-agenti quando necessario. Il suo ruolo fondamentale è quindi quello di comprensione e delegazione, non si limita a inoltrare la query originale, ma scompone task complessi o ambigui in sotto-problemi, consentendo agli agenti invocati di eseguire al meglio il compito loro assegnato. Il Travel Agent non coordina solamente agenti specialisti di dominio, l'Activity Agent e l'Hiking Agent, ma è in grado di interpellare direttamente anche i Data Expert, cioè il Location Agent e il Weather Agent.

7.2 Strumenti a disposizione

Questo agente pur avendo principalmente una prospettiva ad alto livello, esso ha comunque degli strumenti. Questi, però, come si è detto sono volontariamente semplici e concepiti come supporto alla pianificazione turistica.

- **Accesso ai dati utente**

Questo strumento consente all'agente di accedere a un dataset memorizzato in S3 contenente le caratteristiche degli utenti. Il dataset include sia informazioni anagrafiche sia attributi descrittivi, come le preferenze di viaggio (ad esempio, relax o avventura), le preferenze culinarie, il livello di esperienza e altre caratteristiche rilevanti per la personalizzazione.

Una volta recuperate queste informazioni, l'agente può invocare uno dei propri sotto-agenti, fornendogli non solo la query dell'utente da elaborare, ma anche il

profilo utente ottenuto tramite questo strumento. Così i sotto-agenti affrontano il problema tenendo conto delle caratteristiche e delle preferenze di chi ha fatto la richiesta, e i consigli finali risultano più su misura.

- **Convertitore di Valuta** Questo strumento è progettato per fornire stime istantanee dei tassi di cambio. A livello implementativo, la funzione accetta in input un importo numerico, la valuta di partenza (es. 'EUR') e la valuta di destinazione (es. 'USD'). Il tool utilizza un'API pubblica e gratuita fornita dal servizio ExchangeRate-API [12], al fine di ottenere i tassi di cambio aggiornati in tempo reale.
- **Convertitore di fuso orario** Questo strumento calcola l'ora corrente in due località e la differenza oraria tra esse, accettando come input i nomi dei fusi orari (ad esempio Europe/Rome o America/New_York). Il risultato è un messaggio leggibile che indica l'orario effettivo nei due luoghi e il numero di ore che li separa.
- **Ricerca delle festività pubbliche** Questo strumento consente di ottenere l'elenco delle festività ufficiali di un paese per un dato anno, richiedendo come parametri l'anno di interesse e il codice paese nel formato ISO 3166-1 alpha-2 (ad esempio IT o US). Le informazioni restituite includono la data, il nome della festività e l'eventuale applicazione su scala nazionale o regionale.
- **Scarica profilo utente** Permette di scaricare da S3 le caratteristiche dell'utente e le sue preferenze. In modo da poter passare ai sotto-agenti delle istruzioni più dettagliate.

7.3 Rapporto con gli altri agenti

Viene ora illustrato come il Travel Agent si interfaccia con gli altri agenti del sistema:

- **Orchestratore:** L'Orchestratore riceve le query iniziale direttamente dall'utente e quando identifica un intento legato a: viaggi, vacanze, attività o luoghi di interesse, invoca il Travel Agent. Oltre a passargli la richiesta, l'Orchestratore fornisce al Travel Agent un ID che permette al Travel Agent di recuperare l'area geografica di partenza su cui basare l'intera pianificazione.
- **Hiking Agent:** Il Travel Agent lo invoca quando rileva che l'utente desidera fare attività all'aria aperta, trekking o percorsi in montagna. Consente anche di valutare se le condizioni meteo sono sicure per queste attività.
- **Activity Agent:** Viene invocato dal Travel Agent per decidere "cosa fare", che sia visitare un luogo o raggiungere un punto di interesse. Viene chiamato per

gestire la fase organizzativa per strutturare l'esperienza turistica (ad esempio, turismo urbano, tempo libero, visite culturali). A differenza del Location Agent che esegue puri calcoli spaziali, l'Activity Agent é colui che sceglie le attività vere e proprie e ha anche la possibilità di creare un itinerario giornaliero.

- **Location Agent:** Viene invocato dal Travel Agent per individuare punti di interesse all'interno dell'area geografica di riferimento (il poligono ottenuto dall'orchestratore), ma anche per ricerche più mirate: nei pressi di un punto specifico, all'interno della zona raggiungibile in un dato tempo (isocrona) oppure lungo un percorso.
- **Weather Agent:** viene interrogato dal Travel Agent per ottenere dati meteorologici aggiornati, così da poter arricchire le risposte fornite all'utente con suggerimenti sull'abbigliamento più adatto e con eventuali avvisi relativi ad allerte meteo.

Capitolo 8

L'Activity Agent

L'Activity Agent appartiene alla categoria degli agenti specialisti e si occupa sia dell'individuazione delle tipologie di attività più adatte alle richieste dell'utente, sia della ricerca di luoghi concreti in cui svolgerle, quando questa ricerca riguarda tipologie di attività coperte dai propri dataset. Qualora invece la richiesta dell'utente riguardi l'individuazione di un punto di interesse generico, non riconducibile a una specifica categoria di attività gestita dall'agente, la ricerca viene delegata al Location Agent, responsabile dell'identificazione dei punti di interesse.

Per migliorare la qualità dei suggerimenti, l'agente collabora inoltre con il Weather Agent, valutando le condizioni meteorologiche previste e verificando che l'attività proposta sia compatibile con il meteo atteso, così da poter scartare, quando necessario, le opzioni all'aperto in presenza di condizioni avverse.

8.1 Gestione dei dati: due dataset per due scopi differenti

L'agente si appoggia su due archivi di dati distinti, costruiti per rispondere a due esigenze diverse: uno pensato per la copertura ampia del territorio, l'altro per la comprensione specifica di ciò che si può fare in un singolo luogo.

Il primo è un dataset precalcolato, ottenuto da OpenStreetMap, che copre l'intero territorio su cui il sistema è in grado di operare. Il suo scopo è offrire un'assistenza di carattere generale, garantendo che l'agente sappia rispondere qualunque sia la zona indicata dall'utente. E' l'archivio a cui si attinge, ad esempio, quando si vogliono individuare gli spot dove praticare una certa attività in una regione o quelli più vicini a un punto di partenza. Le informazioni che contiene sono di natura essenzialmente amministrativa e fattuale: nome dell'attività, coordinate geografiche, grado di difficoltà, quota, orari di apertura e sito web. Sono dati che dicono dove si trova un luogo

e cosa offre sul piano pratico, ma non raccontano che tipo di esperienza attende chi lo visita.

Questi dati vengono estratti da OpenStreetMap e poi filtrati, arricchiti e organizzati in file di formato GeoParquet. Proprio perché l'archivio deve coprire un'area così vasta, i file sono suddivisi in cartelle su due livelli, lo Stato e la regione amministrativa di appartenenza, questa partizione permette di mantenere tempi di risposta rapidi anche su una base dati di grandi dimensioni, caricando di volta in volta solo la porzione di territorio che interessa. La dimensione dell'archivio è piccolo abbastanza da copiarlo per intero all'interno dell'ambiente dell'agente, il suo container, al momento del deploy. In questo modo l'agente ha sempre i dati pronti all'uso e non deve interrogare server esterni per reperire le informazioni sui luoghi.

Il secondo dataset è stato ottenuto tramite scraping della piattaforma di prenotazione turistica GetYourGuide. A differenza del primo, non si limita a descrivere i luoghi ma ne racconta l'esperienza: da ciascuna elemento si estraggono un titolo, un abstract, una descrizione estesa, l'elenco di ciò che è incluso ed escluso e i dati di gradimento, ovvero la valutazione media e il numero di recensioni ricevute. A questi campi, ottenuti direttamente dallo scraping, se ne affianca un altro ottenuto da una classificazione che definisce le esperienze come: al chiuso, all'aperto o riparata/mista. Tale etichetta viene assegnata da un modello linguistico che legge la descrizione dell'esperienza; il risultato è memorizzato tra i metadati di ogni elemento del dataset. Come si vedrà, è questa etichetta a permettere di tenere conto del meteo in modo deterministico durante la pianificazione. E' invece la componente descrittiva a rendere il dataset adatto al confronto semantico e quindi alla pianificazione degli itinerari. Va infine precisato che questo dataset al momento comprende solo l'area di Cancún, allo scopo di verificare la validità dell'algoritmo di ottimizzazione su una porzione di area circoscritta.

Generazione degli embedding

Come si vedrà nella sezione dedicata alla pianificazione degli itinerari, la selezione delle esperienze da proporre avviene confrontando tra loro dei testi: la descrizione di ciascuna esperienza da un lato, e dall'altro un testo che rappresenta ciò che si sta cercando, cioè il profilo dell'utente. Un confronto di questo tipo non può essere eseguito direttamente sul testo scritto normalmente. Non si è in grado di determinare direttamente quanto due frasi siano simili dal punto di vista del significato semplicemente analizzandone le parole. Per superare questo limite si utilizzano gli embedding, che rappresentano il testo sotto forma di vettori numerici. Questi vettori sono costruiti in modo tale che testi semanticamente simili siano vicine nello spazio vettoriale. In questo modo, la vicinanza di significato tra due testi può essere quantificata calcolando la similarità tra i rispettivi vettori.

Per ogni voce del secondo dataset viene composto un unico testo descrittivo che unisce titolo, tipologia, abstract e descrizione dell'esperienza, l'elenco di ciò che è incluso ed escluso (la valutazione media e il numero di recensioni ricevute non sono incluse in quanto causerebbero solo rumore nella trasformazione con l'embedding). Il testo risultante viene inviato al modello Titan Embeddings V2, che restituisce un vettore a 1024 dimensioni. I vettori sono quindi normalizzati, in modo tale che la similarità semantica tra due testi si riduce a un semplice prodotto scalare tra i rispettivi vettori.

In pratica, il procedimento si svolge in due momenti distinti. Il primo avviene una sola volta, offline, utilizzando il secondo dataset: per ogni esperienza si costruisce il testo descrittivo, lo si invia al modello di embedding e si ottiene il vettore corrispondente. I vettori così prodotti vengono impilati in una matrice, in cui la riga i -esima è l'embedding dell'esperienza i -esima componente del secondo dataset, e questa matrice viene salvata su file accanto al dataset stesso. Da questo momento gli embedding delle esperienze non vengono più ricalcolati: quando l'agente viene avviato, la matrice è semplicemente caricata in memoria insieme ai dati.

Il secondo momento avviene a tempo di esecuzione, a ogni richiesta. L'unico embedding che manca è quello che rappresenta l'utente tutte le sue caratteristiche vengo trasformate in frasi in linguaggio naturale (ad esempio, un punteggio alto sulla dimensione culturale diventa "apprezza molto le attività culturali"), il testo così ottenuto viene inviato allo stesso modello di embedding usato nella prima fase e se ne ricava un vettore della stessa dimensione. Essendo tutti i vettori normalizzati, il prodotto scalare tra il vettore dell'utente e ciascuna riga della matrice restituisce, in un'unica operazione, un punteggio di affinità tra l'utente e ogni esperienza del dataset. E' questo insieme di punteggi che i filtri semantici dello strumento della pianificazione ottimizzata utilizza per ordinare le esperienze, tenendo le più pertinenti.

La pipeline descritta è generica: può essere applicata a qualsiasi insieme di attività, purché ciascuna sia sufficientemente descritta in linguaggio naturale e completa. Non è quindi legata ad alcun luogo geografico specifico. Questa generalità, però, non si traduce in una copertura automatica dell'intero territorio: la generazione degli embedding ha un costo, poiché richiede una chiamata al modello per ciascuna esperienza e tale costo cresce con il numero di attività da elaborare. Estendere la funzionalità a un'area molto vasta comporterebbe quindi di precalcolare e conservare un numero elevatissimo di embedding. Di conseguenza, la pianificazione degli itinerari non è pensata per essere disponibile ovunque, ma solo per un insieme circoscritto di destinazioni, selezionate in anticipo, per le quali il dataset descrittivo e i relativi embedding vengono predisposti.

In questa tesi il secondo dataset è stato costruito appositamente solo per l'area di Cancún. L'obiettivo infatti non era quello di coprire un territorio ampio, ma verificare che l'algoritmo di ottimizzazione funzionasse bene su una zona ristretta e

controllata.

8.2 Strumenti

8.2.1 Ricerca Attività per Area Amministrativa

Interroga il dataset delle attività relativo alla regione identificata dai metadati geografici e restituisce le attività disponibili in quella porzione di territorio. L'utente può specificare il tipo di esperienza desiderata, per esempio arrampicata, surf, sci o ciclismo. Lo strumento è utilizzato quando si desidera identificare luoghi adatti alle esigenze presenti nel territorio.

8.2.2 Ricerca Attività di Prossimità

Individua le attività disponibili entro un raggio specifico a partire da un punto preciso, come un indirizzo, per esempio quello di un hotel, oppure un punto qualsiasi di cui si conoscono le coordinate GPS. Questo strumento è indicato quando l'utente formula una richiesta del tipo, vicino a me, o nei pressi di un luogo già individuato.

8.2.3 Valutazione delle Condizioni Ambientali

Questo strumento stabilisce se le condizioni meteorologiche siano adatte a svolgere l'esperienza richiesta, trasformando le previsioni meteo in indicazioni qualitative sulla sicurezza e sul comfort dell'utente. È stato introdotto per aggirare una difficoltà nota dei modelli linguistici: il calcolo di valori fisici quantitativi. Poiché l'idoneità di un'attività all'aperto dipende da interazioni complesse tra temperatura, umidità, vento e altri fattori, il calcolo è affidato a un modello fisico deterministico (lo stesso utilizzato dall'Hiking Agent) anziché all'LLM. In questo modo nessun valore numerico viene prodotto dal modello linguistico, ed è così eliminato alla radice il rischio di allucinazioni proprio dove sarebbe più pericoloso, cioè sui numeri.

Il modello valuta lo stress termico e i rischi legati alle precipitazioni, permettendo all'agente di sconsigliare le attività potenzialmente pericolose. Per ottenere i dati meteorologici dell'area di interesse, lo strumento invoca il Weather Agent. Una descrizione più dettagliata del funzionamento del modello fisico è riportata nella sottosezione 9.5.1 del capitolo 9. Questo strumento ha una natura generale: può valutare autonomamente se le condizioni previste siano adatte a una determinata attività in un certo luogo, rispondendo a richieste dirette dell'utente del tipo “domani è una buona giornata per correre al parco di Valentino?”. Le previsioni necessarie a questa valutazione vengono richieste al Weather Agent.

8.2.4 Pianificazione Itinerari Ottimizzati

Questo strumento ha un ruolo centrale in questo agente. A partire da un punto di partenza (come un hotel o un B&B), da un intervallo temporale e dal profilo dell'utente, è in grado di selezionare quali esperienze includere nell'arco di tempo richiesto e in quale ordine visitarle. La pianificazione tiene conto sia della durata delle singole attività sia dei tempi di spostamento tra un luogo e l'altro, stimati in base al mezzo di trasporto preferito dall'utente (indicato durante la conversazione con il sistema oppure recuperato dal suo profilo utente). In questo modo il tragitto tra un'esperienza e la successiva viene valutato realisticamente rispetto ai mezzi disponibili per l'utente.

Invece di restituire un semplice elenco di luoghi da visitare, lo strumento costruisce un vero e proprio piano di viaggio suddiviso in quattro fasi:

1. **Input e pre-filtraggio geografico:** lo strumento riceve il profilo dell'utente, le previsioni del tempo e il secondo dataset descritto in precedenza (quello costruito a partire da GetYourGuide), che contiene i dati necessari per l'utilizzo di questo strumento.

Come prima operazione, lo strumento calcola, mediante la formula di Haversine (l'equazione utilizzata per determinare la distanza in linea d'aria tra due coordinate geografiche), la distanza di ciascuna attività rispetto al punto di partenza selezionato. Successivamente, vengono escluse le attività che superano una soglia massima prestabilita, ad esempio 100 km nel caso in cui sia previsto l'utilizzo dell'automobile come mezzo di trasporto. Si tratta di un calcolo poco costoso che riduce il numero di candidati, alleggerendo le fasi successive.

2. **Filtro per compatibilità meteo e filtro semantico:** a questo punto il sistema restringe i candidati in due passaggi successivi.

Il primo tiene conto del meteo atteso, ma non ricorre agli embedding: sfrutta invece la classificazione: al chiuso / all'aperto / riparata assegnata offline a ciascuna esperienza (descritta nella sezione sui dati). Il meccanismo è volutamente semplice e mette in relazione due elementi di natura diversa. Da un lato, ogni attività possiede una di queste tre etichette, dall'altro, le previsioni meteo vengono ridotte a una semplice indicazione binaria, ovvero se le attività all'aperto siano sconsigliate oppure no. Questa indicazione è ottenuta in quanto prima di utilizzare questo strumento si può usare il weather agent e ottenere da esso una risposta in linguaggio naturale. Per poi mettere questo testo all'interno di un modello linguistico per esprimere un giudizio qualitativo sulle condizioni previste (ad esempio, in caso di pioggia il valore è "sconsigliate"). Si noti che all'LLM non è richiesto alcun calcolo numerico, ma solo una valutazione qualitativa di tipo sì/no.

In pratica, l'indicazione binaria sul meteo funge da interruttore: se le attività all'aperto sono sconsigliate, le attività etichettate come "all'aperto" vengono scartati e restano solo quelli "al chiuso" o "riparata". Se il meteo è favorevole, non viene escluso nulla. Essendo un semplice confronto tra un'etichetta e un valore sì/no e non una misura di similarità, il filtro è deterministico e inoltre si sa sempre perché un luogo è stato scartato.

Il secondo passaggio è invece semantico e sfrutta gli embedding descritti nella sottosezione dedicata alla loro generazione: un embedding costruito unendo la descrizione del profilo dell'utente e la query ricevuta dall'orchestratore viene confrontato con l'embedding completo delle attività superstiti dal filtro precedente, restringendo la selezione a un numero ristretto di candidati, ad esempio i 50 migliori. La scelta finale di quali includere nell'itinerario e in che ordine è delegata interamente all'algoritmo del punto successivo.

3. **Ottimizzazione logistica:** i candidati vengono processati da un algoritmo di ottimizzazione basato sulla libreria OR-Tools, che stabilisce quali attività includere e in quale sequenza visitarli. Il problema è modellato come un problema di routing su grafo: i luoghi candidati e il punto di partenza diventano i nodi, Gli archi che collegano i nodi sono pesati con il tempo di percorrenza necessario per spostarsi da un nodo all'altro, stimato a partire dalla distanza geografica tra i due punti, calcolata con la formula di Haversin (utilizzata per la sua rapidità e tenendo conto che però questa non misura la lunghezza del tragitto effettivo tra i tra i nodi, e che conseguentemente in futuro dovrebbe essere cambiata) e da una velocità media di percorrenza dipendente dal mezzo di trasporto. A ciascun nodo è inoltre associata la durata prevista della visita, cioè il tempo in cui il turista rimane fermo in quel luogo. Il percorso è assegnato a un unico "veicolo", il turista, che parte dal punto di partenza e vi fa ritorno al termine della giornata. Il vincolo fondamentale è il budget temporale: la somma dei tempi di spostamento e dei tempi di visita non può superare la disponibilità dichiarata dall'utente, ad esempio un'intera giornata di otto ore oppure soltanto la mattinata.

Rispetto ai problemi di routing classici, come il Vehicle Routing Problem, c'è però una differenza sostanziale: lì tutti i nodi devono obbligatoriamente essere visitati, mentre qui le visite sono facoltative, ed è proprio questa la caratteristica più importante dell'intero algoritmo. Essendo impossibile raggiungere tutti i candidati nel tempo a disposizione, il sistema deve prima di tutto decidere quali luoghi lasciare fuori. E' questo che distingue questo problema dal semplice trovare il percorso più breve. A ciascun luogo, infatti, è associato un valore che ne esprime l'interesse per l'utente e la disponibilità di tempo è limitata: l'obiettivo non è quindi visitare tutti nodi percorrendo meno strada possibile,

ma raccogliere il maggior valore complessivo restando entro il budget di tempo (si veda dopo per comprendere cosa si intende per valore in questo caso). Un problema con queste caratteristiche è noto in letteratura come Orienteering Problem. In OR-Tools, però, non viene "massimizzato un valore": sa soltanto minimizzare un costo. Questo strumento sfrutta a questo scopo il meccanismo delle *disjunction*, che permette di rendere ogni luogo facoltativo: a ciascun nodo viene associata una penalità, e l'algoritmo è libero di escluderlo dal percorso a patto di "pagarla". Il costo che il risolutore cerca di minimizzare diventa allora la somma di due componenti: il tempo totale speso a spostarsi e la penalità di tutti i luoghi lasciati fuori.

La penalità è stata posta pari al valore di importanza dell'attività. In questo modo, scartare una di grande valore "costa" molto, scartarne uno irrilevante costa poco. Minimizzare questo costo complessivo equivale allora a tenere le attività più preziosi e a viaggiare il meno possibile. Di conseguenza, massimizzare il valore raccolto, cioè i nodi inclusi e minimizzare il costo diventano la stessa cosa. La conseguenza pratica è intuitiva: un luogo viene incluso solo se il valore che porta giustifica il tempo necessario per raggiungerlo, se arrivarci costa più di quanto vale, l'algoritmo lo salta.

Perché la somma nella funzione obiettivo abbia senso, valore e tempo devono essere espressi nella stessa unità di misura: l'affinità di un luogo è un numero compreso tra 0 e 1, mentre i tempi di percorrenza sono misurati in minuti. Per convertire l'uno negli altri non si è scelto un fattore arbitrario ma è il tempo massimo di spostamento che vale la pena affrontare per raggiungere un singolo luogo di interesse massimo. Tale soglia è stata fissata a 80 minuti, un valore che riflette la tolleranza ragionevole di un turista per un singolo trasferimento. La penalità di ciascun luogo è quindi il suo valore di affinità moltiplicato per questi 80 minuti: un luogo con affinità massima (pari a 1) "vale" fino a 80 minuti di deviazione, uno con affinità 0,5 ne vale 40, e così via. In questo modo la conversione tra valore e tempo è una scelta interpretabile: se per raggiungere un luogo serve più tempo di quello che il suo valore "giustifica", il modello lo scarcerà.

Questo strumento definisce il valore di importanza di ciascun luogo a partire dall'affinità semantica calcolata nella fase precedente, ovvero da quanto la descrizione del luogo risulta vicina al profilo dell'utente e alla sua richiesta. Questo punteggio di affinità viene poi amplificato da un fattore di popolarità, calcolato come media bayesiana di valutazione e numero di recensioni: così un luogo con una poche recensioni molto positive non risulta essere necessariamente migliore di un'attrazione consolidata con migliaia di giudizi positivi.

Poiché ogni spostamento e ogni sosta consumano il budget temporale, il sistema è costretto a bilanciare l'interesse per una singola attrazione con il tempo necessario per raggiungerla e "viverla": tenderà a scartare anche luoghi molto affini se il viaggio per arrivarci è troppo lungo, preferendo magari un gruppo di attività vicine tra loro che, sommate, offrono un'esperienza complessiva migliore.

Va precisato che il modello attuale non considera gli orari di apertura dei singoli luoghi in quanto ci sono delle inconsistenze all'interno del dataset ma questo può essere una naturale estensione di questo approccio di risoluzione, trasformandolo in Orienteering Problem with Time Windows. Va infine precisato che il modello utilizzato opera con strategie euristiche entro un limite di tempo prefissato: la soluzione restituita è quindi di buona qualità ma non necessariamente l'ottimo assoluto, un compromesso inevitabile, data la natura NP-hard del problema.

4. **Output:** al termine del processo, lo strumento restituisce all'Activity Agent l'itinerario ordinato: per ciascuna tappa sono indicati le attività selezionate con la sua descrizione, la durata prevista della visita e il tempo di spostamento tra una tappa e l'altra.

8.3 Esempio Itinerario

Per fare un esempio pratico, nel caso in cui un utente formuli al sistema la seguente richiesta: "Voglio mangiare cibo locale, esplorare la città in modo divertente e scoprire le tradizioni messicane".

Il Travel Agent individua l'agente più appropriato e inoltra una richiesta all'Activity Agent: "Trova delle attività a Cancún legate all'esplorazione, al cibo di strada e alla cultura. Crea quindi un itinerario giornaliero con partenza e ritorno in hotel", includendo anche le informazioni del profilo utente.

Ricevuta l'istruzione, l'Activity Agent invoca il weather agent per ottenere dati meteo e poi il tool di pianificazione, che seleziona tra i candidati le attività da includere e l'ordine da tra una tappa e l'altra. Il risulta l'itinerario finale viene illustrato nella Figura 8.1:

1. **Mopeds Adventure (1,5 h):** raggiungibile in 50 minuti (in auto come scritto nel profilo utenete) dall'hotel, è un tour guidato su scooter elettrici per esplorare la storia e i migliori panorami della zona, con sosta in una spiaggia privata.
2. **Cancun Taco Tour (2,5 h):** a 22 minuti di distanza, è un'avventura culinaria nel centro città, tra street food e degustazione di tequila.

3. **Day of the Dead Tour (2,0 h):** dopo un breve spostamento di 12 minuti, si partecipa a una visita dedicata alla tradizione messicana del Giorno dei Morti, scoprendone usanze e simboli attraverso altari decorati e offerte tipiche.
4. **Rientro in hotel:** il percorso si conclude con un viaggio di ritorno di 22 minuti verso l'hotel.

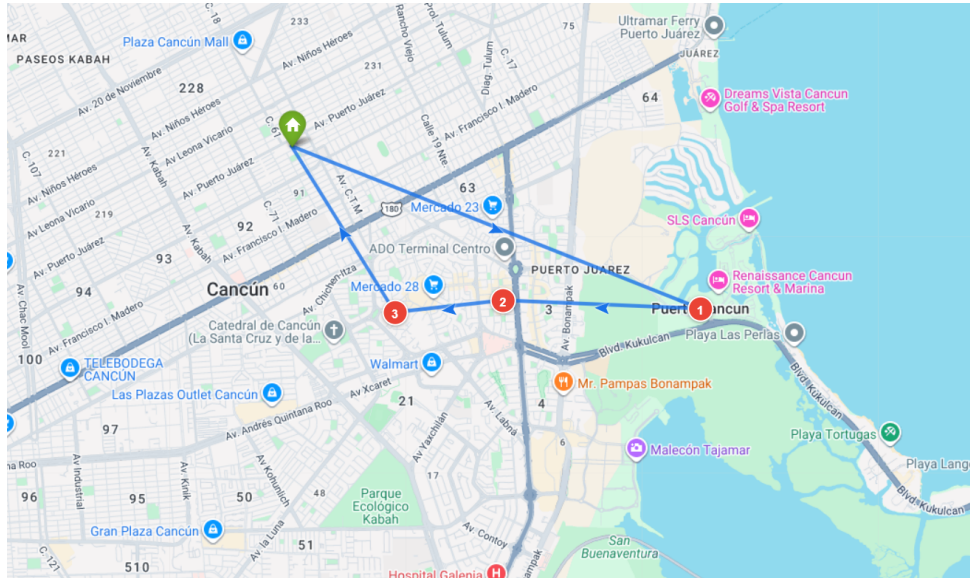


Figura 8.1: Itinerario giornaliero prodotto dall'Activity Agent: l'hotel e le tre esperienze selezionate, collegati nell'ordine di visita. I segmenti raffigurano schematicamente i collegamenti in auto tra una tappa e la successiva.

8.4 Rapporto con gli Altri Agenti

L'Activity Agent e il Location Agent hanno ruoli complementari, ma si distinguono anche nel modo in cui gestiscono e interrogano i propri dati. L'Activity Agent lavora su file locali, copiati sul container al momento del deploy: il suo dataset di attività è sufficientemente piccolo da rendere praticabile questo approccio. Il dataset del Location Agent è invece di natura diversa: ha al suo interno sei macro-categorie di punti di interesse (ristorazione, alloggi, negozi, attrazioni e intrattenimento, elementi naturali e servizi). Per questa sua estensione è troppo grande per essere caricato interamente nella memoria del container. I dati risiedono perciò su Amazon S3 e vengono interrogati in due passaggi: prima il sistema apre soltanto la partizione della regione d'interesse, poi applica il poligono geografico dell'area richiesta come

maschera spaziale, estraendo unicamente i punti di interesse che ricadono entro quel confine.

Infine, è possibile che i due agenti collaborino tra loro. Quando l'Activity Agent genera un itinerario, calcola i segmenti di collegamento tra una tappa e la successiva e li trasmette al Location Agent, che può trovare punti di interesse non soltanto nei punti di partenza e di arrivo, ma anche lungo il tragitto che l'utente percorre tra un'attività e l'altra.

Oltre al Location Agent, l'Activity Agent si avvale anche del Weather Agent. Nello strumento di pianificazione degli itinerari le previsioni meteo servono a stabilire quali attività consigliare in base alle condizioni ambientali: ciò introduce una dipendenza tra i due agenti, ma permette di ottenere itinerari migliori tenendo conto di questo fattore. Il Weather Agent è impiegato anche dallo strumento di Valutazione delle Condizioni Ambientali, dove però ha un uso più generico, potendo rispondere anche a domande dirette come “domani è una buona giornata per correre in un determinato luogo?”.

Capitolo 9

Hiking Agent

9.1 Introduzione

L'Hiking Agent gestisce la pianificazione e la validazione delle escursioni naturalistiche. Il componente estende le funzionalità di ricerca geospaziale tradizionale integrando un motore di raccomandazione sensibile al profilo utente e vincoli di sicurezza ambientale e il comfort dell'utente.

Il sistema adotta un approccio gerarchico in cui l'agente delega la raccolta dei dati grezzi a due sotto-agenti specializzati: interpella il *Weather Agent* fornendogli l'arco temporale richiesto per ottenere informazioni meteorologiche e chiama il *Location Agent* basandosi sulle coordinate spaziali dei sentieri per ottenere i Punti di Interesse (POI) limitrofi.

Sono qui elencati i tool a disposizione dell'agente:

- **Filtraggio dei percorsi:** Consente la ricerca e il filtraggio dei sentieri all'interno di un database geografico basato su parametri fisici e tecnici.
- **Valutazione delle condizioni meteorologiche per l'escursione:** verifica la fattibilità e la sicurezza dell'escursione confrontando i dati meteorologici, elaborati da un modello fisico, con soglie di rischio e di comfort derivate dalla letteratura.
- **Build Report e LLM come giudice:** Assembla tutte le informazioni raccolte in un documento strutturato finale.

Dato che questa tesi si concentra sull'implementazione di una architettura multi-agente, i tool e gli algoritmi deterministici di supporto saranno descritti in modo da capire a "cosa" servono e "perchè" sono presenti. Tuttavia, in alcuni casi i dettagli sul loro funzionamento verranno omessi per mantenere l'attenzione sul comportamento di questo agente, a scapito della precisione nella descrizione di alcuni algoritmi.

Per questo agente esiste una separazione tra la fase di acquisizione dei dati e quella di interpretazione fisica. In generale, per prima cosa l'agente chiama il Weather Agent, che interroga le API meteorologiche. Una volta ottenute queste previsioni grezze vengono incluse nell'Archivio Interno dell'agente. La seconda fase riguarda l'utilizzo degli altri strumenti per filtrare e selezionare i migliori percorsi, oppure verificare, tramite un modello fisico, se le condizioni ambientali consentono di svolgere l'attività in sicurezza.

9.2 Rapporto con gli Altri Agenti

- **Travel Agent:** E' il padre dell'hiking e decide di invocarlo quando la richiesta dell'utente riguarda specificamente il trekking, le escursioni in montagna o i percorsi naturalistici. Questa delega avviene per separare le attività generiche (ad esempio culturali, culinarie o sportive) dalle escursioni, permettendo a questo specialista di fornire dettagli precisi in questo ambito.
- **Weather Agent:** E' un sotto-agente dell'Hiking Agent e viene interpellato ogni volta che è necessaria una valutazione delle condizioni meteorologiche. Senza i suoi dati, gli strumenti dell'Hiking Agent non potrebbero funzionare, poiché le previsioni orarie sono necessarie per calcolare il comfort termico e valutare i potenziali rischi per la sicurezza dell'utente.
- **Location Agent:** Mentre l'Hiking Agent si concentra sulle caratteristiche dei sentieri, il Location Agent (figlio dell'Hiking Agent) si occupa degli aspetti logistici, fornendo informazioni su servizi essenziali come parcheggi, rifugi e punti di ristoro. Inoltre, dopo che l'Hiking Agent ha filtrato i percorsi e ne ha selezionato uno, ne estrae la traccia GPS per creare un'unica struttura, denominata Focus Geometry, che viene salvata nello spazio cloud condiviso. Questa rappresenta un'area geografica più ristretta rispetto a quella individuata inizialmente dall'orchestratore. Al Location Agent viene quindi trasmesso esclusivamente il codice identificativo (ID) della Focus Geometry. Utilizzando tale geometria, crea un buffer lungo il tracciato e ricerca al suo interno i Punti di Interesse (POI) richiesti dall'utente.

9.3 Costruzione del database dei percorsi

I percorsi escursionistici gestiti da questo agente provengono da file in formato GPX scaricati dal portale ufficiale del CAI (Club Alpino Italiano)[13]. Inizialmente, in questo file sono presenti solo percorsi con attributi i metadati di base, come il nome

del sentiero, il codice identificativo, la scala di difficoltà ufficiale, la lunghezza e altre caratteristiche.

Successivamente, i dati GPX vengono elaborati da un classificatore che esegue un processo di arricchimento, aggiungendo nuovi attributi rispetto a quelli già presenti in modo da creare un nuovo database arricchito. Alcuni esempi di questi nuovi parametri sono:

- **Pendenza media:** viene calcolata segmentando il percorso in tratti di un chilometro e determinando la pendenza di ciascun tratto. La pendenza media del percorso è quindi ottenuta facendo la media delle pendenze dei singoli chilometri.
- **Tempo di percorrenza stimato:** calcolato in base alla lunghezza e pendenza del terreno.
- **Localizzazione Amministrativa:** Ad ogni percorso viene aggiunta i dati amministrativi: regione, provincia e comune di appartenenza in modo tale che il database sia filtrabile in base al luogo in cui si vuole trovare un percorso.
- **Il "Crux":** identificazione del chilometro più impegnativo dell'intero percorso.

9.4 Valutazione condizioni ambientali

Per interpretare le previsioni meteorologiche è stato sviluppato un modello fisico capace di stabilire se un ambiente naturale sia adatto allo svolgimento di specifiche attività ricreative. Il modello considera l'interazione tra l'ambiente e il corpo umano, per ogni ora della giornata esso classifica le condizioni in una tra quattro classi di idoneità: ideale, accettabile, sconsigliata e pericolosa. Tale classe non deriva da un singolo parametro, ma dalla combinazione di più fattori ambientali (stress termico, raggi UV, precipitazioni, visibilità e vento): ciascuno viene confrontato con le proprie soglie e produce un verdetto, e la classe finale dell'ora coincide con il verdetto peggiore tra tutti i fattori. E' quindi sufficiente un solo elemento critico per abbassare l'intera valutazione, anche quando tutti gli altri risultano favorevoli. Le criticità così rilevate vengono inoltre raccolte in un elenco che accompagna l'ora nelle fasi successive.

La scelta di un output qualitativo, anziché di un punteggio numerico, è dovuto a due esigenze. La prima è che le soglie utilizzate per la classificazione provengono da standard e da letteratura consolidati (le categorie di stress termico associate all'UTCI, i criteri di comfort di Lawson, le fasce di esposizione UV della WHO). L'utilizzo di un modello che combinasse grandezze fisiche eterogenee in un unico valore numerico avrebbe richiesto l'introduzione di criteri di aggregazione e difficilmente giustificabili sulla base della letteratura scientifica. La seconda motivazione è dovuta alla natura intrinseca dei modelli linguistici, questi producono risposte non ottimali

quando devono ragionare su lunghe serie di valori numerici. E' quindi preferibile delegare a un componente deterministico la gestione di tutte le variabili meteo. In base a questi l'LLM può poi costruire la risposta discorsiva.

E' opportuno precisare che questo modello fisico non costituisce un tool diretto a disposizione dell'agente LLM: non viene invocato in autonomia dal modello linguistico, ma ha il compito di classificare le condizioni ambientali di un'ora specifica. Nella pratica l'algoritmo viene richiamato più volte dal tool "Valutazione delle condizioni meteorologiche", il quale raccoglie le classificazioni delle singole ore e determina la migliore finestra temporale in cui svolgere l'escursione.

9.4.1 Dalla temperatura dell'aria alla temperatura radiante media

La sola temperatura dell'aria, fornita da open-meteo, non basta a descrivere ciò che una persona percepisce all'aperto: a parità di temperatura, sotto il sole si avverte molto più caldo che all'ombra. Per tenerne conto, il modello calcola la Temperatura Radiante Media (T_{mrt}), un unico valore che riassume tutto il calore che raggiunge il corpo per irraggiamento, sia dal sole sia dall'ambiente circostante.

Il calcolo parte dalla radiazione solare prevista e stima quanta di questa energia viene effettivamente assorbita dall'escursionista, utilizzando i valori di riferimento per il corpo umano indicati dalla norma ISO 7726 (*Ergonomics of the thermal environment*). L'energia così ottenuta viene infine ricondotta a una temperatura equivalente in gradi Celsius, che rappresenta la Temperatura Radiante Media: un valore più alto della semplice temperatura dell'aria quando l'irraggiamento solare è intenso, mentre è vicina ad essa in condizioni di cielo coperto.

9.4.2 Temperatura Percepita (UTCI) e Metabolismo

Una volta calcolata la Temperatura Radiante Media, essa viene unita con i dati relativi alla temperatura dell'aria, all'umidità e alla velocità del vento all'interno dell'Universal Thermal Climate Index (UTCI, Jendritzky et al., 2012). L'UTCI rappresenta la "temperatura percepita", calcolata simulando la reale risposta fisiologica umana (sudorazione, brividi) [14]. Il valore ottenuto non viene convertito in "un punteggio di comfort" che va da 0 a 100, ad esempio. Ma il valore ottenuto viene confrontato con le categorie di stress termico che accompagnano ufficialmente l'indice, le quali suddividono la scala UTCI in fasce che vanno dallo stress estremo da freddo allo stress estremo da caldo. È però necessaria una precisazione: l'UTCI è definito per una condizione di attività fissa, ovvero un soggetto di riferimento che cammina a circa 4 km/h (corrispondente a un tasso metabolico di circa 2,3 MET), e non accetta il metabolismo come parametro di ingresso. Le sue categorie di stress termico, di conseguenza, non tengono conto dello sforzo fisico effettivamente svolto.

Poiché il comfort dipende dal calore prodotto dal corpo, il sistema introduce una correzione approssimata: le soglie delle categorie vengono traslate di una quantità proporzionale all'intensità dell'attività, quantificata in circa 2 °C per ogni MET di differenza rispetto al riferimento. Più l'attività è impegnativa, maggiore è il calore metabolico prodotto e più ampio è lo spostamento delle soglie verso temperature ambientali più fredde. Il valore così ottenuto determina il giudizio sull'ora: quando non c'è stress termico l'ora è ideale, con uno stress moderato resta accettabile, mentre uno stress forte la rende sconsigliata e uno stress estremo la rende pericolosa. Questo giudizio viene poi confrontato con quello degli altri fattori ambientali e come descritto in precedenza, la classe finale dell'ora coincide sempre con il verdetto peggiore.

9.4.3 Effetto del vento sulla sicurezza dell'utente

Il vento influisce sull'escursione in due modi. L'effetto di raffreddamento è già considerato dall'indice UTCI, che include la velocità del vento nel calcolo della temperatura percepita e non richiede quindi una valutazione a parte. Resta però un secondo aspetto, che l'UTCI non coglie: la spinta meccanica delle raffiche, queste sono capaci di ostacolare il cammino e di far perdere l'equilibrio. E' un rischio indipendente dalla temperatura e per questo il modello si confronta la velocità del vento con le soglie di sicurezza di Lawson (1978).

9.4.4 L'effetto dell'altitudine sui raggi UV

Oltre al comfort, il modello valuta il rischio legato all'esposizione ai raggi ultravioletti. Le previsioni meteorologiche standard forniscono un indice UV calcolato a livello del mare, ma in montagna l'irraggiamento è più intenso: salendo di quota l'atmosfera che filtra i raggi UV diminuisce, e la loro intensità aumenta di circa il 10% ogni 1000 metri di dislivello. Il modello applica quindi questa correzione, indicata dalla World Health Organization (WHO), aggiornando l'indice UV in base all'altitudine del percorso [15].

L'indice così corretto viene poi confrontato con le fasce di esposizione definite dalla WHO. Con valori bassi o moderati non viene segnalato alcun rischio particolare; nelle fasce "alta" e "molto alta" il modello segnala una criticità, nella fascia "estrema" l'esposizione è tale da contribuire a classificare l'ora come sconsigliata.

9.4.5 Precipitazioni

Le precipitazioni vengono valutate in modo indipendente dalle condizioni termiche e confrontate con soglie dedicate: anche un'ora con una temperatura ideale può essere

classificata come sconsigliata o pericolosa se piove.

9.4.6 Vetì per la sicurezza

Ogni grandezza ambientale viene classificata su una propria scala, e il gradino più alto di ciascuna scala segnala una situazione di maggior pericolo. Ad esempio, il vento diventa pericoloso quando le raffiche sono così forti da far perdere l'equilibrio, la temperatura quando il freddo espone al rischio di ipotermia, la pioggia quando si trasforma in un temporale. Quando anche uno solo di questi fattori raggiunge il suo livello di pericolo, è il modello stesso a intervenire attraverso un sistema di vetì: quell'ora viene classificata direttamente come pericolosa, senza bisogno di guardare agli altri parametri.

9.5 Strumenti dell'agente

Di seguito vengono analizzati il funzionamento e il ruolo di ciascun tool all'interno di questo agente.

9.5.1 Filtraggio dei percorsi

Questo tool agisce come un filtro selettivo in grado di identificare un sottoinsieme di percorsi partendo dal vasto database descritto in precedenza. Il suo compito è estrarre esclusivamente i tracciati che rispettano sia i vincoli geografici (distanza raggiungibile dall'utente) sia far combaciare la difficoltà del percorso con preferenze espresse dall'utente. Questo tool consente di scartare quasi la totalità dei percorsi all'interno del dataset e gli altri due tool potranno concentrarsi sui restanti candidati, portando un risparmio in termini di chiamate API e di token. L'infrastruttura tecnologica si basa su GeoParquet, Hive Partitioning e DuckDB. In pratica, questo consiste nell'applicazione di un filtro geospaziale e di uno sui vincoli imposti dall'utente.

Filtraggio geospaziale

Il filtraggio geospaziale usa la geometria ricevuta dall'Orchestratore per limitare la ricerca ai soli percorsi presenti nell'area di interesse. Il processo avviene in due passaggi.

1. **Selezione delle partizioni:** DuckDB legge solo i file GeoParquet relativi al Paese e alla regione amministrativa individuati dall'Orchestratore, ad esempio tramite percorsi del tipo `country=IT/admin\_level\_1=piemonte/trails.geoparquet`. In questo modo non viene scansionato l'intero database, ma solo la porzione rilevante.

2. **Intersezione spaziale:** dopo la selezione delle partizioni, è comunque necessario un controllo più preciso. La partizione amministrativa indica solo che un percorso si trova nella stessa regione o area generale della richiesta, ma non garantisce che attraversi davvero la zona cercata dall'utente. Ad esempio, due sentieri possono appartenere alla stessa regione, ma trovarsi in province molto distanti tra loro. Per questo motivo il sistema confronta la forma reale del sentiero con la geometria dell'area geografica ottenuta dall'orchestratore e mantiene solo i percorsi che la toccano o la attraversano almeno in un punto. In questo modo vengono eliminati i tracciati che appartengono alla partizione corretta, ma che non sono effettivamente pertinenti..

Al termine di questa fase restano solo i percorsi effettivamente compatibili con l'area geografica richiesta, che vengono poi passati al filtro sui vincoli dell'utente.

Filtro sui vincoli utente

Sui candidati che hanno superato il filtro spaziale, vengono applicati i vincoli fisici estratti dal profilo dell'utente o dalla query (ad esempio: distanza massima percorribile, dislivello massimo tollerato o difficoltà tecnica (scala CAI)). Questo stadio assicura che i risultati estratti dal database precalcolato 9.3 siano compatibili con l'esperienza e la preparazione dell'escursionista.

L'effetto combinato di questi due stadi riduce drasticamente lo spazio di ricerca, passando da migliaia di potenziali tracciati a un insieme ristretto di candidati (solitamente tra i 5 e i 10). Questi dati vengono salvati nella Archivio Interno dell'agente per poi essere utilizzati per altri tool.

Valutazione delle condizioni meteorologiche

Questo strumento serve a trovare l'ora migliore e più sicura per fare un'escursione. Invece di fermarsi alle medie del giorno, guarda le previsioni ora per ora, sul giorno preciso indicato nella richiesta.

Questo strumento si affida al Weather Agent per estrarre le previsioni meteorologiche necessarie e le include nel proprio Archivio Interno. Questi dati poi vengono elaborati attraverso il modello fisico descritto nella Sezione 9.4. Questa fase preliminare permette di ottenere, per ogni ora dell'intera giornata, una classe di idoneità (ideale, accettabile, sconsigliata o pericolosa). Questo strumento esegue quindi i seguenti passaggi:

1. **Generazione e Scansione delle Finestre Temporal:** Questa fase prevede la definizione di un insieme di "finestre temporali" per ogni sentiero candidato.

La durata di tali intervalli è determinata in base alla lunghezza in ore di percorrenza del tracciato e viene fatta scorrere lungo l'arco delle 24 ore per identificare ogni possibile fascia oraria di attività (ad esempio: 08:00-12:00, 09:00-13:00 e così via, per un'escursione della durata di 4 ore). Tale approccio permette di individuare tutti gli intervalli per cui l'utente potenzialmente potrebbe percorrere. Tuttavia, qualora l'utente esprima una preferenza esplicita sull'orario di partenza (ad esempio richiedendo di partire "la mattina" o indicando un'ora precisa), il sistema restringe l'intervallo temporale partendo dall'ora stabilita e con durata pari a quella del percorso.

2. **Valutazione delle Finestre: Classe e Criticità:** Definiti gli intervalli, il sistema esamina le classificazioni orarie prodotte dal modello fisico e ne ricava una valutazione complessiva della finestra, aggregando i dati orari in due modi distinti. La *classe* della finestra coincide con quella della sua ora peggiore: poiché le classi sono ordinate per pericolosità (ideale, accettabile, sconsigliata, pericolosa), basta una sola ora pericolosa per scartare l'intero intervallo, e una sola ora sconsigliata per rendere sconsigliata tutta la finestra. Le criticità, invece, non essendo ordinabili tra loro (un forte UV non è "peggiore" di un vento fastidioso, sono problemi di natura diversa), vengono raccolte nell'unione delle criticità segnalate nelle singole ore. Di conseguenza, una finestra raggiunge la classe ideale soltanto quando ogni sua ora è a sua volta ideale e priva di criticità.
3. **Confronto tra Candidati e Isolamento della Finestra Ottimale:** Per ogni sentiero candidato il sistema sceglie la sua "finestra ottimale", cioè la migliore tra tutte le finestre disponibili. Per capire quale sia la migliore le confronta seguendo un ordine di priorità: conta prima di tutto la classe più alta; se due finestre hanno la stessa classe, vince quella con meno criticità; se sono ancora pari, quella con più ore nella classe migliore; e infine, a parità di tutto, quella più vicina all'orario preferito dall'utente. Se per un sentiero tutte le finestre sono pericolose, quel sentiero viene segnato come non percorribile in quel giorno. Gli altri vengono invece salvati nell'Archivio Interno insieme alla loro finestra ottimale, alla classe raggiunta e all'elenco delle criticità.

9.5.2 Build Report e LLM come giudice

L'obiettivo di questo strumento è quello di sintetizzare i dati raccolti per preparare la risposta finale. Il processo si sviluppa in tre fasi:

1. **Ordinamento e Selezione dei Top 5:** Lo strumento recupera tutti i sentieri ancora validi, vengono ora riordinati in base alla classe di idoneità della loro "finestra ottimale": prima i percorsi con finestra ideale, poi quelli accettabili e

infine gli sconsigliati, mentre i percorsi risultati non praticabili vengono esclusi. A parità di classe si privilegia il percorso con il minor numero di criticità segnalate. Al termine di questo riordino, il sistema mantiene soltanto i 5 percorsi con le condizioni più sicure e favorevoli.

2. **Aggregazione di Attributi e Condizioni:** Per ciascuno dei 5 candidati selezionati, il sistema raccoglie in un unico blocco strutturato tutte le caratteristiche specifiche dei percorsi (ad esempio livello di difficoltà, distanza, tempi di percorrenza, dislivello, profilo altimetrico), i parametri meteorologici della finestra temporale ottimale (temperatura, raffiche di vento, precipitazioni previste) insieme alla classe di idoneità assegnata e all'elenco delle criticità individuate dal modello fisico.
3. **LLM come Giudice Finale:** Tutte le informazioni raccolte vengono tradotte in formato testuale e assemblate in un unico "Prompt-esteso" contenente istruzioni dirette. Questo documento costituisce l'output finale di questo specifico tool e viene trasmesso all'Hiking Agent. Ricevuto il Prompt-esteso, l'agente lo aggiunge automaticamente alla query originale. A questo punto l'LLM assume il ruolo di giudice: considera i dati tecnici e atmosferici dei 5 percorsi, le classi di idoneità trovate dal modello fisico (quindi in una forma adatta al ragionamento linguistico) con le reali preferenze dell'escursionista, sceglie i 3 itinerari migliori ed elabora la risposta finale. Il modello non si limita a riportare le informazioni ricevute, ma fornisce raccomandazioni pratiche (es. l'equipaggiamento tecnico necessario) e giustifica in modo discorsivo il motivo per cui ogni specifico tracciato rappresenta una valida scelta.

9.6 I vantaggi di più tools rispetto a una pipeline fissa

Mantenere gli strumenti logicamente separati, invece di raggrupparli in un unico blocco, è una scelta architetturale che garantisce all'agente maggiore flessibilità e ottimizza l'efficienza computazionale.

9.6.1 Adattamento alle richieste

In un software tradizionale, le operazioni seguono un ordine prestabilito che non può essere cambiato. Nell'Hiking Agent, invece, l'uso di strumenti separati permette all'agente di adattarsi a ciò che l'utente chiede davvero. Se l'utente desidera solo un consiglio rapido sul meteo per un sentiero che già conosce, l'agente può attivare direttamente i tool meteorologici senza dover sprecare tempo e risorse nella ricerca di nuovi percorsi, rendendo l'interazione molto più simile a una conversazione umana che a un comando informatico.

9.6.2 Resistenza agli imprevisti

Un altro punto di forza di questa scelta è la capacità del sistema di gestire eventuali problemi. In una pipeline rigida, se anche solo una parte non è funzionante l'intero processo fallisce. Se l'agente utilizzando uno dei tool ottiene un errore, può riprovare o provare ad aggirare il problema. Questo approccio rende il sistema più affidabile e capace di comunicare in modo chiaro i propri limiti all'utente.

9.7 Visualizzazione del percorso migliore

Una volta generata la risposta finale l'agente prende il percorso che ritiene migliore e genera una mappa interattiva in formato HTML e un grafico del profilo altimetrico in formato PNG (per semplicità si è scelto di visualizzare solo il migliore e non tutti gli altri). Questo consente all'utente di comprendere anche visivamente le caratteristiche del percorso e riflettere se esso soddisfa o meno le sue esigenze. Un esempio di ciò viene riportato qui:

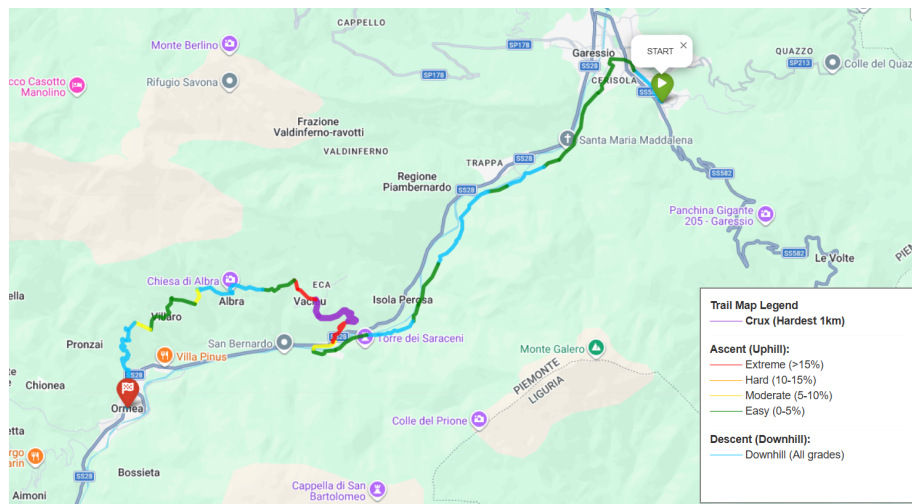


Figura 9.1: Percorso SI E01

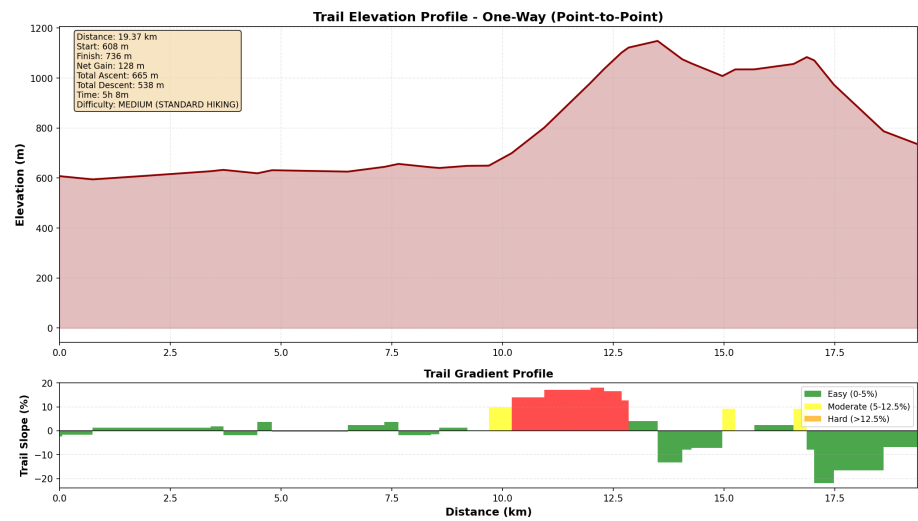


Figura 9.2: Profilo altimetrico SI E01

Capitolo 10

Weather Agent

Introduzione

Il Weather Agent rappresenta il componente del sistema specializzato nel fornire dati meteorologici completi e localizzati. Si è scelto di sviluppare questo agente "ad hoc" per adeguarsi alle esigenze specifiche del progetto, preferendo una soluzione personalizzata rispetto a modelli generici preesistenti. Tra le sue funzioni principali vi sono il monitoraggio delle condizioni attuali, il confronto con i dati storici e l'elaborazione di previsioni sia a breve che a lungo termine.

L'utilizzo di un agente dedicato in questo sistema risponde alla necessità di superare un limite intrinseco dei modelli linguistici, i quali, basandosi esclusivamente sui propri dati interni di addestramento, non possono accedere autonomamente a fonti esterne senza l'ausilio di API. Questo agente non ha come solo compito quello di chiamare i tool più adatti ad una richiesta, ma ha anche la capacità di interpretare e sintetizzare le informazioni raccolte. Tuttavia, è da tener presente che questo agente ha sì una componente di ragionamento ma poichè è parte dei "Data Expert" nella gerarchia del sistema, ha prevalentemente un ruolo di assistenza per gli agenti di livello superiore.

Inoltre, per quanto riguarda la gestione della localizzazione, l'Orchestratore fornisce la geometria spaziale di riferimento sotto forma di poligono. Per le interrogazioni alle API meteorologiche si è scelto di utilizzare un singolo punto coordinato, identificato calcolando il centroide del poligono stesso. Questa semplificazione è ritenuta idonea in quanto l'area geografica considerata è limitata al massimo a un territorio comunale o provinciale, un'area sufficientemente ridotta da non giustificare l'uso di molteplici punti di campionamento.

L'agente opera attraverso una serie di strumenti che ne definiscono le capacità:

- **Nowcasting:** ottiene informazioni ad altissima precisione per l'immediato futuro (prossime 2 ore).
- **Previsioni a breve termine:** scenari dettagliati per i prossimi 1-3 giorni.
- **Previsioni a medio termine:** proiezioni probabilistiche per un orizzonte fino a 10 giorni.
- **Previsioni a lungo termine:** condizioni climatiche a lungo termine, da 1 a 9 mesi.
- **Dati Storici:** accesso ad archivi meteorologici passati.
- **Raccolta dati grezzi:** raccolta di parametri meteorologici specifici per modelli fisici di altri agenti.
- **Strumento di gestione dati:** garantisce il recupero delle informazioni dal Working Storage.

Tramite questi strumenti l'agente meteo permette di tradurre grandi quantità di dati tecnici in informazioni di facile interpretazione.

10.1 Open-Meteo

Tutti gli strumenti che l'agente ha a disposizione si basano sull'utilizzo di un'API open-source, che aggrega i dati prodotti dai principali centri meteorologici mondiali, tra cui il NOAA statunitense, il DWD tedesco e l'ECMWF europeo. A differenza dei servizi commerciali, Open-Meteo rende accessibili dati ad alta precisione senza restrizioni né abbonamenti. La piattaforma si occupa internamente di scaricare ed elaborare i modelli meteorologici grezzi dai supercomputer governativi, convertendo formati scientifici come GRIB e NetCDF in serie temporali pronte all'uso. Questo ha permesso di concentrare lo sviluppo sulla logica dell'agente anziché sull'elaborazione dei dati.

Ho scelto Open-Meteo per tre motivi principali:

- **Copertura globale:** Requisito indispensabile per un sistema agentico progettato per operare su qualsiasi coordinata geografica. Grazie all'utilizzo dei più avanzati modelli fisico-matematici, l'agente può fornire risposte anche in aree remote, riducendo la dipendenza da stazioni fisiche locali, pur restando l'accuratezza inferiore nelle regioni con scarsa copertura osservativa.

- **Continuità temporale:** La piattaforma consente di ottenere tutti i tipi di dati meteorologici: dalla ricostruzione storica al nowcasting, fino alle previsioni di breve, medio raggio e lungo stagionali.

10.1.1 Utilizzo di ICON-EU, GFS Seamless, GFS Ensemble e ECMWF

Come accennato, Open-Meteo non produce previsioni proprie, ma aggrega e rende accessibili i modelli fisico-matematici sviluppati dai principali centri meteorologici mondiali. Questo significa che, al momento di effettuare una richiesta per la maggior parte degli strumenti, l'agente deve specificare esplicitamente quale modello utilizzare. La selezione del modello è un'operazione gestita dall'agente per ottimizzare i risultati in base all'area geografica e all'orizzonte temporale richiesto. Fanno eccezione le richieste a brevissimo termine (nowcasting), per le quali la selezione del modello di calcolo è interamente delegata ad Open-meteo. Negli altri casi l'agente adotta una strategia specifica basata su questi quattro modelli principali:

- **ICON-EU:** Sviluppato dal DWD tedesco, è il modello specifico per il territorio europeo grazie alla sua altissima risoluzione spaziale (circa 7 km), superiore a quella dei modelli globali. L'agente lo utilizza per la ricostruzione storica ad alta risoluzione in Europa e per le previsioni a breve termine. Il suo limite risiede nella copertura esclusivamente europea e in un orizzonte previsionale di circa 5 giorni, oltre il quale la sua affidabilità decade rapidamente.
- **GFS Seamless:** Sviluppato dal NOAA statunitense, è il sistema che l'agente utilizza per la ricostruzione storica a copertura globale, nelle regioni non coperte da ICON-EU. Grazie all'assimilazione continua di osservazioni satellitari e stazioni di rilevamento, permette di ricostruire serie storiche affidabili per qualsiasi coordinata geografica nel mondo.
- **GFS Ensemble 0.25°:** E' il componente probabilistico del modello GFS. A differenza della versione deterministica, non produce un'unica previsione ma simula decine di scenari paralleli ottenuti perturbando leggermente le condizioni iniziali. L'agente lo utilizza per le previsioni a medio termine fino a dieci giorni, restituendo una distribuzione statistica descritta tramite percentili (p10, p50, p90) che permette di quantificare l'incertezza e identificare scenari di rischio.
- **ECMWF:** l'agente si affida al modello stagionale dell'ECMWF, accessibile tramite il Copernicus Climate Data Store per ottenere previsioni a lungo termine, fino a 9 mesi. SEAS5 non produce previsioni deterministiche giornaliere, ma distribuzioni probabilistiche mensili basate su dati storici di hindcast. Questo permette all'agente di quantificare anomalie termiche rispetto alla norma e di

stimare la probabilità che temperature o precipitazioni superino i valori storici attesi.

In questo modo la combinazione dei quattro modelli copre l'intero spettro temporale delle esigenze del sistema: ICON-EU garantisce la massima precisione locale a breve termine in Europa e per la ricostruzione storica europea. GFS Seamless copre la ricostruzione storica su scala globale, GFS Ensemble 0.25° (o ICON-EU EPS per l'Europa) viene impiegato per le previsioni probabilistiche a medio termine. Mentre ECMWF SEAS5 viene utilizzato per previsioni a lungo raggio. Fissato questo criterio l'agente in base all'orizzonte temporale e all'area geografica richiesta selezionerà il modello corretto.

10.2 Strumenti

Nowcasting

Per le previsioni a brevissimo termine, ho implementato uno strumento di nowcasting che fornisce informazioni ogni 15 minuti per le successive 2 ore. E' utilizzato in situazioni in cui il tempo può cambiare rapidamente. A differenza degli altri strumenti del sistema, l'agente non impone un modello specifico alla piattaforma, ma delega a Open-Meteo la selezione della sorgente dati. Questo dato ad alta frequenza temporale è disponibile solo in alcune aree del mondo, dove la piattaforma attinge a modelli locali ad altissima risoluzione (1-2 chilometri) come NOAA HRRR per il Nord America o DWD ICON-D2 e Météo-France AROME per l'Europa centrale. Nelle regioni non coperte da questi modelli ad alta risoluzione, la piattaforma restituisce il dato a 15 minuti calcolandolo per interpolazione a partire dai valori orari disponibili.

Previsioni a breve termine

Lo strumento di previsione a breve termine è progettato per fornire un quadro meteorologico molto dettagliato fino ai prossimi tre giorni. Questo strumento può analizzare l'evoluzione delle variabili meteorologiche sia su base oraria che giornaliera a seconda delle necessità richieste dall'utente. Anche per questo strumento viene applicata la strategia di selezione automatica del modello meteorologico più adatto in base alla locazione richiesta.

Dati storici e ricostruzione meteorologica

Questo strumento permette di recuperare informazioni meteorologiche sfruttando database di osservazioni scientifiche. Anche in questo caso, l'agente seleziona at-

tivamente il modello più adatto in base alla coordinata geografica richiesta, come descritto in 10.1.1: i dati vengono elaborati tramite il modello ICON-EU per il territorio europeo e tramite GFS Seamless per coprire il resto del globo.

Come si è visto in precedenza, partendo da dati di rilevazione reali provenienti da stazioni al suolo, boe marine o satelliti, questi modelli applicano leggi fisico-matematiche ed eseguono operazioni di interpolazione spaziale e temporale per “riempire i buchi” della griglia geografica. Questo processo consente di ricostruire con precisione i parametri meteorologici anche in aree prive di sensori. Grazie a questo approccio, l’utente può ottenere dati storici accurati per qualsiasi coordinata geografica e ogni punto (x, y) della griglia possiede la propria serie storica di temperatura, umidità, precipitazioni e altre variabili meteorologiche.

Una volta ottenuti i dati meteorologici del periodo richiesto, il sistema li confronta con le medie degli anni passati. In modo da capire se le condizioni attuali sono in linea con la norma o se si è di fronte a un evento raro, dettaglio che non si potrebbe notare analizzando solo i dati del momento senza conoscere la storia passata in quel determinato luogo.

Questo strumento consente all’agente di comportarsi come un esperto del territorio. Se un utente richiede informazioni su una determinata località, l’agente è infatti in grado di indicare se, sulla base dei dati storici, quell’area soggetta a fenomeni ricorrenti, come la formazione di ghiaccio o la presenza di nebbia fitta, nel periodo dell’anno considerato. In questo modo, le raccomandazioni fornite non si limitano alle condizioni meteorologiche attuali, ma includono anche l’analisi dell’andamento storico, offrendo indicazioni più complete e affidabili.

Previsioni a medio termine

Le previsioni a medio raggio rappresentano uno strumento che estende l’orizzonte temporale del sistema fino a dieci giorni. La caratteristica principale di questo modulo è l’utilizzo di un sistema ensemble, che non si affida a un unico calcolo deterministico, ma analizza contemporaneamente decine di simulazioni parallele (l’ensemble) dello stesso modello, ognuna avviata con condizioni iniziali leggermente perturbate per rappresentare dei fenomeni atmosferici.

Invece di fornire un singolo valore puntuale, lo strumento restituisce una distribuzione di probabilità descritta tramite la media e i percentili della distribuzione stessa (p_{10} , p_{50} , p_{90}). Il modello utilizzato è GFS Ensemble 0.25° del NOAA per la copertura globale, oppure ICON-EU EPS per il territorio italiano. Questo output probabilistico può essere utilizzato per confrontarle con i dati storici ottenuti tramite lo strumento descritto in 10.2.

Previsioni a lungo termine

Lo strumento per le previsioni a lungo termine servono per analizzare il futuro remoto, coprendo un arco temporale che va da uno a nove mesi. A differenza degli altri moduli di previsione a breve scadenza, questo strumento non fornisce dati precisi per il singolo giorno, ma offre delle stime sulle tendenze stagionali, indicando ad esempio se un'estate sarà mediamente più calda o una stagione più piovosa rispetto alla norma storica.

Per il funzionamento di questo tool, il sistema utilizza i dati forniti dal Copernicus Climate Service (CDS), prendendo nello specifico il modello ECMWF. Poiché i dati delle proiezioni stagionali vengono rilasciati con cadenza mensile, a ogni invocazione il tool dell'agente fa riferimento all'ultimo dataset pubblicato, il quale rimane invariato per l'intera durata del mese in corso. Questo dataset consiste in una collezione di proiezioni statistiche su base mensile per i successivi nove mesi. Al suo interno sono contenute le stime probabilistiche e lo scostamento previsto rispetto alla norma per numerosi parametri (ad esempio, la temperatura media, le precipitazioni cumulate, la velocità del vento e la copertura nuvolosa media).

Su questi dati grezzi, il tool applica un'analisi statistica basata sull'hindcast, una procedura necessaria per correggere l'errore sistematico tipico di ogni modello climatico.

Nelle previsioni stagionali, infatti, lo scopo non è prevedere il valore meteorologico assoluto (es. prevedere esattamente 25.4°C tra tre mesi), che sarebbe del tutto impossibile a così distanza. Lo scopo è calcolare l'anomalia climatica, ovvero stimare se un mese sarà più caldo o più piovoso rispetto alla normalità, e di quanto.

Tuttavia, ogni modello matematico tende a sovrastimare o sottostimare i valori. Se il modello avesse la tendenza a prevedere sempre 1°C in meno rispetto alla realtà, confrontare la sua proiezione futura direttamente con i dati storici reali produrrebbe un'anomalia falsata per il difetto del modello.

L'hindcast risolve il problema in questo modo: la proiezione futura non viene confrontata con le misurazioni storiche reali, ma con l'archivio delle simulazioni passate generate dallo stesso modello (la sua "storia interna"). In questo modo si corregge l'errore che il modello tende a ripetere sempre, mentre rimane la parte di incertezza legata all'imprevedibilità dell'atmosfera. Inoltre, si mantiene costante il tempo di anticipo (lead time): ad esempio, una proiezione a quattro mesi viene confrontata solo con le simulazioni passate fatte con lo stesso anticipo. In questo modo si tiene conto dell'aumento di incertezza dovuto al passare del tempo.

Se il valore attuale si discosta significativamente da questa media storica interna del modello, l'agente rileva un'anomalia reale. Data la caoticità dell'atmosfera sul lungo periodo, queste stime vengono usate dall'agente solo come supporto strategico generale e mai per decisioni operative di dettaglio.

E' importante sottolineare che questa correzione tramite hindcast è una prerogativa fondamentale delle previsioni stagionali e climatiche. Nei modelli a breve e medio raggio, infatti, l'errore sistematico non fa in tempo ad accumularsi poiché i sistemi vengono costantemente ricalibrati ogni poche ore tramite l'assimilazione di dati reali. Invece, un modello stagionale simula il meteo a distanza di mesi senza poter assimilare nuovi dati in corso d'opera.

Nonostante queste correzioni statistiche, è fondamentale sottolineare che i risultati forniti da questo strumento devono essere interpretati con estrema cautela. A causa della natura intrinsecamente caotica dell'atmosfera, l'affidabilità delle proiezioni a così lungo termine è nettamente inferiore rispetto a quella delle previsioni a breve o medio raggio.

Raccolta dati grezzi per altri agenti

A differenza degli strumenti previsionali descritti in precedenza, che producono risposte testuali sintetiche, questo modulo ha come unico scopo quello di raccogliere un insieme di variabili meteorologiche (temperatura, umidità, vento, copertura nuvolosa e altre) per renderle disponibili ad altri agenti del sistema, in quanto vengono caricate nel Working storage. L'agente chiamante potrà quindi recuperare la totalità delle variabili salvate e selezionare quelle che utili per lo svolgimento del suo task. Questo approccio disaccoppia la responsabilità di raccolta delle informazioni dall'uso di quei dati.

A titolo di esempio, l'agente Hiking attinge a dati grezzi quali la pendenza del terreno, l'esposizione del versante e l'umidità del suolo per il funzionamento del suo modello fisico di valutazione. Queste variabili con dati orari sulla qualità dell'aria (AQI) e sull'indice UV, permettono all'agente di calcolare parametri scientifici come l'indice UTCI (Universal Thermal Climate Index) e determinare con precisione la sicurezza e la praticabilità di un sentiero. Questo rappresenta solo un caso d'uso: questo tool permette a qualsiasi altro agente di richiedere un set di variabili meteorologiche che vuole ricevere, senza che l'agente meteorologico debba conoscere le logiche di come vengono usate.

10.3 Rapporto con gli altri agenti

Il Weather Agent si relaziona direttamente con tutti gli agenti posti al di sopra di esso nella gerarchia del sistema, ad esclusione dell'Orchestratore. Questa sua trasversalità nel sistema permette di comprendere la fattibilità e sicurezza dell'intero viaggio.

- **Travel Agent:** Il Travel Agent può invocare il weether Agent in modo diretto, senza la necessità di passare attraverso gli agenti specialisti. Questa interazione avviene tipicamente per ottenere una valutazione climatica generale su una destinazione in un determinato periodo, permettendo al Travel Agent di effettuare considerazioni ad alto livello prima di utilizzare altri strumenti.
- **Hiking Agent:** Interagisce strettamente con il Weather Agent poiché le condizioni atmosferiche sono un parametro fondamentale per la sicurezza e la fattibilità delle escursioni. L'Hiking Agent lo interroga anche per ottenere previsioni mirate (come probabilità di precipitazioni, temperature o allerte specifiche), utilizzandole per filtrare i sentieri, sconsigliare percorsi pericolosi in caso di maltempo o suggerire l'equipaggiamento più adeguato.
- **Activity Agent:** L'Activity Agent usa i dati meteorologici per scegliere le attività da consigliare all'utente in modo intelligente, ad esempio, consigliare le attività all'aperto durante i momenti di bel tempo (visite a parchi o tour cittadini) e favorendo le visite a musei o a strutture al coperto nelle giornate o negli orari in cui è prevista pioggia.

Capitolo 11

Location Agent

11.1 Introduzione

Il Location Agent opera al livello "Data Expert" della gerarchia di sistema, con il compito principale di reperire ed elaborare dati geospaziali all'interno dell'area geografica individuata dall'Orchestratore. L'agente è abilitato alla ricerca di qualsiasi tipologia di Punto di Interesse (POI) all'interno dell'area stabilita.

11.2 Architettura del Database Geospaziale

Il Location Agent funge da servizio dati geospaziale per l'intero sistema: sono gli altri agenti a delegargli le ricerche, invece di gestire in proprio archivi geografici. Per garantire scalabilità e prestazioni elevate, il suo database è organizzato come un Data Lake residente su Amazon S3 in formato GeoParquet, che contiene sei macro-categorie di punti di interesse (ristorazione, alloggi, negozi, attrazioni e intrattenimento, elementi naturali e servizi). I file sul cloud non vengono caricati integralmente in memoria, ma interrogati direttamente tramite query SQL spaziali eseguite con il motore DuckDB, che estrae di volta in volta soltanto i dati necessari.

Questa gestione permette di superare i limiti dei database spaziali tradizionali grazie a tre meccanismi:

- **Memorizzazione colonnare:** a differenza dei database tradizionali a memorizzazione per riga, il formato GeoParquet organizza i dati per colonna. Quando una query richiede solo alcuni campi (ad esempio il nome e le coordinate dei POI, o il filtro dei ristoranti per tipo di cucina), il sistema legge esclusivamente le colonne interessate, riducendo drasticamente le operazioni di lettura e i tempi di risposta.

- **Partition pruning per regione:** per evitare di scansionare l'intero archivio, i dati sono suddivisi fisicamente in cartelle su S3 secondo lo schema Hive, fino al livello regionale (es. `country=IT/admin_level_1=piemonte/`). In fase di interrogazione DuckDB apre e legge unicamente i file delle regioni d'interesse, ignorando completamente il resto del database. I livelli amministrativi più specifici, cioè la provincia e comune, non generano invece cartelle separate: farlo produrrebbe decine di migliaia di micro-file su S3, rallentando il sistema a causa della latenza necessaria ad aprire ciascun file. Per questo restano memorizzati come semplici colonne di attributi all'interno del file regionale.
- **Filtro geometrico:** il partizionamento porta ad aprire il file della regione, che contiene però tutti i POI di quell'intera regione. Applicando la geometria ricevuta, cioè il poligono dell'area (trovata dall'orchestratore) vengono restituiti soltanto i POI le cui coordinate ricadono entro quel confine. Nel caso di una geometria specifica, come un'isocrona o un corridoio con buffer vengono usate quest'ultime piuttosto che il poligono.

11.3 Gestione di operazioni geospaziali

Quando il Location Agent viene invocato esso identifica la tipologia della geometria di riferimento ricevuta, classificandola come punto singolo, percorso (lineare) o area definita (poligonale).

In base a questa classificazione, l'agente determina la strategia per definire lo spazio di ricerca:

- **Percorso:** viene applicata un'operazione di buffering per generare un'area di ricerca adiacente al tracciato, consentendo l'individuazione dei servizi lungo la via.
- **Punto singolo:** l'area di ricerca può essere definita in due modi. Il primo, più semplice, applica un buffering attorno al punto per ottenere un'area circolare; il secondo, più accurato, genera un'isocrona che delimita l'area effettivamente raggiungibile in base ai tempi di percorrenza dal punto di origine.
- **Area definita:** la geometria poligonale viene utilizzata così com'è per la ricerca di POI al suo interno.

Una volta definita l'area di ricerca, l'agente interroga il database per estrarre i punti di interesse (POI). Vengono selezionati solo i POI che si trovano fisicamente dentro i confini di quest'area, escludendo tutti quelli esterni.

11.4 Strumenti a disposizione

Per svolgere i propri compiti, l'agente dispone di specifici strumenti di analisi spaziale:

- **Ricerca di Punti di Interesse (POI):** Interroga un database geospaziale in formato GeoParquet con milioni di elementi già categorizzati. Il filtro è solo geometrico: usa l'area definita prima come confine e tiene solo i POI che ci ricadono dentro.
- **Buffering geografico:** Operazione applicabile esclusivamente a geometrie lineari (percorsi) per generare un'area di ricerca circostante. Lo strumento traccia un corridoio che si sviluppa lungo entrambi i lati del tracciato, consentendo di individuare i punti di interesse situati nelle immediate vicinanze del percorso stesso.
- **Generazione di Isocrone:** Calcola l'area raggiungibile a partire da un punto di origine entro una determinata soglia temporale (ad esempio, 10 o 15 minuti), considerando diverse modalità di trasporto (a piedi o in auto). Questa analisi si basa sulla rete stradale e pedonale. Rispetto a una ricerca basata su un semplice raggio geometrico in linea d'aria, questo approccio permette di escludere quei POI che, pur trovandosi a breve distanza lineare, risultano di fatto irraggiungibili nel tempo selezionato.

11.5 Rapporto con gli Altri Agenti

Il Location Agent riceve sempre dall'Orchestratore la geometria generale del contesto geografico (come i confini di un comune) e vi lavora come area di riferimento predefinita. Ecco nello specifico come questo agente collabora con gli altri agenti del sistema:

- **Travel Agent**

Il Travel Agent non esegue ricerche geografiche in autonomia, ma ha un ruolo di coordinatore nel dominio turistico. Quando necessita di informazioni su servizi o strutture presenti nell'area, delega il compito al Location Agent, che opera all'interno dell'area geografica generale già definita dall'Orchestratore cercando i servizi rilevanti entro quei confini (non è necessario invocare l'Activity Agent in quanto si cercano POI specifici).

- **Hiking Agent**

L'Hiking Agent gestisce la pianificazione delle escursioni e delega al Location Agent il compito di trovare i servizi utili lungo il percorso, come ristoranti, rifugi, parcheggi di partenza o fonti d'acqua. Per farlo, salva la traccia GPS del

sentiero come Focus Geometry nello spazio cloud, creando così una geometria di tipo “linea” e passa al Location Agent solo il relativo codice identificativo. In quel caso, il Location Agent ignora l’area geografica generale definita dall’Orchestratore e lavora esclusivamente sul tracciato ricevuto. Ricevuto l’ID, costruisce automaticamente un corridoio poligonale attorno all’intera linea e cerca i servizi che ricadono al suo interno.

- **Activity Agent** L’Activity Agent si occupa di suggerire attività personalizzate all’utente in base alle sue preferenze e alle condizioni del momento. Il Location Agent viene invocato per richiedere informazioni di POI nelle vicinanze ad esempio utilizzando lo strumento dell’isocroma. Può quindi non solo validare la raggiungibilità di un luogo ma anche cercare altri punti di interesse.

11.5.1 Agente dedicato piuttosto che un servizio commerciale

La scelta di costruire un agente di localizzazione su misura, capace di consultare direttamente i propri archivi geografici è una decisione progettuale ben precisa. Si potrebbe utilizzare dei servizi commerciali standard, come Google Maps, per le ricerche spaziali ma questi sistemi presentano dei limiti. Quest’ultimi, infatti, sono pensati soprattutto per ricerche semplici, limitandosi a trovare i luoghi che si trovano entro un certo raggio da un punto centrale. Faticano invece a gestire richieste che coinvolgono forme complesse del territorio. Ad esempio, non permettono di cercare facilmente dei servizi lungo un itinerario giornaliero fissato o anche all’interno di un “corridoio”. Avere un agente dedicato permette invece di superare questi limiti e di analizzare lo spazio in modo più flessibile.

Capitolo 12

Limitazioni, sviluppi futuri e conclusioni

12.1 Limitazioni

Nel capitolo relativo ai fondamenti teorici sono state trattate le limitazioni intrinseche che caratterizzano le architetture multi-agente in generale. In questa sezione, invece, l'attenzione si sposta sulle difficoltà pratiche e sui limiti incontrati durante lo sviluppo di questo specifico progetto. Se da un lato la scelta di questa architettura permette di gestire richieste complesse, dall'altro introduce un inevitabile aumento della latenza globale. Questo ritardo nell'esecuzione è causato da due fattori principali: in primo luogo, la concatenazione delle chiamate gerarchiche obbliga gli agenti superiori ad attendere il completamento di quelli inferiori. In secondo luogo, la natura "stateless" del sistema impone continui tempi di lettura e scrittura per recuperare e salvare i dati su Amazon S3 a ogni turno. Oltre a questi rallentamenti, a causa dei continui scambi di messaggi tra i diversi livelli, l'uso dei token può ancora risultare dispendioso.

Il meccanismo della gestione della memoria è stato concepito per sostenere la natura stateless del sistema, ma la sua implementazione si è rivelata particolarmente complessa, poiché richiede numerosi passaggi tra i vari livelli, quali il payload, lo scaricamento e salvataggio su S3 e la loro corretta interpretazione da parte di ciascun agente. In un'architettura di questo tipo, capire con precisione chi invia cosa e a chi risulta fondamentale per il corretto funzionamento dell'intero sistema. Proprio per questo, apportare modifiche al meccanismo è di per sé complesso e comporterebbe un cambiamento globale, dato che coinvolge tutti gli agenti, con la sola eccezione dell'orchestratore. A questo si aggiunge un ulteriore limite comportamentale: non

sempre gli agenti riconoscono che sarebbe più efficiente recuperare l'archivio interno prodotto nel turno precedente, e capita che, invece di riutilizzarlo, ripetano interamente l'esecuzione, con un evidente spreco di tempo e risorse. Questa criticità è ulteriormente aggravata dalla necessità di mantenere la coerenza in conversazioni su più turni all'interno della gerarchia. Di conseguenza, il meccanismo creato per gestire questo continuo scambio di dati si è rivelato a volte fragile, specialmente quando gli agenti devono passarsi specifiche informazioni. Per questi motivi, la soluzione adottata non può essere considerata la più efficiente in assoluto, ma è un compromesso per unire lo stateless degli agenti con la gestione di grandi quantità di dati.

Dato che la gestione della geometria spaziale nel meccanismo della memoria è complessa e delicata, si è dovuto imporre un vincolo che limita l'utilizzo a una sola geometria spaziale di base per intera conversazione. Il registro globale conserva infatti un unico codice identificativo per la zona geografica, questo significa che l'utente può fare domande di approfondimento sulla zona scelta, ma non può chiedere improvvisamente di confrontarla con una nuova località all'interno dello stesso dialogo, poiché l'architettura attuale non permette di gestire due riferimenti spaziali contemporaneamente.

12.2 Sviluppi futuri

Il sistema descritto in questa tesi pone le basi per un'architettura robusta, ma presenta margini di miglioramento. Come già accennato, l'intera architettura è stata progettata per essere estremamente flessibile, quindi aggiungere nuovi agenti esperti di altri domini è molto facile. E' sufficiente includerli all'interno del livello dei generalisti, lasciando all'orchestratore il compito dello smistamento semantico. Un esempio pratico di questa modularità è rappresentato dal Retail Agent, già presente nella progettazione iniziale ma non approfondito in questa sede per limiti di tempo. Il suo sviluppo permetterebbe al sistema di estendere il proprio dominio oltre l'ambito turistico, includendo anche l'aspetto commerciale. Lo stesso vale per l'aggiunta di ulteriori agenti in domini differenti.)

Un ulteriore margine di miglioramento riguarda il superamento dell'attuale rigidità spaziale del sistema. Come discusso tra le limitazioni, l'architettura attuale vincola l'utente conversare su un'unica area geografica per l'intera durata della conversazione. La geometria viene definita in fase di inizializzazione e rimane fissa fino al termine dell'interazione, senza possibilità di sovrascrittura o affiancamento. Un'evoluzione naturale del sistema di gestione della memoria consisterebbe nel supportare la possibilità di gestire più geometrie all'interno della stessa sessione. Questo permetterebbe all'utente di poter spostare la sua attenzione da una località all'altra, oppure confrontare più destinazioni.

Dal punto di vista puramente tecnologico, potrebbe essere utile includere nel sistema i restanti moduli offerti da AgentCore. L'introduzione del modulo Browser doterebbe gli agenti della capacità di navigare autonomamente sul web, consentendo loro di verificare aggiornamenti in tempo reale che non sono presenti nei database pre-calcolati. Includere il modulo Observability permetterebbe di tracciare, fare debug e monitorare le performance degli agenti in modo molto più intuitivo. Tenendo anche traccia di metriche come: latenza, token utilizzati e error rate.

Infine, va detto che finora l'obiettivo principale è stato costruire le fondamenta del sistema e mostrarne la fattibilità tecnica. A causa dei limiti di tempo, la fase di verifica e validazione non è stata approfondita quanto necessario. Sarà quindi indispensabile, in un lavoro futuro, estendere la fase di testing, individuare nuove strategie per ridurre la latenza e adottare una gestione più robusta dei casi limite,

12.3 Conclusioni

Il presente lavoro di tesi ha avuto come obiettivo la costruzione delle fondamenta di un sistema multi-agente gerarchico applicato al dominio turistico, realizzato nel corso del tirocinio svolto presso l'azienda Data Reply. È bene chiarire fin da subito la natura di quanto ottenuto: non si tratta di un prodotto finito, pronto per essere messo in produzione, bensì di un'architettura funzionante che dimostra come sia possibile coordinare più agenti specializzati per affrontare richieste che un singolo modello linguistico faticerebbe a gestire in modo affidabile. Il risultato va quindi inteso come una base solida, sulla quale il progetto potrà essere ampliato e perfezionato in futuro.

Il turismo, scelto come banco di prova per le ragioni esposte nell'introduzione, si è confermato un dominio adatto a questo tipo di architettura. Trattandosi di un ambito che per sua natura mette insieme dati eterogenei, come distanze, meteo, luoghi di interesse e preferenze personali, la scomposizione del problema tra competenze distinte si è rivelata la strategia più naturale di procedere. In questo senso l'organizzazione a livelli, adottata per evitare il disordine e la saturazione del contesto che deriverebbero da una comunicazione libera tra tutti gli agenti, ha effettivamente prodotto i benefici attesi: un flusso di dati tracciabile e finestre di contesto contenute.

Ciò che è emerso durante lo sviluppo è che il valore della gerarchia sta soprattutto nel ruolo distinto assegnato a ciascun livello. L'Orchestratore, agendo da primo filtro semantico, permette di localizzare la zona geografica su cui si è interessati per far sì che gli altri agenti del sistema la possano utilizzare. Inoltre, è il responsabile di indirizzare la richiesta dell'utente verso l'agente generalista più adatto. Gli agenti generalisti, come il Travel Agent hanno il ruolo di coordinatori per gestire il dominio turistico, delegando i sotto-compiti agli specialisti e agli esperti dei dati, ovvero Acti-

vity Agent, Hiking Agent, Weather Agent e Location Agent. Il risultato più concreto di questa separazione è che i prompt di ciascun agente sono rimasti brevi e mirati, e che i risultati di ciascun agente man mano risalgono la gerarchia trasformandosi da informazioni grezze in sintesi via via più raffinate, fino alla risposta ricomposta dall'Orchestratore.

La parte più impegnativa, e quella che ha richiesto il maggior numero di compromessi, è stata la gestione della memoria. Per evitare che lo scambio diretto di grandi quantità di dati, come migliaia di punti geografici o previsioni meteorologiche dettagliate, saturasse la finestra di contesto, si è scelto di far operare il sistema in modalità stateless: ogni agente non conserva nulla e riparte da zero a ogni invocazione, mentre la persistenza dei dati è affidata ad Amazon S3, che funge da archivio condiviso. Nei pacchetti scambiati tra i livelli non transitano i file completi, ma soltanto i riferimenti che ne consentono il recupero quando servono. Inoltre questo stesso meccanismo rende possibile la gestione delle conversazioni su più turni con l'utente. Questa soluzione ha permesso di limitare il problema dovuto a un'eccessiva quantità di dati nella finestra di contesto, ma ha mostrato anche la sua fragilità: coordinare correttamente chi salva, chi legge e cosa viene passato tra i livelli si è rivelato delicato, al punto che in alcuni casi gli agenti tendono a ripetere le elaborazioni già svolte invece di riutilizzare i risultati disponibili. Il modo in cui viene gestita la memoria resta un compromesso ragionato tra la natura stateless degli agenti e la necessità di trattare grandi volumi di dati. Sul piano tecnologico, l'orchestrazione dei flussi è stata affidata a LangGraph e l'esecuzione degli agenti all'infrastruttura serverless di AWS Bedrock AgentCore, che attiva ciascun agente solo quando serve e semplifica così la scalabilità.

Un punto merita particolare attenzione, perché condiziona l'impiego pratico del sistema. Un'architettura di questo tipo resta esposta al rischio tipico dei modelli linguistici, ovvero produrre risposte che appaiono corrette ma non sono fondate sui fatti. In un contesto turistico fornire al cliente un'informazione plausibile ma errata, come un orario inesistente o un percorso non praticabile, comprometterebbe l'utilità stessa dello strumento. Per questa ragione il sistema è pensato per operare con un supervisore umano (Human-in-the-Loop), incaricato di validare le risposte prima che vengano consegnate al cliente. E' proprio questa verifica a conferire al sistema la affidabilità necessaria per un impiego reale: l'architettura raccoglie, coordina e sintetizza le informazioni, mentre la persona garantisce che il risultato finale sia affidabile.

I limiti riscontrati aiutano a definire con onestà l'ambito in cui questo approccio ha senso. La latenza cresce inevitabilmente rispetto all'impiego di un singolo agente, questo perché ogni livello aggiunge tempo di ragionamento e di attesa, a cui si somma quello degli accessi ad Amazon S3. Inoltre, il consumo di token è più elevato, dal momento che attivare a cascata più modelli, ciascuno con il proprio prompt e il

proprio ciclo ReAct, costa più di un singolo agente. L'osservabilità, infine, resta problematica poiché individuare la causa di un errore richiede di risalire l'intera catena di invocazioni. Nessuno di questi aspetti invalida la soluzione, ma insieme chiariscono che un sistema simile conviene quando la correttezza e l'affidabilità delle risposte valgono il costo aggiuntivo in tempo e risorse.

Al di là dei limiti appena descritti, il sistema offre ampi margini di ampliamento. Il completamento del Retail Agent, rimasto abbozzato, è il passo più immediato per estendere il sistema oltre l'ambito turistico, mentre un livello di tracciamento dedicato, anche appoggiandosi ai moduli aggiuntivi di AgentCore, renderebbe più agevole il monitoraggio. Restano poi da affrontare il superamento della rigidità che vincola l'utente a una sola area geografica per conversazione. Più in generale, la struttura sviluppata non è legata al solo turismo: è sufficientemente astratta da poter essere trasferita a qualunque contesto che richieda il coordinamento di competenze eterogenee.

Nel complesso, il lavoro conferma che un'organizzazione gerarchica si adatta bene al dominio turistico, offrendo un modo ordinato di scomporre richieste complesse e ricomporne i risultati. Quanto realizzato non pretende di essere una soluzione definitiva, ma costituisce una fondazione coerente, sulla quale sarà possibile continuare a costruire, ampliando i domini trattati, irrobustendo il meccanismo di memoria e rafforzando le verifiche, fino a renderlo pronto per un impiego in contesti reali.

Bibliografia

- [1] Salvatore Raieli e Gabriele Iuculano. *Building AI Agents with LLMs, RAG, and Knowledge Graphs*. Packt Publishing, 2025.
- [2] Michael Albada. *Building applications with AI agents*. O'Reilly, 2025.
- [3] *How Does Memory for AI Agents Work?* URL: <https://www.decodingai.com/p/how-does-memory-for-ai-agents-work>.
- [4] *ReAct Agent: Guide to understand its functionalities and create it from scratch*. URL: <https://www.plainconcepts.com/react-agent-ai/>.
- [5] *Building Production ReAct Agents From Scratch Is Simple*. URL: <https://www.decodingai.com/p/building-production-react-agents>.
- [6] *Building Collaborative AI: A Developer's Guide to Multi-Agent Systems with ADK*. URL: <https://cloud.google.com/blog/topics/developers-practitioners/building-collaborative-ai-a-developers-guide-to-multi-agent-systems-with-adk>.
- [7] 多智能体(Multi-Agent): AI時代の指揮官と合理分工, [Multi-Agente: Comandante e ragionevole divisione del lavoro nell'era dell'IA]. URL: <https://www.woshipm.com/ai/6215805.html>.
- [8] Feyza Duman Keles, Pruthuvi Maheshakya Wijewardena e Chinmay Hegde. «On The Computational Complexity of Self-Attention». In: *International Conference on Algorithmic Learning Theory*. 2022. URL: <https://api.semanticscholar.org/CorpusID:252198880>.
- [9] 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. DOI: 10.1145/3605764.3623985.
- [10] *Amazon Bedrock AgentCore Developer Guide*. URL: <https://docs.aws.amazon.com/bedrock-agentcore/latest/devguide/what-is-bedrock-agentcore.html>.
- [11] *Amazon Bedrock AgentCore Tutorials*. URL: <https://github.com/aws-labs/amazon-bedrock-agentcore-samples/tree/main/01-tutorials>.
- [12] ExchangeRate-API. *Free Exchange Rate API Documentation*. 2026. URL: <https://www.exchangerate-api.com/docs/free> (visitato il giorno 31/05/2026).
- [13] *Club Alpino Italiano*. URL: <https://www.cai.it/>.
- [14] Błażejczyk, K., Jendritzky, G., Bröde, P., Fiala, D., Havenith, G., Epstein, Y., Psikuta, A., Kampmann, B. *An introduction to the Universal Thermal Climate Index (UTCI)*. 2013. URL: https://www.researchgate.net/publication/239525964_An_introduction_to_the_Universal_Thermal_Climate_Index_UTCI (visitato il giorno 06/07/2026).

- [15] *Radiation: Ultraviolet (UV) radiation*. URL: [https://www.who.int/news-room/questions-and-answers/item/radiation-ultraviolet-\(uv\)](https://www.who.int/news-room/questions-and-answers/item/radiation-ultraviolet-(uv)).

Elenco delle figure

2.1	Livelli di memoria per un agente intelligente	15
2.2	Architettura per un agente ReAct	19
2.3	Architetture possibili per un multi-agente	24
2.4	Suddivisione dei token: input, reasoning e output [5].	31
3.1	Insieme dei componenti di Amazon Bedrock AgentCore. Immagine tratta da [11] sotto licenza Apache 2.0.	37
4.1	Architettura del sistema multi-agente	40
8.1	Itinerario giornaliero prodotto dall'Activity Agent: l'hotel e le tre esperienze selezionate, collegati nell'ordine di visita. I segmenti raffigurano schematicamente i collegamenti in auto tra una tappa e la successiva. .	76
9.1	Percorso SI E01	87
9.2	Profilo altimetrico SI E01	88

Elenco delle tabelle

5.1	Riepilogo dei termini chiave nella gestione della memoria del sistema multi-agente.	50
-----	---	----

Ringraziamenti

Ringrazio la mia famiglia a cui devo tutto, mi hanno sostenuto in ogni momento e in ogni mia scelta. Avere qualcuno su cui poter sempre contare è qualcosa che spesso viene dato per scontato nella vita quotidiana, ma che in momenti come questi è necessario fermarsi per poter ringraziare per tutti i sacrifici fatti. Mi rendo conto che nulla di quello che potrei mai scrivere in queste poche righe sia sufficiente per mostrare appieno quanto tutto ciò significhi per me. Spero che questo traguardo vi renda fieri di me, io invece mi impegnerò a dare a voi anche solo una piccola parte di quanto avete dato a me.

Ringrazio anche Daniele che durante questo periodo torinese mi è sempre stato a fianco, sopportando tutte le mie stranezze e trovando sempre il modo di rallegrarmi.