

CORSO AVANZATO SULL'INTELLIGENZA
ARTIFICIALE

AI²Life

ADVANCED
SCHOOL AI

Strumenti e tecniche per progettare e programmare una RAG

Dott. Matteo Gnocchi

Chi Sono



Matteo Gnocchi - Tecnologo
Consiglio Nazionale delle Ricerche
Area Territoriale della Ricerca di Milano 4
matteo.gnocchi@cnr.it

Laurea Magistrale in Tecnologie dell'Informazione
e della Comunicazione

- Master in Intelligenza artificiale e blockchain
- Master in Intelligenza artificiale per la pubblica amministrazione (in corso)
- Docente AS-AI

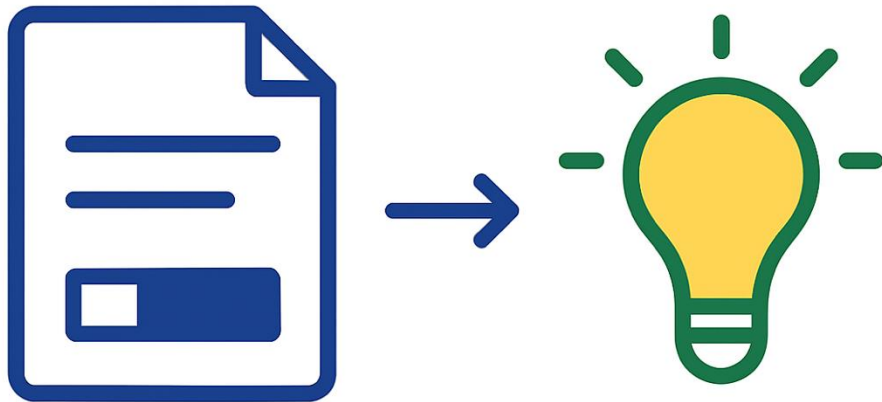


Studio, progettazione e sviluppo di Applicazioni basate sull'utilizzo di Intelligenza artificiale generativa e coordinamento nello sviluppo e integrazione di piattaforme e tecnologiche web

Premessa

L'obiettivo del webinar è quello di creare una consapevolezza sullo stato e i meccanismi alla base della metodica RAG utilizzata nell'ambito della AI generativa così da poterla utilizzare in modo consapevole.

Ci focalizzeremo sulle metodologie e sulle possibili soluzioni senza approfondire troppo lo specifico servizio che può implementarlo



Non si tratta di sostituire l'uomo, ma di affiancarlo, offrendo strumenti che aumentano creatività ed efficienza.

Indice



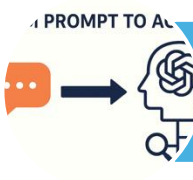
Large Language Model



Fine-Tuning



Retrieval Augmented Generation



Architetture



Ambienti di test e sviluppo

1. Large Language Model

GENERATIVE AI



Intelligenza Artificiale Generativa e LLM

L'intelligenza artificiale generativa è una branca dell'IA che si concentra sulla creazione di contenuti nuovi (testi, immagini, audio, codice, etc...) partendo da dati esistenti.



Il suo obiettivo è “generare” qualcosa di originale, non solo analizzare o classificare.

Generative Artificial Intelligence

Artificial Intelligence

Large
Language
Models

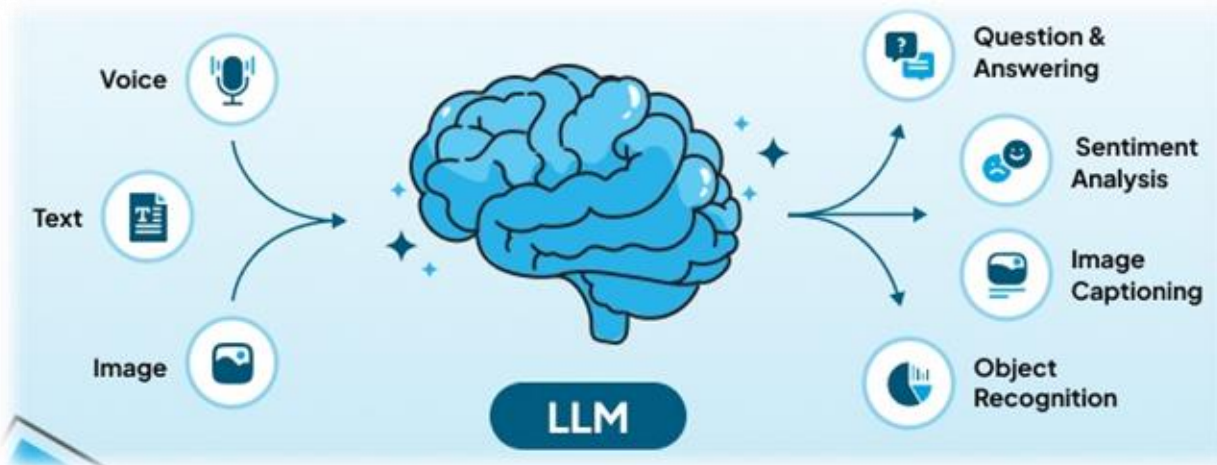
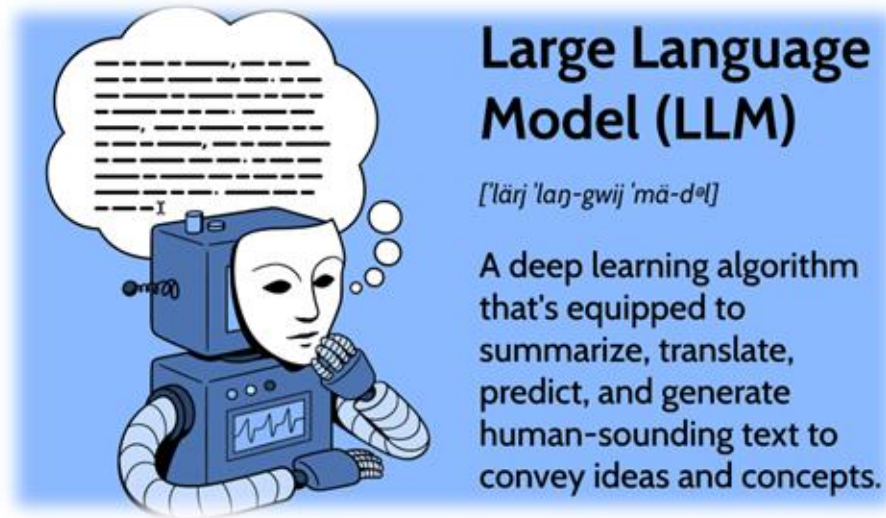
Transformers

Large Language Model (LLM)

Un Large Language Model (LLM) è un tipo di modello di rete neurale che è stato addestrato su una quantità massiva di dati testuali (miliardi o trilioni di parole provenienti da libri, articoli, siti web, ecc.).

Large (Grandi Dimensioni): Si riferisce sia al **numero di parametri** che rendono il modello estremamente complesso, sia alla **dimensione del set di dati di addestramento**

Language Model (Modello Linguistico): Indica che la funzione primaria del modello è quella di comprendere e generare un linguaggio naturale. L'obiettivo originale (e fondamentale) di questi modelli è prevedere la **parola successiva** in una sequenza, data la sequenza di parole che l'ha preceduta.



Architettura Transformers

Gli LLM si basano sulla architettura Transformers

Il **Transformer** è un'architettura di rete neurale introdotta nel 2017 con il paper *Attention is All You Need*. È progettata per elaborare sequenze di dati (come il testo) in modo **parallelo**, superando i limiti delle reti ricorrenti (RNN) che lavoravano in maniera sequenziale. Questo approccio consente di gestire **lunghe dipendenze** tra parole e di accelerare l'addestramento su grandi dataset



Architettura Transformers: Vantaggi

Parallelismo

elabora tutte le parole contemporaneamente rendendo l'addestramento più veloce

Modulo Encoder

Il modulo di codifica trasforma i dati in un formato che la rete neurale può comprendere e utilizzare

Modulo Decoder

Il modulo di decodifica elabora i dati codificati e genera una risposta

Meccanismo self-attention

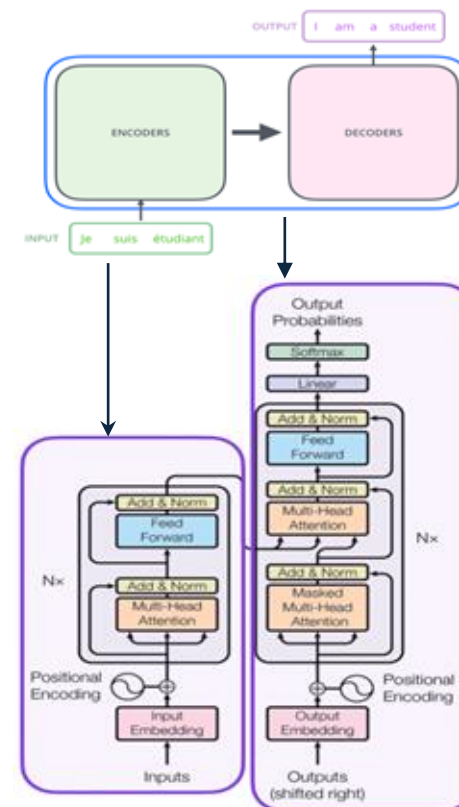
permette di catturare relazioni tra parole all'interno della sequenza di input

Scalabilità

gestisce miliardi di parametri

Versatilità

applicabile a NLP, visione artificiale, generazione di codice, multimodalità



LLM: Allucinazioni

Una allucinazione in un LLM è quando il modello genera una risposta che sembra plausibile e ben scritta, ma è falsa, imprecisa o inventata

In altre parole:

l'LLM "si inventa" informazioni invece di basarsi su fatti reali o sul contesto fornito.

Perché succede

Gli LLM non verificano i fatti e non "sanno" se qualcosa è vero.

Generano testo predicendo la sequenza di parole più probabile, non controllando una base di conoscenza

Domanda:

"Chi ha scritto Il nome della rosa?"

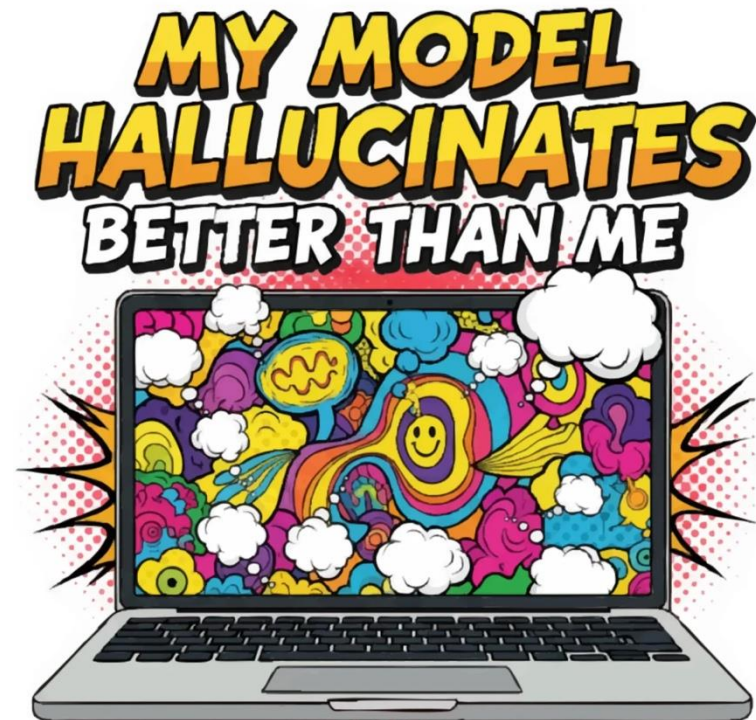


Risposta corretta: Umberto Eco



Allucinazione: "È stato scritto da Italo Calvino nel 1983"

La risposta è fluida e credibile, ma sbagliata.

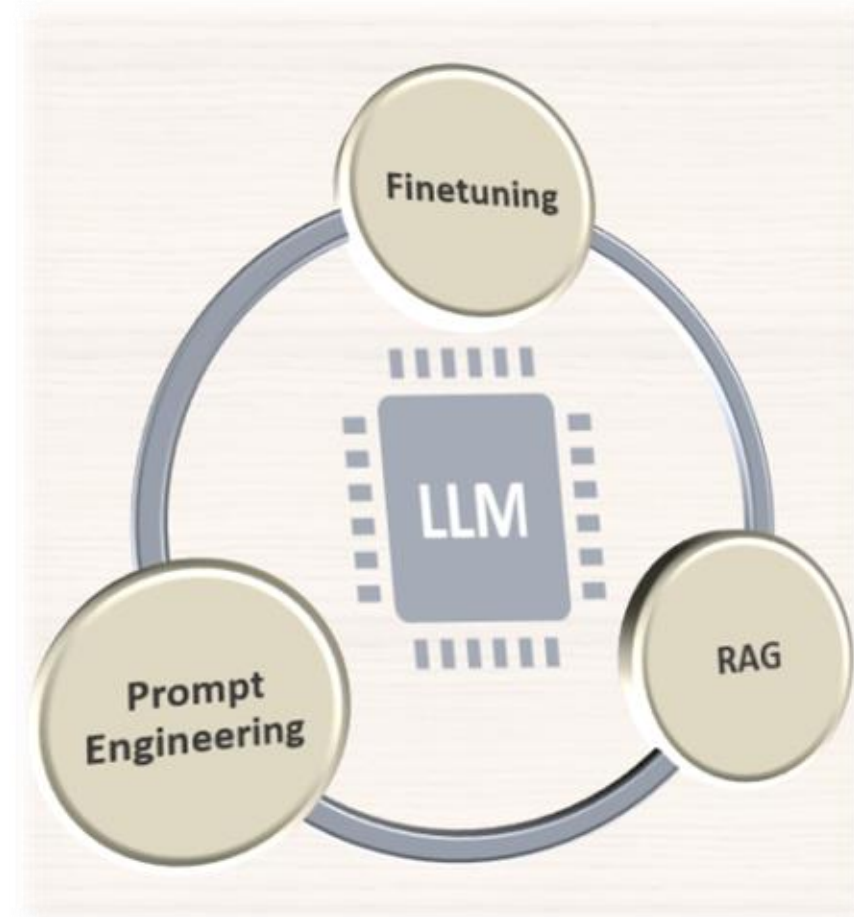


LLM: Miglioramento delle funzionalità

Per migliorare le funzionalità e la qualità delle risposte generate dai Large Language Model (LLM), vengono utilizzati diversi metodi, che vanno dalla semplice formulazione della domanda fino all'addestramento specialistico del modello stesso.

- Prompt Engineering (Miglioramento dell'Input)
- Fine-Tuning (Addestramento Aggiuntivo del Modello)
- Retrieval-Augmented Generation (RAG) (Aumento del Contesto)

Questi tre metodi non sono mutuamente esclusivi; spesso vengono **combinati** (ad esempio, usando RAG con un modello sottoposto a Fine-Tuning) per ottenere risultati ottimali in termini di accuratezza, pertinenza e costo-efficacia.

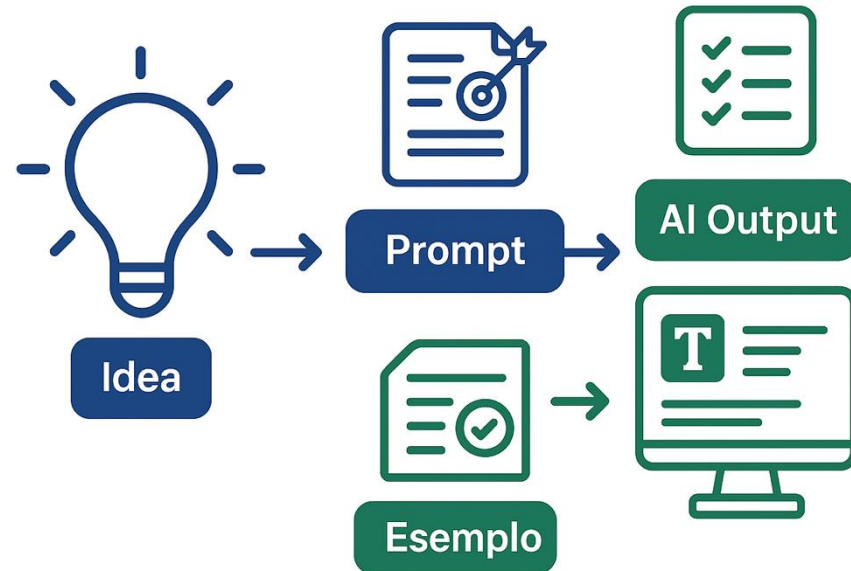


Prompt Engineering

Gli LLM generano testo in base al prompt fornito.

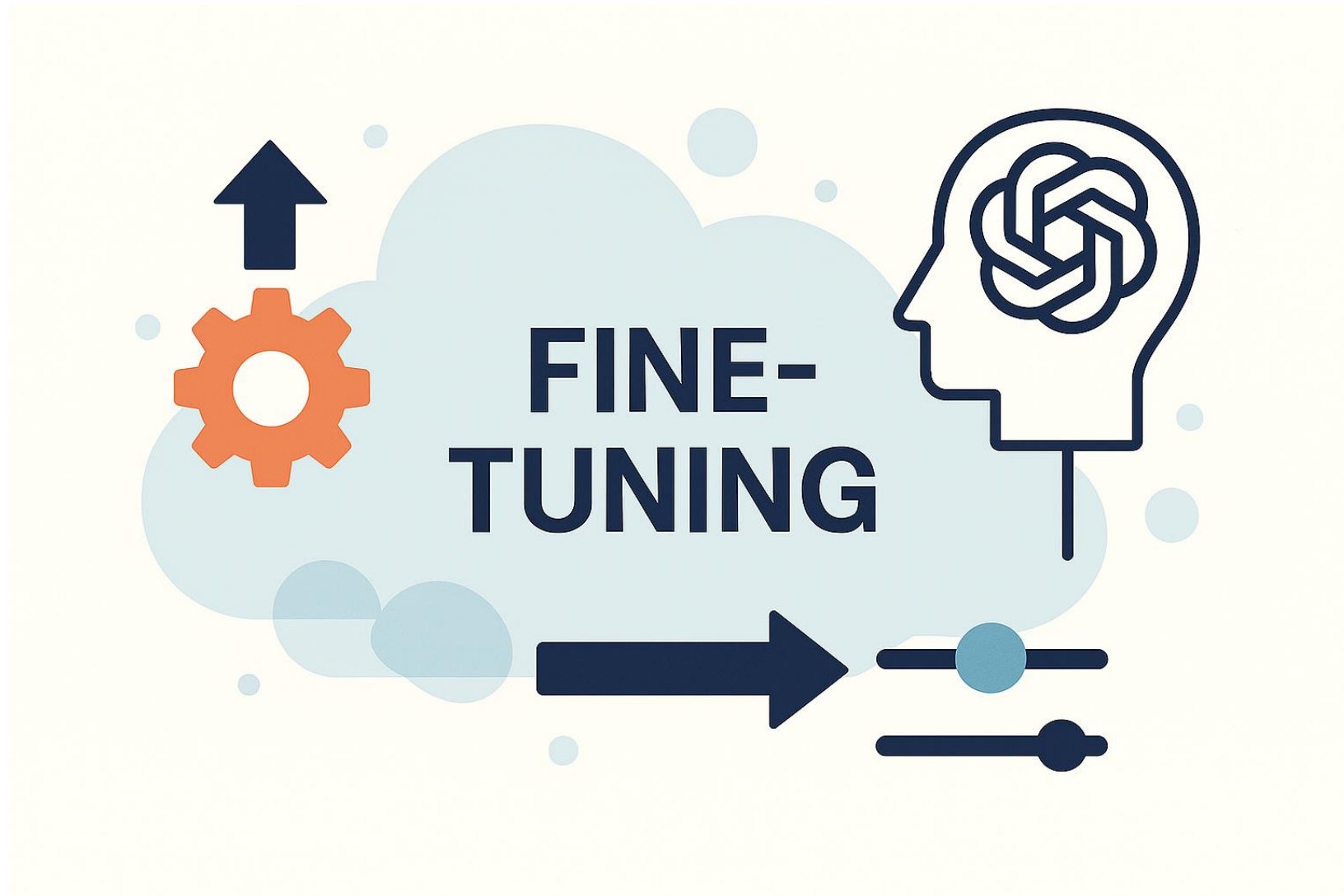
Il Prompt Engineering è la pratica di progettare e ottimizzare prompt per guidare i modelli LLM verso risposte accurate e utili.

è un insieme di istruzioni o input forniti per guidare la risposta del modello, aiutandolo a comprendere il contesto e a generare un output pertinente e coerente basato sul linguaggio utilizzato o rispondere a domande, completare frasi e seguire una conversazione.



Non richiede modifiche al modello

2. Fine-Tuning



Fine-Tuning: definizione

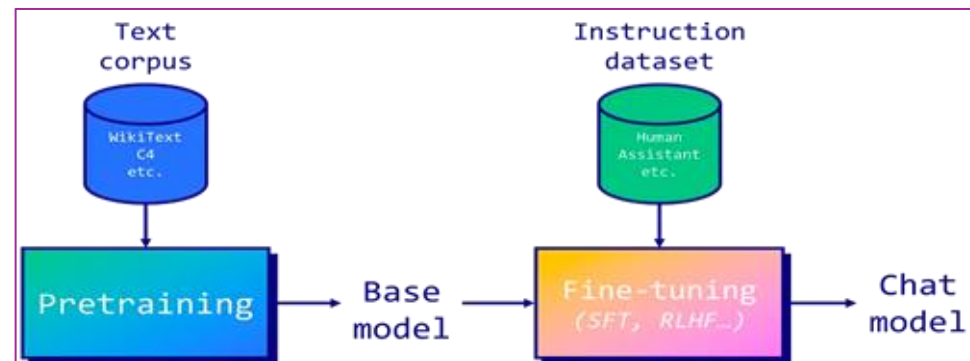


regolazione e ri-adattamento di un modello pre-addestrato

Adattamento permanente

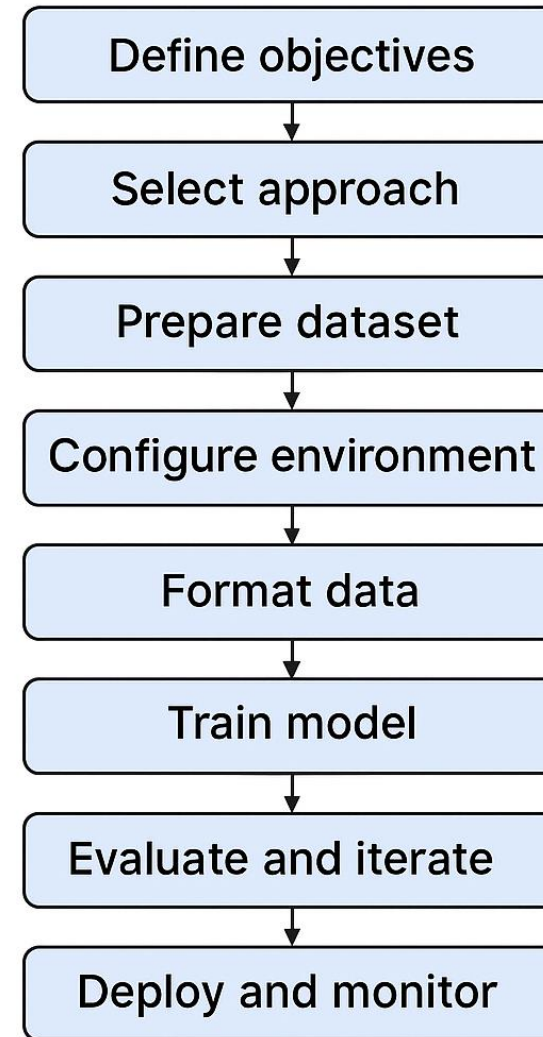
Il modello modifica permanentemente i pesi interni per integrare le nuove informazioni.

Si trasforma un modello generalista in uno specialista, adattandolo a un compito specifico con dati mirati.



Fine-Tuning: processo di funzionamento

1. **Modello di base:** Si parte da un modello generale che conosce bene il linguaggio ma non il tuo contesto specifico
2. **Dati specifici:** Si prepara un dataset rappresentativo del caso d'uso (dominio, stile, task)
3. **Addestramento mirato:** Il modello viene ri-addestrato, aggiornando alcuni parametri per imparare il nuovo comportamento
4. **Valutazione:** Si verifica che il modello risponda meglio al compito richiesto rispetto al modello generico
5. **Modello specializzato:** Il risultato è un modello più accurato, coerente e allineato al dominio



Fine-Tuning: vantaggi



Miglioramento delle Prestazioni e della Precisione

incremento della qualità e della pertinenza dell'output.

- **Massima Precisione sul Compito (Task Specificity)**
- **Migliore Generalizzazione sul Dominio**
- **Gestione di Formati Specifici (es Json, Markdown, atc ...)**

Controllo dello Stile e del Tono

permette di indirizzare il comportamento del modello in modi che il solo prompt engineering non può eguagliare.

- **Adozione di uno Stile Aziendale (Branding)**
- **Riduzione delle Allucinazioni**
- **Conformità e Sicurezza**

Efficienza

Le tecniche di fine-tuning moderno, offrono vantaggi significativi in termini di efficienza.

- **Costi di Inferenza Ridotti (il modello può essere più piccolo)**
- **Maggiore Velocità di Sviluppo (il set di dati può essere minore)**
- **Indipendenza dal Prompt Engineering (un modello ben affinato richiede prompt più semplici)**

Fine-Tuning: svantaggi



Economici e Computazionali

in particolare su modelli molto grandi, è un processo intensivo in termini di risorse.

- **Costi di Calcolo Elevati**
- **Requisiti di Memoria Elevati**
- **Costi di Archiviazione**

Rischi di Prestazioni e Apprendimento

Il processo non è sempre garanzia di successo e può introdurre problemi di performance

- **Dimenicanza Catastrofica: perde (in parte o del tutto) le conoscenze apprese in precedenza**
- **Overfitting (Sovrallenamento)**
- **Sensibilità agli Iperparametri**

Sfide Relative ai Dati

La qualità del fine-tuning dipende interamente dalla qualità del set di dati specifico.

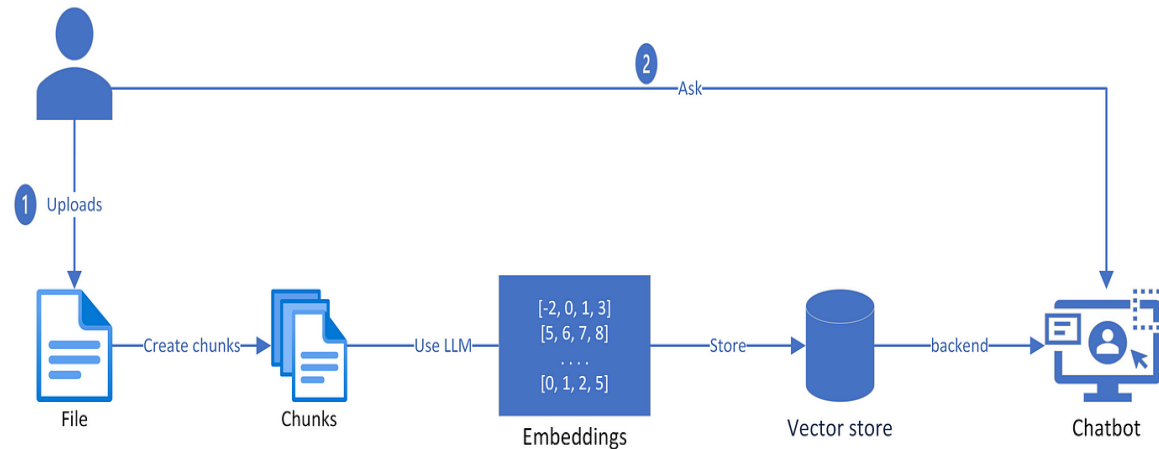
- **Necessità di Dati Etichettati e rilevanti**
- **Costo della Curatela (pulizia del dataset)**
- **Gestione della Versione (Versioning)**

3. RAG (Retrieval Augmented Generation)



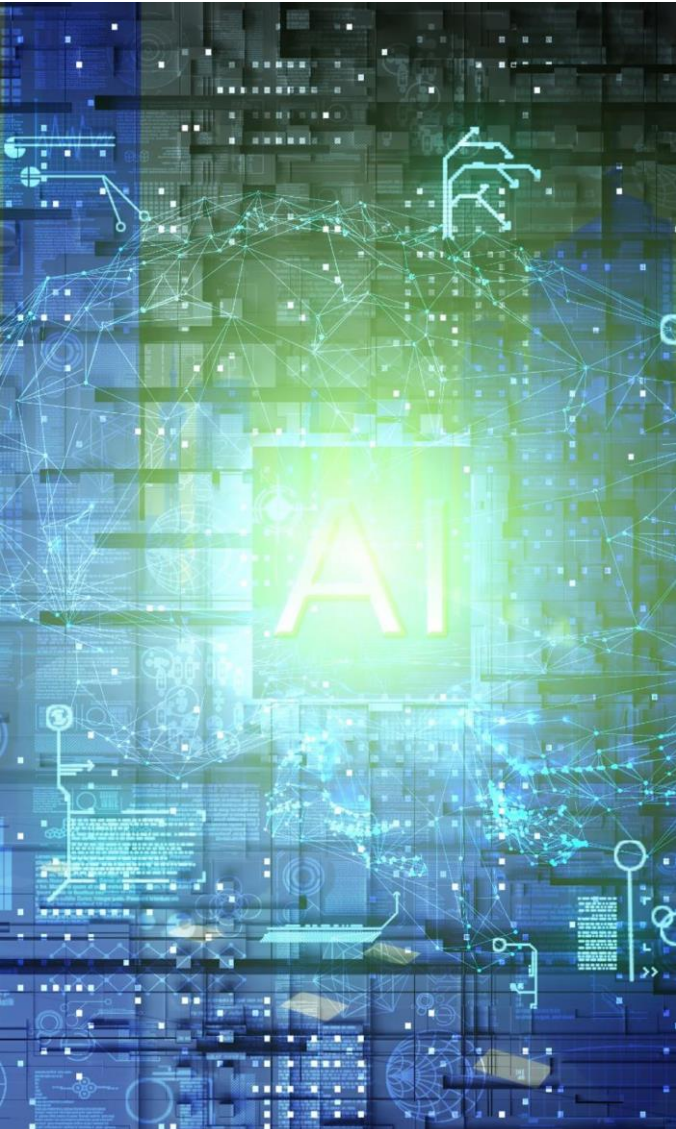
Retrieval Augmented Generation (RAG)

RAG (Retrieval-Augmented Generation) è una metodologia che combina i punti di forza dei tradizionali sistemi di recupero delle informazioni (come la ricerca e i database) con le capacità dei modelli linguistici generativi di grandi dimensioni (LLM). Combinando i dati e la conoscenza di uno specifico dominio di conoscenza con le competenze linguistiche del modello LLM la generazione delle risposte è più accurata, aggiornata e pertinente alle esigenze specifiche.



supera i limiti degli LLM statici mitigando le allucinazioni e costosi riaddestramenti con dati specifici e/o aggiornati.

RAG: Elementi chiave



Large Language Model (LLM)

Il LLM è responsabile della generazione del testo finale basato sul contesto fornito dal sistema.

Pipeline di indicizzazione

La pipeline suddivide i documenti in chunk, crea embedding e aggiunge metadati per migliorare la pertinenza dei risultati.

Retriever e database vettoriale

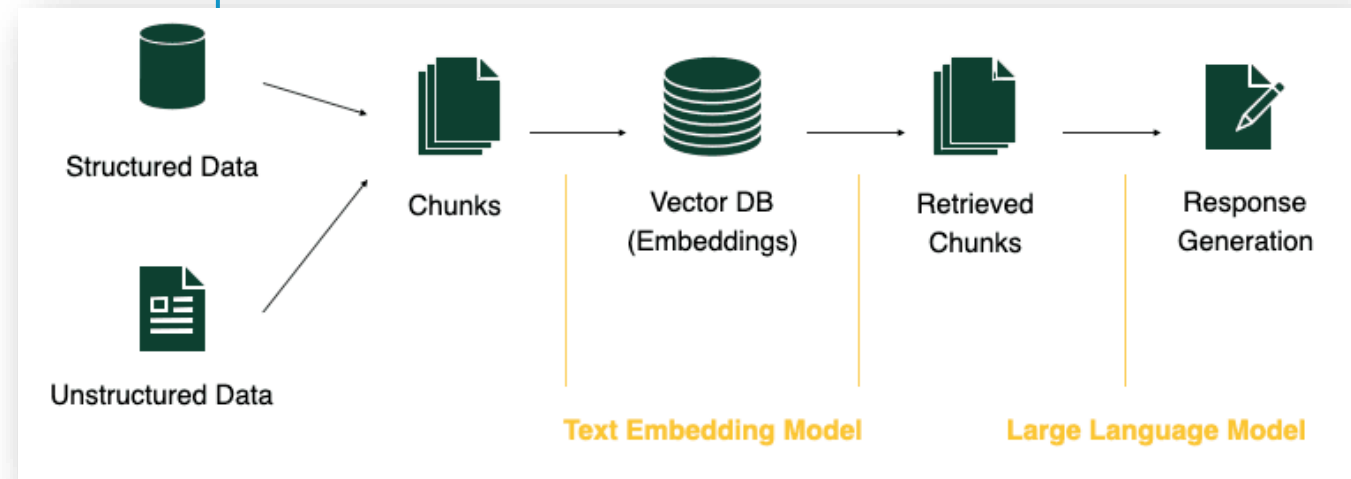
Il Retriever identifica i documenti più rilevanti usando un database vettoriale che archivia gli embedding per ricerche semantiche efficienti.

Orchestratore

L'orchestratore gestisce il flusso tra retrieval e generazione, integrando correttamente il contesto nel prompt.

RAG: Processo di funzionamento

1. Definizione del caso d'uso e dei requisiti
2. Analisi e selezione delle fonti di conoscenza
3. Pipeline di Indicizzazione
4. Chunking
5. Embedding
6. Database Vettoriale
7. Retriever
8. Orchestratore
9. Prompt Contestuale
10. Scelta e configurazione del modello LLM
11. Valutazione, testing e osservabilità
12. Deployment e integrazione applicativa



RAG: 1 Definizione caso d'uso e requisiti

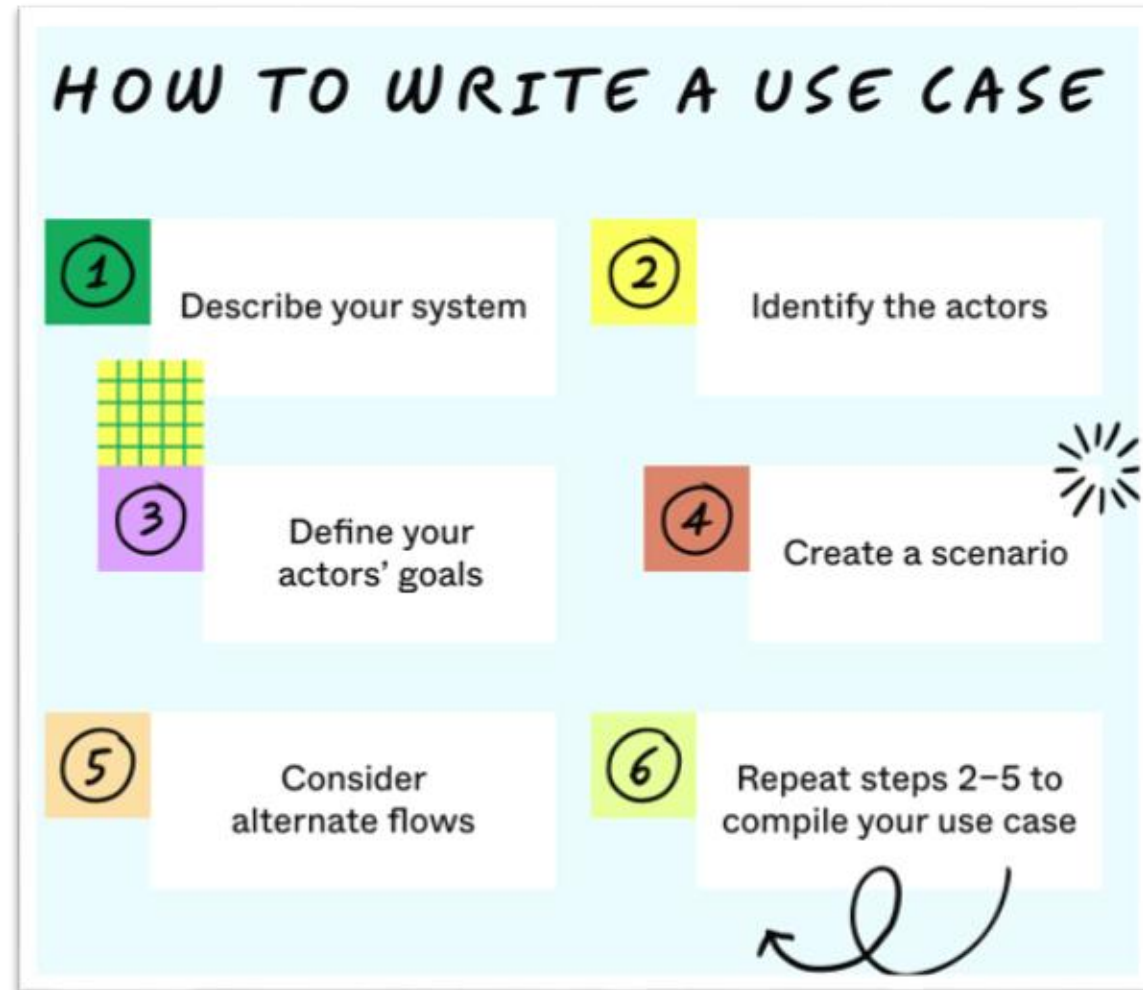
La progettazione RAG parte sempre dal *perché*. È fondamentale chiarire cosa deve fare il sistema e in quale contesto operativo.

Attività chiave

- Tipologia di utenti (interni, clienti, tecnici, decisori)
- Tipologia di domande (fact-based, procedurali, analitiche)
- Criticità di accuratezza, latenza e aggiornamento
- Vincoli di sicurezza, compliance e audit

Output

- Use case RAG formalizzato
- Requisiti funzionali e non funzionali



RAG: 2 Analisi e selezione delle fonti

RAG non “insegna” al modello, ma lo *ancora* a fonti esterne. La qualità delle risposte dipende dalla qualità delle fonti.

Attività chiave

- Identificazione delle sorgenti (PDF, wiki, SharePoint, DB, API)
- Valutazione di:
 - affidabilità
 - frequenza di aggiornamento
 - struttura del contenuto
- Definizione di politiche di accesso

Output

- Catalogo delle fonti dati
- Mappatura fonte → dominio semantico



RAG: 3 Pipeline di Indicizzazione

La pipeline trasforma la tua sorgente di dati (es. documenti PDF, pagine web, database) in un **indice vettoriale** pronto per l'uso.

1. Caricamento dei Dati (Data Loading)

si recuperano i documenti dalla loro sorgente tramite una connessione alle fonti di dati (database, API, storage cloud, file system) e caricamento dei documenti (testo, PDF, HTML, Markdown, ecc.) nel sistema.

2. Suddivisione (Chunking)

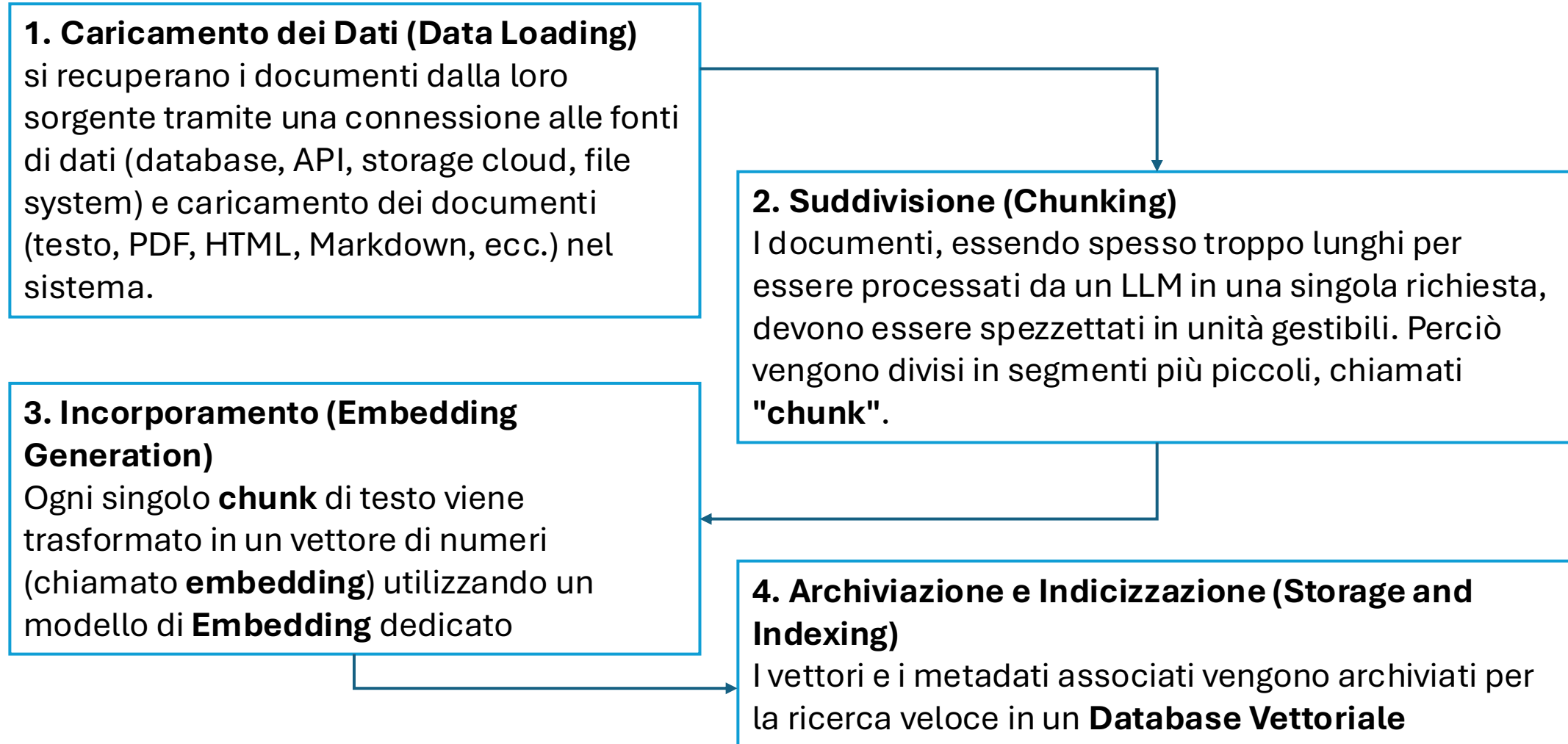
I documenti, essendo spesso troppo lunghi per essere processati da un LLM in una singola richiesta, devono essere spezzettati in unità gestibili. Perciò vengono divisi in segmenti più piccoli, chiamati **"chunk"**.

3. Incorporamento (Embedding Generation)

Ogni singolo **chunk** di testo viene trasformato in un vettore di numeri (chiamato **embedding**) utilizzando un modello di **Embedding** dedicato

4. Archiviazione e Indicizzazione (Storage and Indexing)

I vettori e i metadati associati vengono archiviati per la ricerca veloce in un **Database Vettoriale**



RAG: 4 Chunking

I documenti vengono suddivisi in unità semantiche (“chunk”) per ottimizzare il recupero.

Decisioni progettuali

- Dimensione chunk (token-based vs semantica)
- Overlap tra chunk
- Chunking strutturale (sezioni, titoli, paragrafi)

Trade-off

- Chunk piccoli → recall migliore, contesto frammentato
- Chunk grandi → contesto ricco, rischio rumore

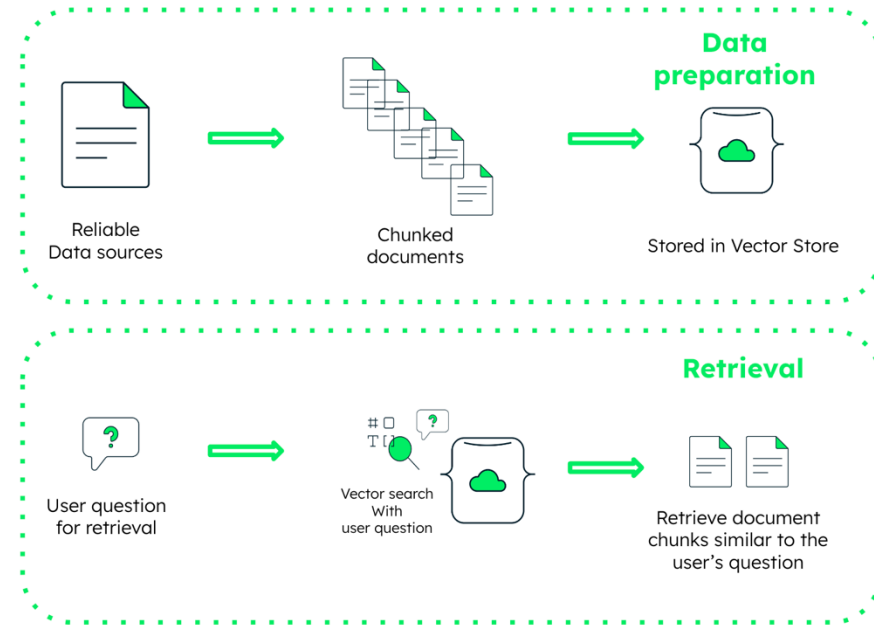
Output

- Set di chunk pronti per embedding

Il chunking è il processo di suddivisione di documenti di grandi dimensioni (PDF, Word, pagine web, ecc.) in porzioni più piccole e semanticamente coerenti, chiamate chunk.

Ogni chunk:

- contiene una quantità limitata di testo
- rappresenta un’unità di significato autonoma
- viene poi convertito in embedding e indicizzato in un vector database



Inviare documenti interi:

- supera i limiti
- aumenta i costi
- peggiora la qualità della risposta

I modelli LLM hanno un numero massimo di token.

RAG: 5 Embedding

Ogni chunk viene trasformato in un vettore numerico che rappresenta il significato semantico.

Attività chiave

- Scelta del modello di embedding
- Strategia di versioning degli indici

Output

- Indice vettoriale interrogabile
- Mapping vettore ↔ documento ↔ metadati

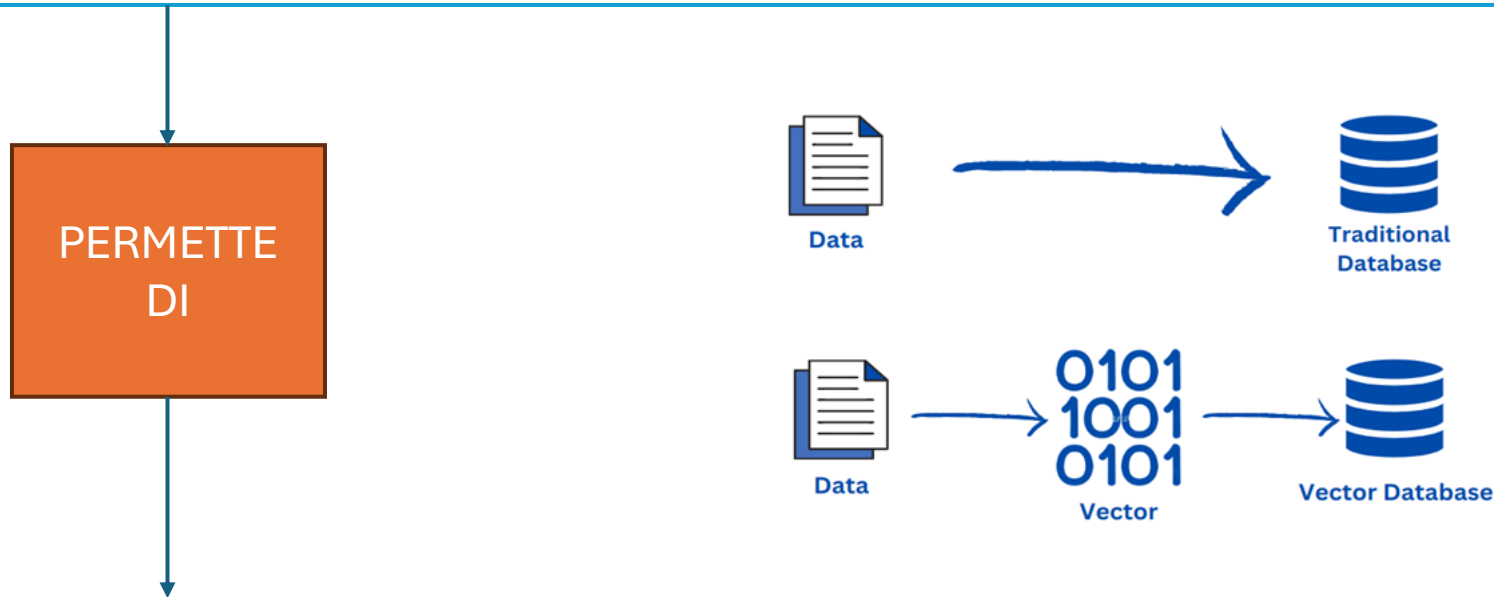
Un modello di embedding è un modello di machine learning che trasforma dati complessi (testo, immagini, audio, ecc.) in vettori numerici (liste di numeri) tali che elementi semanticamente simili risultino vicini tra loro nello spazio vettoriale.

Input → analisi semantica → rappresentazione vettoriale numerica

Model	Max Context Length	Embedding Output	Retrieval Capabilities	Key Strengths
BGE-M3	8,192 tokens	1,024 dims	Dense, sparse, and multi-vector retrieval	Unified hybrid retrieval in a single model
Qwen3-Embedding-8B	32,000 tokens	32 to 4,096 dims (configurable)	Dense embeddings with instruction-aware retrieval	Top-tier retrieval accuracy on long and complex queries
Arctic-Embed-L-v2.0	8,192 tokens	1,024 dims (MRL compressible)	Dense retrieval	High-quality retrieval with strong compression support
jina-embeddings-v3	8,192 tokens	32 to 1,024 dims (Matryoshka)	Task-specific dense retrieval via LoRA adapters	Flexible multi-task embeddings with minimal overhead
gte-multilingual-base	8,192 tokens	128 to 768 dims (elastic)	Dense and sparse retrieval	Fast, efficient retrieval with low hardware requirements

RAG: 6 Database Vettoriale

Un **Database Vettoriale** (*Vector Database*) è un tipo specializzato di database progettato per archiviare, indicizzare e interrogare in modo efficiente i **vettori di embedding**. Può essere considerato come il **motore di recupero** del RAG.



- A. Fornire Contesto Esterno e Aggiornato
- B. Ricerca Semantica Rapida e Accurata
- C. Mitigare le Allucinazioni e Migliorare l'Affidabilità


RAG: 6 Metodologie di Recupero (Retrieval)

Il retriever decide *quali* informazioni fornire al modello

Tipologia di ricerca	Descrizione	Quando usarla	Impatto sul RAG
Vector / Semantic Search	Recupera documenti semanticamente simili alla query tramite embedding.	Linguaggio naturale, sinonimi, parafrasi.	Fondamentale: determina quali documenti entrano nel contesto del LLM.
Metadata Filtering	Limita la ricerca usando filtri strutturati (data, tipo, dominio, permessi).	Compliance, sicurezza, perimetro controllato.	Riduce rumore e rischi: il LLM vede solo fonti ammesse e rilevanti.
Hybrid Search	Combina vector search e keyword search	Serve semantica + termini esatti.	Migliora precisione: meno allucinazioni da documenti “quasi pertinenti”.
Re-ranking	Riordina i risultati con modelli più accurati dopo il primo retrieval.	Ottimizzare il top-K recuperato.	Aumenta qualità del contesto: il LLM riceve le evidenze migliori per prime.
Diversified / MMR Search	Seleziona risultati rilevanti ma diversi tra loro, evitando duplicati.	Domande multi-aspetto, sintesi complesse.	Migliore copertura informativa: risposte più complete e bilanciate.

RAG: 7 Orchestratore

Nel contesto del processo **RAG**, l'**orchestratore** è il componente che **coordina e gestisce il flusso delle operazioni** tra le diverse parti del sistema.



non genera contenuti, ma **coordina retrieval e generation**, assicurando che il modello lavori con le informazioni più pertinenti

Gestione del flusso: decide come combinare il recupero delle informazioni (retrieval) con la generazione del testo (generation).

Interfaccia tra moduli:

- Riceve il prompt utente.
- Invia la query al motore di ricerca o al retriever per ottenere i documenti rilevanti.
- Passa i documenti recuperati al modello generativo.

Applicazione di logiche aggiuntive:

- Filtraggio dei risultati.
- Ranking dei documenti.
- Eventuale normalizzazione o arricchimento del prompt (prompt augmentation).

Controllo qualità: può implementare regole per garantire che la risposta sia coerente, completa e conforme alle policy.

RAG: 8 Prompt Contestuale

Un **Augmented Prompt** è un prompt “arricchito” o “potenziato” rispetto alla versione originale. In pratica, non è solo la richiesta di base, ma include **informazioni aggiuntive** per migliorare la qualità e la precisione della risposta generata da un modello AI.

Perché è utile?

- Migliora la **chiarezza** e riduce ambiguità.
- Aumenta la **qualità del risultato** (più vicino alle aspettative).
- Permette di **standardizzare** processi complessi (es. verbali, report, checklist).

Il prompt non è statico: viene costruito dinamicamente con i documenti recuperati.

Caratteristiche principali

- **Contestualizzazione**: aggiunge dettagli sul ruolo, obiettivo, vincoli, tono, formato.
- **Struttura**: suddivide il prompt in sezioni (es. contesto, obiettivo, output atteso).
- **Vincoli e regole**: specifica cosa includere o evitare.
- **Esempi**: può contenere esempi di output desiderato.



Incorporates relevant context and details



May specify desired structure, elements, or style



Can define constraints and rules



May offer examples of intended output

RAG: Tecniche di prompting

Prompting Techniques

Zero-Shot

No examples provided

“What are the benefits of exercise?”

Few-Shot

Include examples in the prompt

Article: Artificial Intelligence is transforming industries.

Summary: AI is revolutionizing various sectors. Article: Machine learning is...

Chain-of-Thought

Reasoning step-by-step

“First, identify the main factors contributing to climate change, then explain each one.”

Role/Persona

Specify a role or persona for the model

“As a career counselor, what advice would you give for a job interview?”

- **Zero-shot prompting:** istruzioni senza esempi.
- **Few-shot prompting:** aggiunta di esempi per guidare lo stile.
- **Chain-of-Thought:** incoraggia il ragionamento passo-passo.
- **Ruoli e contesto:** “Agisci come un esperto di AML

Esempio Zero-shot:

Prompt: 'Traduci in francese: Hello, how are you?'

Risposta: 'Bonjour, comment ça va?'

Esempio Chain-of-Thought:

Prompt: 'Spiega passo passo come risolvere 23+47.'

RAG: Struttura Base Prompt

[ROLE]

Sei ...

[TASK]

Il tuo compito è ...

[CONTEXT]

Il pubblico è ...

Il contesto è ...

[INSTRUCTIONS]

- Fai X
- Non fare Y
- Usa Z

[OUTPUT FORMAT]

Restituisci la risposta in questo formato ...

[INPUT]

<<contenuto dell'utente>>

Regola d'oro

Un buon prompt è una specifica di lavoro, non una domanda.

Più è vicino a un brief interno, migliore sarà l'output.

Errori comuni da evitare

- ✗ Prompt vaghi ("spiegami X")
- ✗ Nessun contesto sul pubblico
- ✗ Output non vincolato (testo caotico)
- ✗ Troppe richieste in un solo prompt
- ✗ Dare per scontate conoscenze che il modello non ha

RAG: Buone pratiche di prompting

Area	Buona Pratica	Descrizione
Definizione	Sii Specifico e Chiaro	Evita ambiguità. Indica esattamente cosa vuoi.
	Assegna un Ruolo	Dì all'LLM chi deve essere ("Agisci come un esperto di...") per impostare tono e competenza.
Contesto	Fornisci Contesto	Includi tutte le informazioni necessarie (documenti, dati) prima della domanda.
	Few-Shot Prompting	Usa esempi di input/output nel prompt per mostrare il formato atteso.
Vincoli	Definisci il Formato	Richiedi un output strutturato specifico (es. JSON , elenco puntato, tabella).
	Imposta la Lunghezza/Stile	Specifica limiti di parole, tono (formale, amichevole) o pubblico.
	Istruzioni Negative	Dì al modello cosa non deve fare (es. "Non usare gergo tecnico").
Qualità	Chain-of-Thought (CoT)	Chiedi all'LLM di mostrare i passaggi intermedi ("Pensa ad alta voce...") prima della risposta finale.
	Gestisci le Allucinazioni	Istruisci il modello a rispondere "Informazione non disponibile" se non è sicuro o non ha contesto.
Miglioramento	Itera e Affina	Se l'output è sbagliato, chiedi all'LLM di correggerlo o di riscriverlo, fornendo la critica.

RAG: 9 -12 Modello, Metriche e Deploy

9. Scelta e configurazione del modello LLM

Il modello genera la risposta finale usando il contesto fornito.

Decisioni

- Modello generalista vs specializzato
- Latenza vs qualità
- Eventuale fine-tuning + RAG (approccio ibrido)

Output

- LLM integrato nella pipeline

10. Valutazione, testing e osservabilità

Una RAG senza valutazione è una “black box”.

Attività

- Test su domande reali
- Valutazione grounding (risposta ↔ fonte)
- Logging di query, documenti recuperati e output
- Human-in-the-loop per casi critici

Output

- Metriche di qualità
- Evidenze di affidabilità

12. Deployment e integrazione applicativa

La RAG viene esposta tramite API o UI.

Esempi

- Chatbot interni
- Assistenti tecnici
- Sistemi decisionali

Framework applicativi citati nei materiali interni includono **Chainlit** e **Streamlit** per front-end RAG.

RAG: Metriche di Valutazione

una buona valutazione di una architettura RAG deve considerare diversi criteri. Non basta “la risposta sembra buona”: serve un approccio strutturato con dataset di validazione e metriche quantitative.

Metrica	Descrizione	Come si calcola
Context Relevance	Misura quanto i documenti recuperati sono pertinenti alla query.	Precision@k: # documenti rilevanti tra i primi k / k. Usa embedding + similarità coseno per valutare rilevanza.
Context Recall	Copertura dei documenti rilevanti rispetto alla base di conoscenza.	# documenti rilevanti recuperati / # documenti rilevanti totali.
Context Usefulness (CU)	Quanto il contesto recuperato contribuisce alla risposta finale.	Valutazione manuale o automatizzata: punteggio da 1 a 5 basato su utilità del contesto.
Answer Relevance	Pertinenza della risposta rispetto alla domanda.	BLEU / ROUGE / BERT Score confrontando risposta generata con risposta di riferimento.
Faithfulness	Fedeltà della risposta ai documenti recuperati (evita allucinazioni).	Verifica automatica: ogni affermazione deve essere supportata da almeno un documento (fact-checking).
Answer Correctness	Accuratezza rispetto alla domanda di riferimento.	Confronto con golden set: punteggio binario (corretto/errato) o F1-score.
Latency	Tempo di risposta del sistema.	Misurazione in millisecondi dal momento della query alla risposta finale.
Resource Usage	Efficienza in termini di CPU/GPU e memoria.	Monitoraggio runtime: consumo medio per query.

RAG: Principali tecniche

Oltre alla query singola possono essere integrate altre tecniche volte a migliorare le risposte ottenute

Tecnica	Idea chiave	Come funziona (in breve)	Problema che risolve	Punti di forza	Limiti / Trade-off	Quando usarla
Multi Query	Più riformulazioni della stessa domanda	LLM genera N versioni della query → retrieval per ciascuna → merge/ deduplica dei risultati	Mismatch lessicale, query ambigue o troppo brevi	↑ Recall, copertura semantica più ampia, semplice da integrare	↑ costi e latenza; possibile rumore se le query sono poco centrate	Query vaghe, linguaggio naturale, utenti non esperti
RAG-Fusion	Multi-Query + fusione dei ranking	Come Multi Query, ma i risultati sono combinati con Reciprocal Rank Fusion (RRF)	Stabilità del ranking e robustezza ai “falsi positivi”	Ranking più robusto, riduce dipendenza da una singola query	Più complesso; costo maggiore; qualità dipende dalle query generate	Sistemi enterprise “precision-critical”, QA complesso
Decomposizione	Scomposizione in sotto-domande	LLM divide la query in sub-question → retrieval per ciascuna → sintesi finale	Domande multi-hop o multi-vincolo	Migliora multi-hop reasoning; isolamento delle evidenze	Pipeline più lunga; errori di decomposizione si propagano	Confronti, analisi, domande “a più passi”
Step Back	Astrazione prima del dettaglio	Prima si chiede al modello una domanda/concetto più generale → si usa per reasoning o retrieval	Focalizzazione eccessiva sui dettagli, ragionamento fragile	Migliora coerenza logica; recupera principi/contesto	Può diventare troppo generico; molto prompt-sensitive	Normative, concetti teorici, reasoning strutturale
HyDE	Query → “documento ipotetico”	LLM genera un documento fittizio → embedding → retrieval di documenti reali simili	Query troppo corte vs documenti lunghi (embedding mismatch)	↑ Recall zero-shot; molto efficace su domini nuovi	Rischio “deriva semantica”; costo generazione+embedding	Query esplorative, domini verticali, zero-shot

RAG: Panoramica dei Vantaggi



Riduzione del rischio di allucinazioni

limita errori integrando risposte da fonti reali e verificate, aumentando l'affidabilità.

Accesso a dati aggiornati

Permette di usare dati recenti senza riaddestrare il modello, riducendo tempi e costi.

Personalizzazione e contestualizzazione

Integra dati aziendali specifici per risposte su misura e rilevanti per il contesto.

Scalabilità efficiente e affidabilità

Aggiornando solo la knowledge base si mantiene l'efficienza.

RAG: Panoramica dei Limiti

Complessità di implementazione

La creazione di pipeline di indicizzazione e la gestione di database vettoriali richiede competenze tecniche avanzate.

Costi infrastrutturali elevati

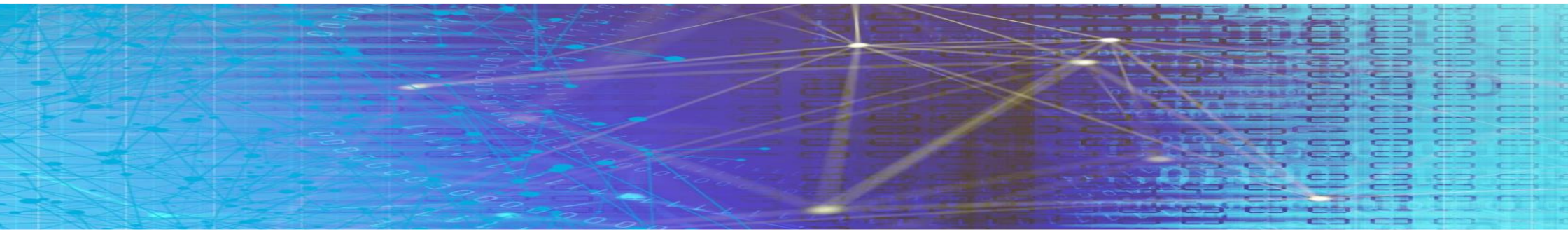
I costi per gestire grandi volumi di dati possono essere significativi per molte organizzazioni.

Qualità della knowledge base

Dati complessi, incompleti o errati compromettono l'efficacia delle risposte generate dal sistema.

Sicurezza e privacy dei dati

Proteggere le informazioni sensibili è fondamentale durante il processo di retrieval.



RAG: Gestione dei Volumi delle Informazioni

Il problema principale non è solo lo spazio di archiviazione, ma la **velocità di recupero** e il mantenimento della **pertinenza** all'aumentare dei documenti (milioni di chunk)

Problematiche:

Latenza di ricerca: All'aumentare dei vettori, la ricerca esaustiva diventa impraticabile.

Costi di Embedding: Ri-indicizzare grandi volumi di dati a ogni aggiornamento del modello di embedding è estremamente costoso.

Rumore: Più dati ci sono, più è probabile recuperare frammenti "simili" ma semanticamente irrilevanti.

Crescita esponenziale dei documenti: repository eterogenei con formati diversi

Soluzioni:

Indicizzazione Avanzata (ANN): Utilizzo di algoritmi specifici per ricerche vettoriali approssimate ma istantanee.

Metadata Filtering: Non affidarsi solo ai vettori. Pre-filtrare i dati usando metadati per restringere drasticamente lo spazio di ricerca prima della comparazione vettoriale.

Data Lifecycle Management: Implementare politiche di "TTL" (Time To Live) o archiviazione per i chunk obsoleti, mantenendo nel database vettoriale solo le informazioni correnti.

RAG: Interpretazione delle Informazioni

Recuperare il documento corretto non serve a nulla se il modello non ne comprende il contenuto o se il contesto è frammentato male.

Problematiche:

Chunking inadeguato: Dividere un testo a metà frase interrompe il senso logico (perdita di coerenza).

Disallineamento Semantico e ambiguo: Le query degli utenti sono spesso brevi e ambigue, mentre i documenti sono tecnici e densi.

Multimodalità: Presenza di tabelle o grafici che, se convertiti in testo semplice, perdono significato.

Contesto conflittuale: documenti con versioni discordanti; normative aggiornate vs storiche

Soluzioni:

Semantic Chunking: Invece di dividere ogni 500 caratteri, usare modelli NLP per identificare i cambi di argomento e dividere i testi in unità di senso compiuto.

Hybrid Search: Combinare la ricerca vettoriale (significato) con la ricerca a parole chiave per non perdere termini tecnici o codici specifici (es. codici errore, Service Tag).

Reranking (Cross-Encoders): Dopo il recupero iniziale dei primi 50 risultati "simili", utilizzare un modello di **Reranker** (più lento ma più preciso) per riordinarli e passare al LLM solo i primi 5 realmente pertinenti.

RAG: Gestione del Contesto nel Prompt

Anche con un LLM con finestra di contesto ampia (es. 128k o 1M di token), inserire troppe informazioni degrada le performance.

Problematiche:

Lost in the Middle: Gli LLM tendono a ignorare le informazioni poste al centro di un prompt molto lungo, dando peso solo all'inizio e alla fine.

Superamento dei Limiti: Inserire troppi chunk consuma token inutilmente, aumentando i costi e la latenza di generazione.

Contesto Irrilevante: L'inserimento di frammenti non pertinenti può portare il modello fuori strada (allucinazioni indotte).

Mancanza di role separation: istruzioni sistema, politiche e dati si confondono

Soluzioni:

Context Compression: Utilizzare algoritmi per rimuovere i token meno informativi dai documenti recuperati prima di inserirli nel prompt.

Prompt Engineering Strutturato: Formattare il contesto in modo chiaro (es. usando tag XML come `<context></context>`) per aiutare il modello a distinguere tra le sue conoscenze e i dati forniti.

Recursive Retrieval / Parent Document Retrieval: Invece di passare chunk piccoli e frammentati, passare un riassunto del documento o il paragrafo intero se il chunk piccolo risulta pertinente, garantendo una narrazione coerente.

Citation-first generation: LLM deve generare prima la lista di fonti con ID e passaggi, poi comporre la risposta; se una parte non ha fonte, indicarlo.

RAG: Framework

Un framework RAG raggruppa diverse funzionalità come ingestione, chunking, incorporamento, recupero e generazione in componenti riutilizzabili, evitando di dover ogni volta rielaborare manualmente il codice attorno a indici e prompt.

Top 10 Open-Source RAG Frameworks: Power Your AI with Grounded Answers



Turn raw data into grounded AI answers.
Plug into any LLM & vector DB. Open-source.
Production-ready. Developer-friendly.

I moderni framework RAG open source offrono:

- Retriever plug-in e modelli di incorporamento.
- Orchestrazione di catene multi-step, valutatori, strumenti e agenti attorno a RAG.
- Integrazioni con linguaggi e tecnologie già esistenti.

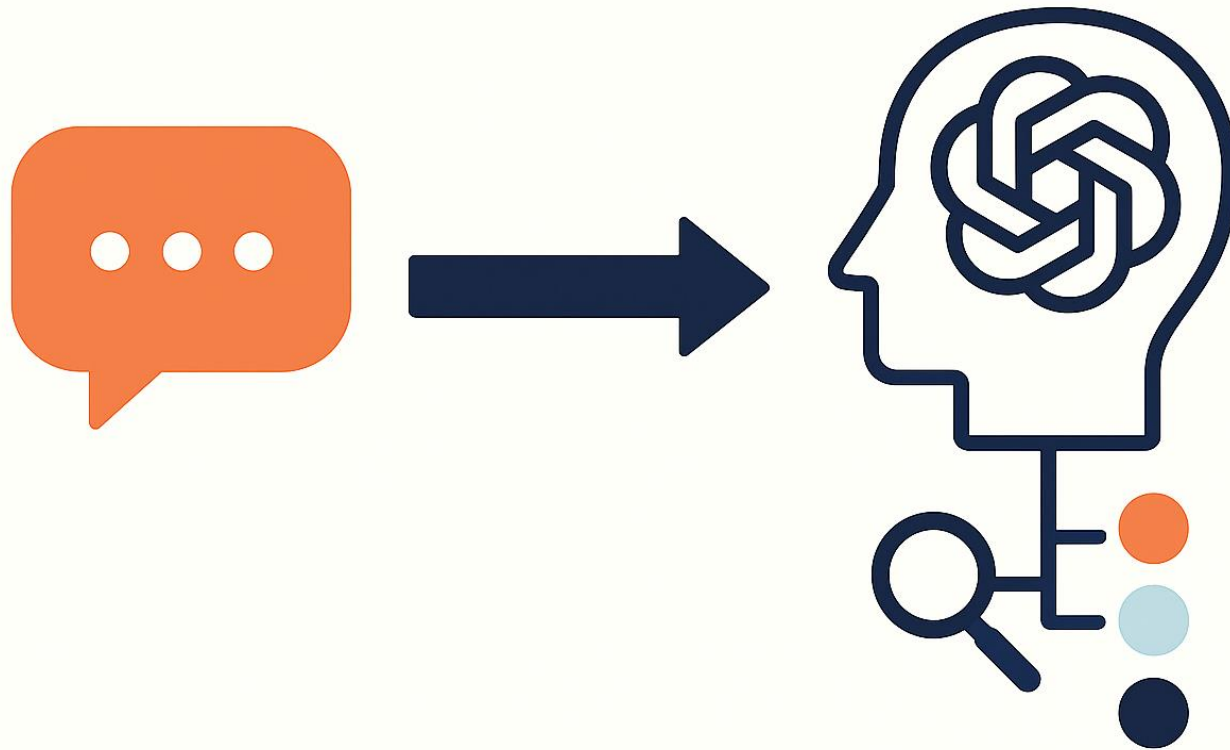
[MAGGIORI
DETTAGLI](#)

Fine-Tuning e RAG Differenze chiave

ASPETTO	FINE-TUNING	RAG
Obiettivo	Modifica permanente del modello	Arricchisce risposte con dati esterni
Fonte di conoscenza	Nei pesi del modello	Database/documenti esterni
Aggiornabilità	Solo con nuovo addestramento	Immediata, basta aggiornare le fonti
Costi	Alti (training)	Bassi (setup indice)
Velocità	Alta (risposta diretta)	Più lenta (recupero + generazione)
Personalizzazione	Molto precisa	Meno profonda
Manutenzione	Complessa	Semplice

4. Architettura RAG

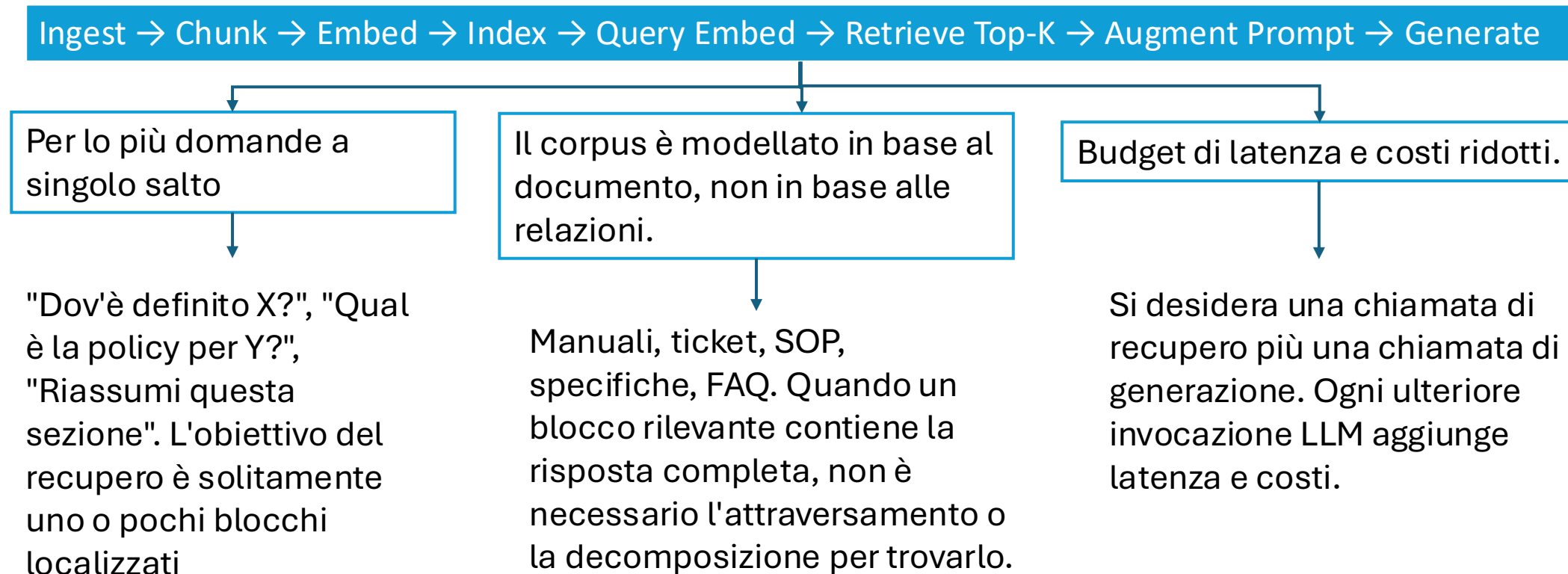
FROM PROMPT TO AGENT



Architettura Lineare

Combina la memoria parametrica (pesi del modello) con la memoria non parametrica (un indice esterno) per migliorare le prestazioni nelle attività ad alta intensità di conoscenza.

Una chiamata di recupero. Una chiamata di generazione. Minimo sovraccarico di orchestrazione. Questa semplicità è la caratteristica, non il limite.



Architettura Lineare: Tipologia di Chain

Prompt Diretto

Fornisce risposte semplici e immediate senza spiegazioni intermedie, ideale per compiti semplici e rapidi.

Chain-of-Thought

Descrive il ragionamento passo dopo passo, utile per problemi complessi e analisi approfondite..

Prompt Diretto per compiti semplici

ideale per domande semplici e risposte rapide senza necessità di trasparenza nel processo.

Chain-of-Thought per problemi complessi

supporta ragionamenti multi-step, analisi dettagliate e spiegazioni approfondite per problemi complessi.

Differenze chiave

Il Prompt Diretto è veloce ma meno trasparente, il Chain-of-Thought è più chiaro e verificabile.

Applicazioni consigliate

Scegliere la tecnica in base al contesto: velocità per Prompt Diretto, accuratezza per Chain-of-Thought

Architettura Agentica

Agentic RAG sostituisce "recupera una volta" con "recupera, valuta, rivedi, recupera di nuovo". Aggiunge un livello di controllo attorno al recupero e alla generazione. Il sistema può decidere quando recuperare, come recuperare e se iterare.

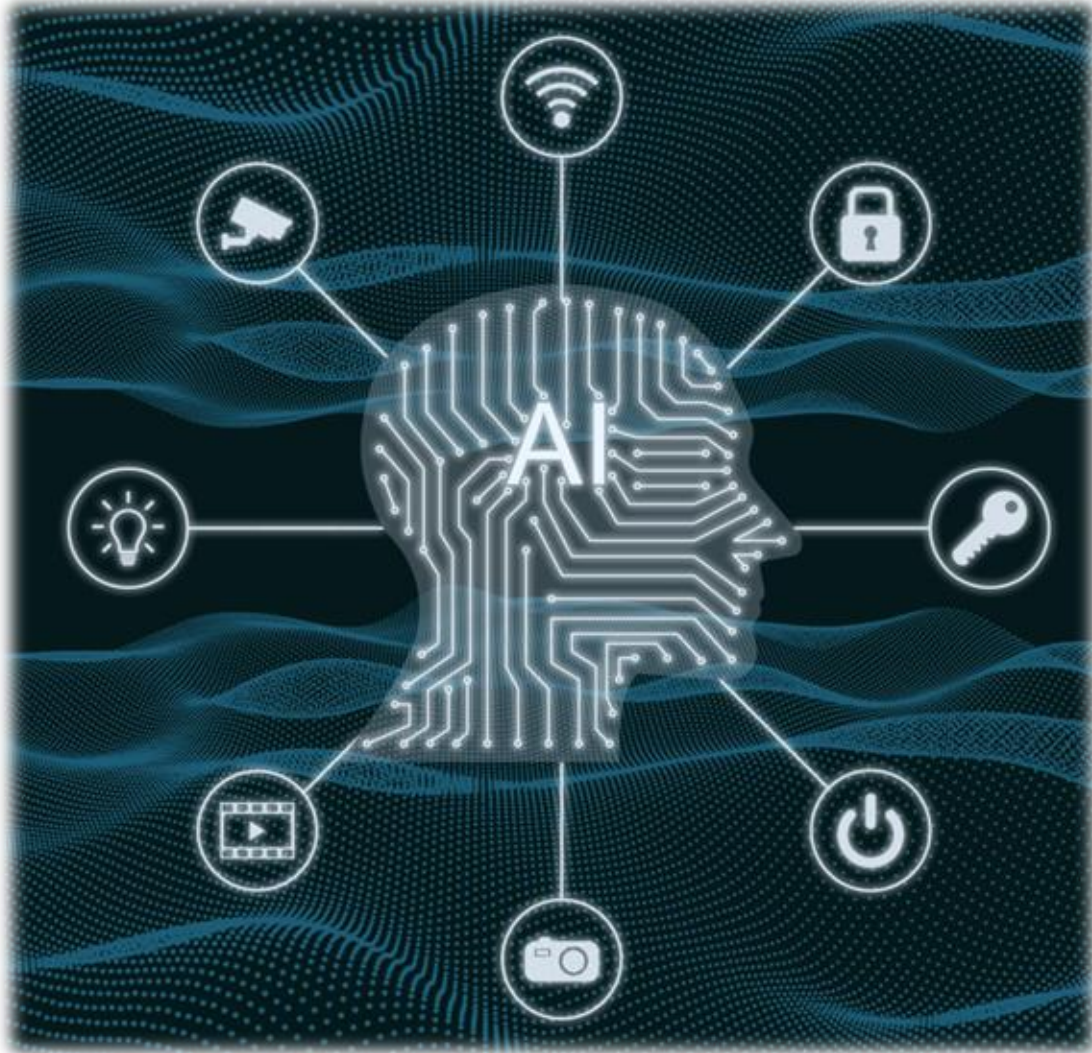
Il paradigma ReAct (ragionamento più azione) dimostra una riduzione delle allucinazioni e della propagazione degli errori. Metodi autoriflessivi come Self-RAG formalizzano il ciclo "recupera, genera, critica" come un loop strutturato.

Domande multiparte o multi-hop. La query contiene più domande, richiede la scomposizione in sottoquery o l'integrazione di prove tra le fasi di recupero.

Query ambigue in cui una singola ricerca semantica recupera un risultato sbagliato. Invece di sperare che la prima ricerca vada a buon fine, il sistema valuta ciò che ha ottenuto e riprova con una query migliore.

Agentic RAG offre agli agenti di recupero l'accesso a più strumenti oltre a un singolo indice vettoriale.

Architettura Agentica: Caratterizzazione



Definizione di agente LLM

Un agente LLM utilizza un modello linguistico avanzato per comprendere e agire autonomamente su compiti complessi.

Funzionalità avanzate

Capace di pianificare azioni, prendere decisioni e interagire con API esterne in modo intelligente.

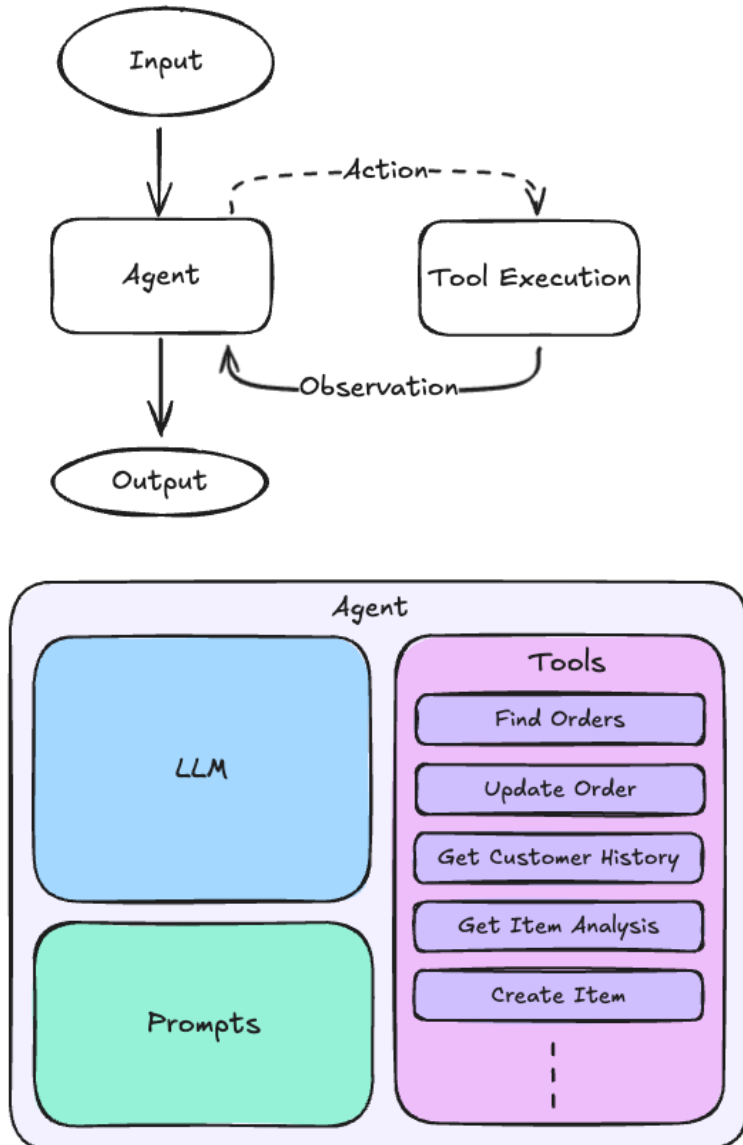
Apprendimento e adattamento

Memorizza interazioni e migliora continuamente grazie al feedback e all'apprendimento contestuale.

Autonomia e flessibilità

Agisce proattivamente in ambienti digitali o fisici per risolvere problemi complessi.

Architettura Agentica: Componenti



Modello linguistico principale: decide in modo intelligente quali strumenti eseguire (e in quale ordine, se necessario) per raccogliere in modo efficace il contesto appropriato necessario per rispondere alle domande di input.

Modulo di pianificazione: scompone obiettivi complessi in attività semplici, definendo sequenze di azioni efficaci.

Memoria e apprendimento: memorizza informazioni passate per apprendere e migliorare risposte e azioni nel tempo (non obbligatoria).

Tools: consentono all'agente di agire. Possono essere forniti dai server MCP o definiti localmente all'interno del codice. E' responsabilità dei prompt e dell'LLM utilizzarli in modo efficace.

Prompt: forniscono le indicazioni su come l'agente dovrebbe comportarsi e su eventuali istruzioni specifiche da seguire.

Architettura Agentica: Reasoning

Modalità di decisione	Come funziona	Cosa usa per decidere	Pro	Contro	Quando usarla
Tool calling model-driven (solo descrizioni + schema)	Il modello, vedendo catalogo tool (nome/descrizione/schema), emette una <i>tool call</i> quando serve.	Descrizione tool + schema parametri	Semplice, veloce da avviare	Sensibile a descrizioni “poco chiare”; rischio chiamate inutili	POC, tool pochi e ben distinti
ReAct / loop “Reason→Act→Observe”	Il modello alterna ragionamento, selezione tool, osservazione risultato e iterazione fino a risposta finale.	Obiettivo + risultati intermedi + tool disponibili	Adattivo: può correggersi e cambiare tool	Più costo/latency; necessita limiti (max iter)	Task multi-step, investigazioni, troubleshooting
Router / gating (regole + condizionali)	Un router decide prima: “serve tool o risposta diretta?” e instrada il flusso (conditional edges / tools_condition).	Regole, classificatore intent, presenza di tool_calls	Controllo e governance; riduce costi	Richiede progettazione del router	Produzione enterprise, scenari compliance/costi
Routing basato su tool_calls (LangGraph tools_condition)	Il sistema ispeziona l’ultimo messaggio: se contiene tool_calls → esegue ToolNode, altrimenti termina.	Presenza/assenza di tool_calls nel messaggio	Pattern standard, robusto, semplice da mantenere	Dipende dalla capacità del modello di emettere tool_calls corretti	Workflow agentici con LangGraph, RAG come tool
MCP Tool Discovery (tools/list → tools/call)	Il client scopre i tool dal server via tools/list e invoca via tools/call usando schema standard; i tool sono “model-controlled”.	Name/description/inputSchema forniti dal server MCP	Standardizzazione, riuso, ecosistema tool; governance migliore	Se il catalogo è enorme serve curare descrizioni/organizzazione	Integrazioni enterprise e tool condivisi tra applicazioni
Policy-based tool approval (Human-in-the-loop)	Il modello propone tool call, ma l’esecuzione richiede approvazione (sempre o condizionata).	Policy + tipo azione (read vs write)	Riduce rischi (write/PII); auditabilità	Più attrito; UX meno fluida	Azioni con side-effect (ticket, email, update DB)
Selezione via metadati (tool tags / scope)	Il router filtra i tool esposti al modello in base a contesto/ruolo/permessi (principio del least privilege).	Tag, permessi, contesto sessione	Riduce errori e rischio di tool sbagliato	Richiede gestione metadati e RBAC/ABAC	Ambienti regolati, strumenti numerosi
Self-reflection / grading per scelta tool	L’agente valuta se il tool usato ha prodotto evidenza sufficiente; se no, riformula o cambia tool (pattern agentic RAG).	Valutazione qualità risultati (relevance/groundedness)	Migliora affidabilità; riduce “false confidence”	Più chiamate; complessità	Domande ambigue o retrieval rumoroso

Architettura Agentica: descrizione dei tool

Per migliorare drasticamente la selezione del tool, investi su:

- descrizioni operative “quando usarlo / quando non usarlo”
- schema input rigoroso
- esempi positivi/negativi
- policy esplicite (read vs write, HITL, logging)

STRUMENTI DISPONIBILI

Usa i tool solo quando necessario. Se puoi rispondere direttamente, NON usare tool.

kb_pdf_search (RAG)

Usa questo tool per cercare informazioni nei documenti indicizzati (policy, contratti, procedure).

✅ USA quando:

- la risposta deve essere basata su documenti
- servono clausole, definizioni, obblighi, valori ufficiali

❌ NON usare quando:

- servono calcoli
- l'utente chiede solo un'azione

Input:

- query (string, obbligatorio)
- top_k (int, opzionale)

Se non trovi evidenze: dichiaralo e non inventare.

calculator

Usa questo tool solo per calcoli numerici.

✅ USA quando:

- l'utente chiede percentuali, interessi, somme

❌ NON usare quando:

- prima devi recuperare dati da documenti

Input:

- expression (string, es. "10000 * 0.025")

Architettura basata su Grafi

Il Knowledge Graph RAG modifica il substrato di recupero da "blocchi di testo più simili" a "entità più relazioni che è possibile attraversare".

Mentre il pipeline RAG chiede "quale blocco di testo è più vicino a questa query?", il knowledge graph RAG chiede "quali entità sono connesse e in che modo?". Si tratta di un'operazione di recupero fondamentalmente diversa.

Domande che incidono molto sulle relazioni. "Qual è la relazione tra X e Y?", "Quali sistemi dipendono da questo componente?", "Chi ha approvato cosa?". E' progettato appositamente per rappresentare fatti interconnessi.

Ragionamento multi-hop su entità in cui un singolo blocco raramente contiene l'intera catena di informazioni.

Creazione di senso per l'intero corpus. "Quali sono le tendenze?", "Riassumi i temi in tutti i documenti". E' progettato per rispondere a domande a cui nessun singolo risultato di ricerca può rispondere.

Requisiti di correttezza temporale. "Chi ricopriva il ruolo X al momento T?", "Quale politica era in vigore in questa data?". Supportano query basate sul tempo e il recupero multi-hop attraverso fatti in continua evoluzione.

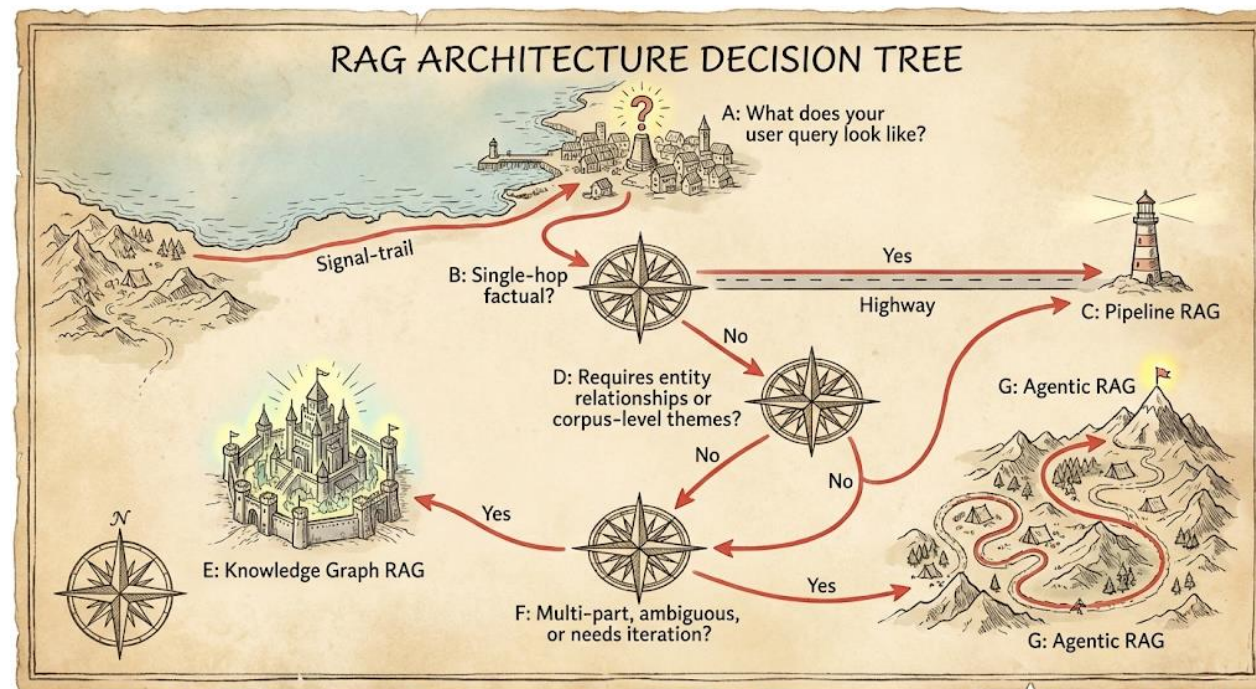
Quale architettura scegliere?

La scelta della corretta architettura non si basa soltanto su un elenco di funzionalità. È confrontare le richieste effettive degli utenti con l'architettura che gestisce al meglio tali domande.

RAG Lineare ottimizza la velocità. Un recupero, una generazione.

RAG Agentico ottimizza l'adattamento. Ripeti il ciclo finché le prove non sono sufficientemente valide.

Graph RAG ottimizza la struttura. Recupera in base a relazioni e vincoli, non solo tramite similarità.

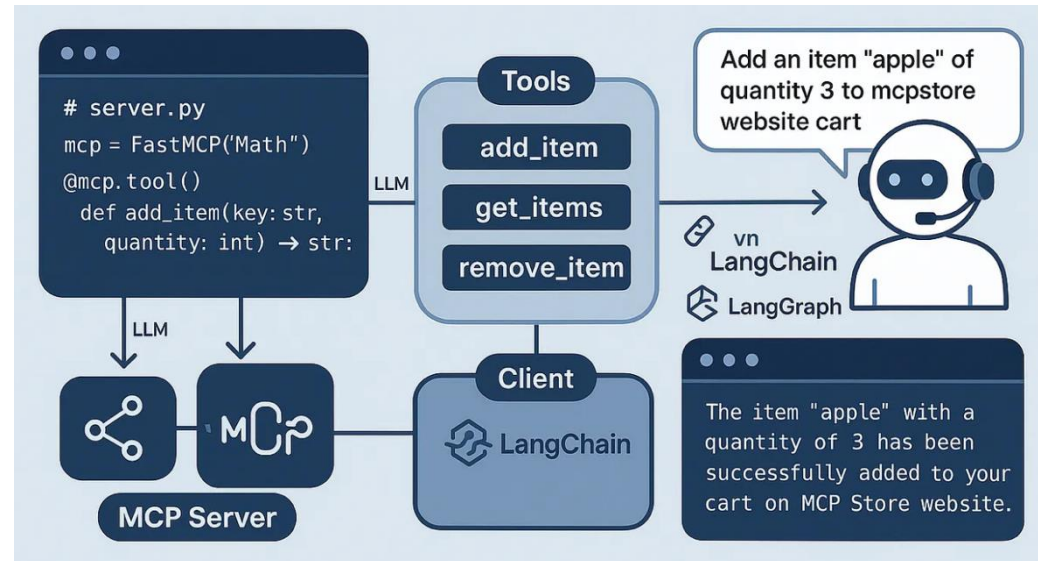


Model Context Protocol (MCP)

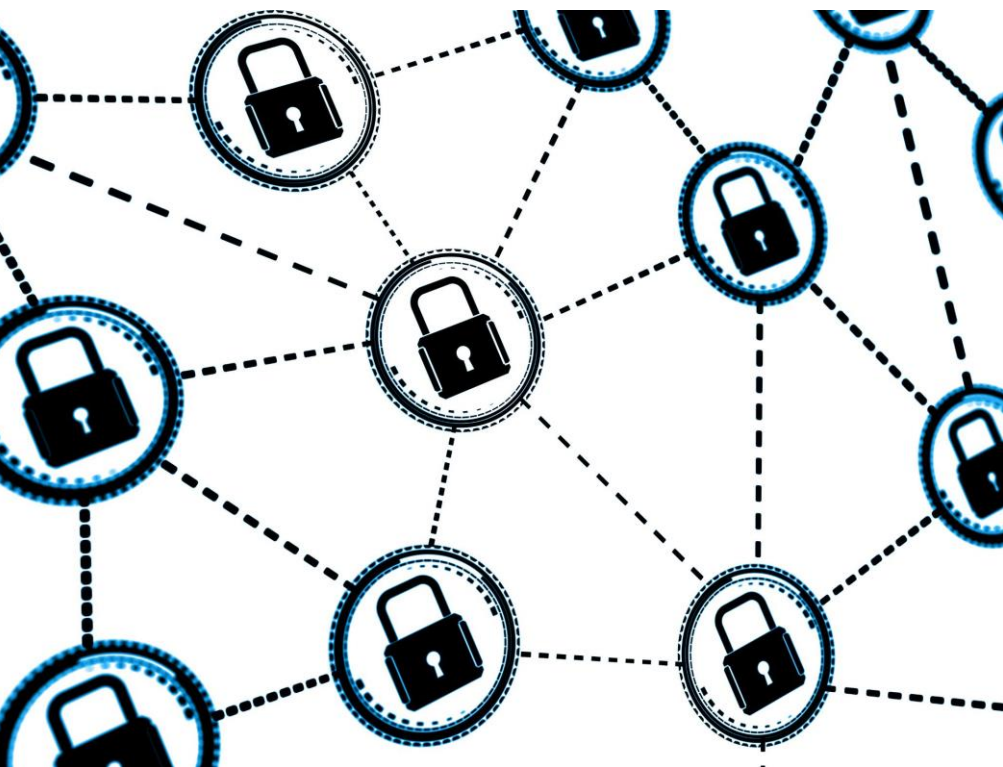
standard open source per la connessione di applicazioni di intelligenza artificiale a sistemi esterni

È possibile connettere applicazioni di intelligenza artificiale che utilizzano LLM a molteplici fonti di dati (ad esempio file locali, database), strumenti (ad esempio motori di ricerca) e flussi di lavoro (ad esempio prompt specializzati), consentendo loro di eseguire attività specifiche.

utilizza un'architettura client-server



MCP: Obiettivi



Protocollo universale MCP

è uno standard aperto che connette modelli AI a dati e strumenti esterni con un linguaggio comune.

Risoluzione problema M×N

elimina complessità e costi delle integrazioni personalizzate tra molteplici applicazioni e strumenti.

Sicurezza e interoperabilità

Il protocollo migliora la sicurezza, integra policy di audit e garantisce risposte affidabili e pertinenti.

Evoluzione verso AI agentic

abilita l'AI a orchestrare processi complessi autonomamente, superando i limiti della conoscenza statica.

MCP: Architettura

MCP Host

- utilizza uno o più LLM
- gestisce l'interazione con l'utente
- orchestra più MCP Client
- decide quando richiedere contesto esterno
- integra il contesto ricevuto nel prompt del modello

MCP Client

- è il connettore logico tra Host e Server
- mantiene una connessione dedicata
- invia richieste strutturate (JSON-RPC 2.0)
- riceve contesto, risultati o strumenti disponibili
- gestisce sessione, errori, timeout
- non contiene logica di business né dati sensibili

MCP Server

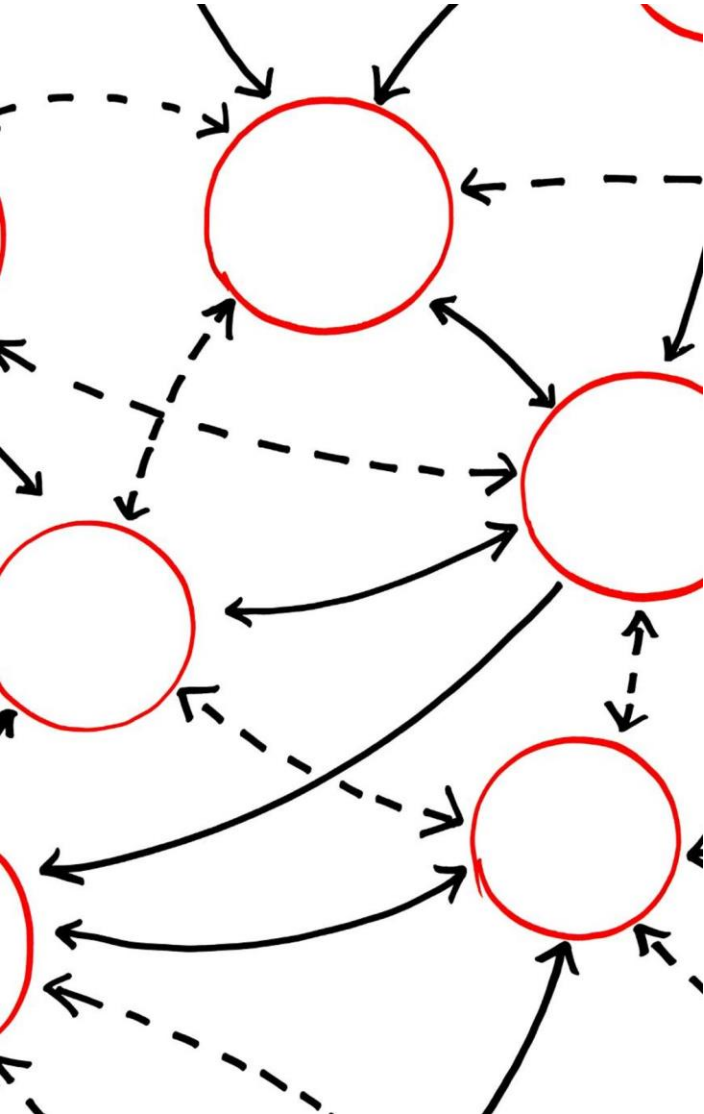
- è il componente che espone contesto esterno al modello.
- Può essere: locale (filesystem, DB) o remoto (API, cloud)
- mantiene ownership e controllo dei dati
- applica policy di sicurezza e autorizzazione
- non conosce il modello né il prompt finale



TIRIAMO LE FILA

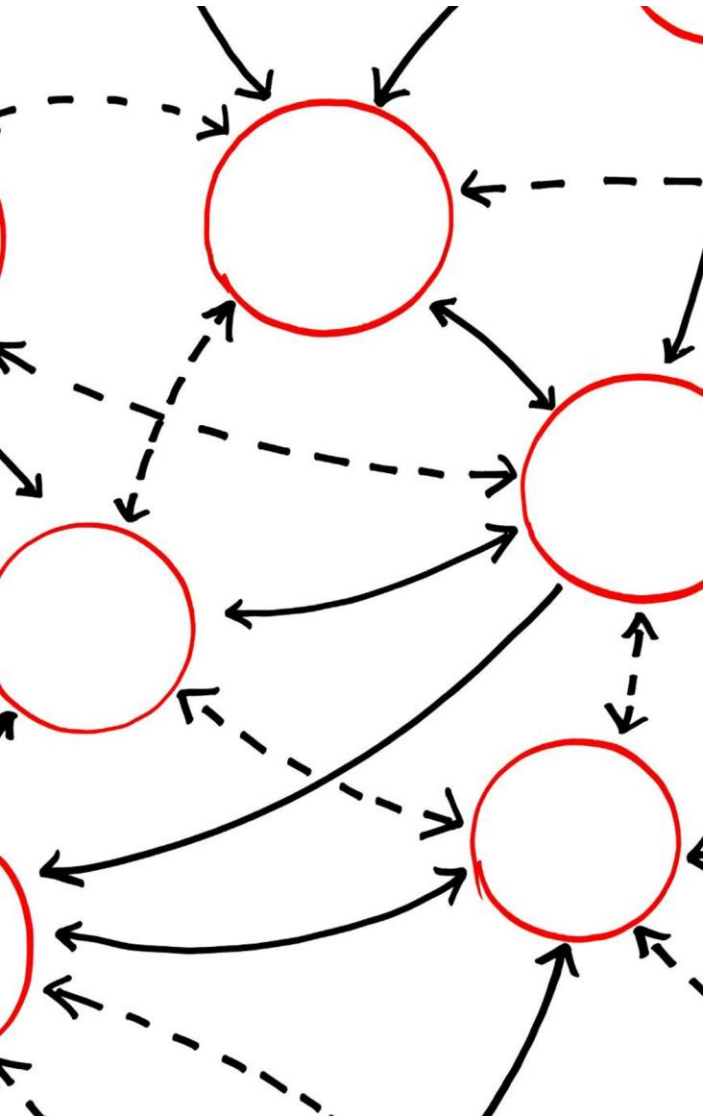


RAG Tradizionale: caratteristiche e limiti



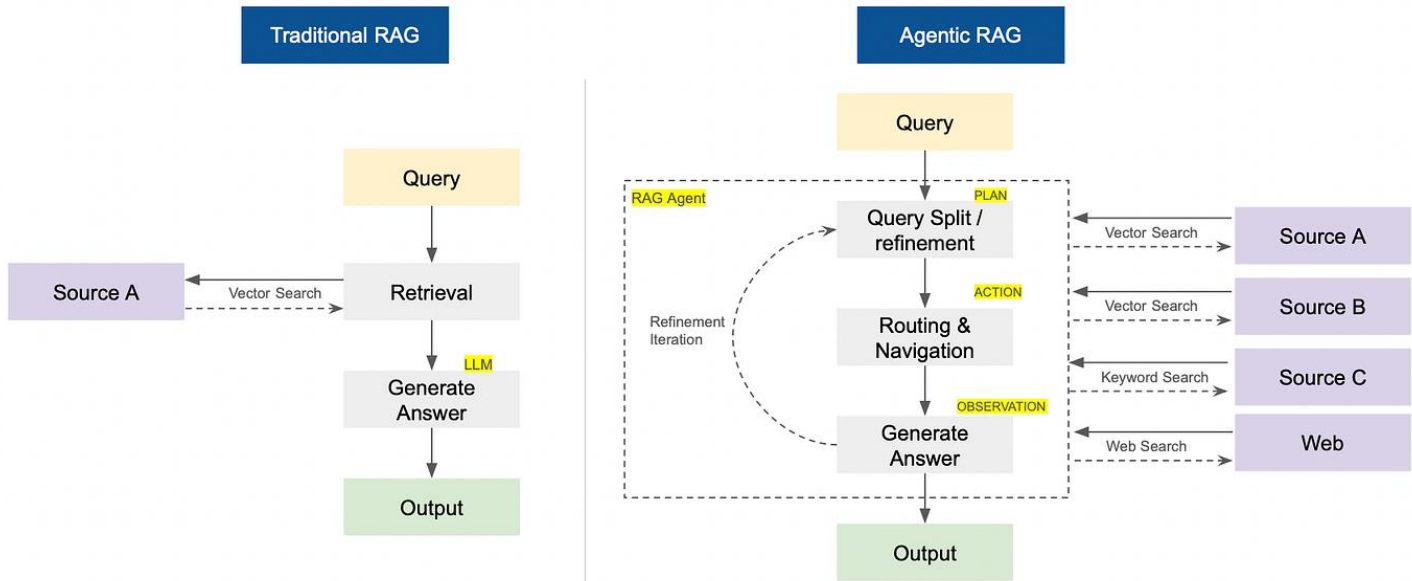
- **Funzionamento lineare e deterministico**
Il sistema trasforma la query in embedding, ricerca documenti rilevanti e genera risposte basate sul contesto recuperato.
- **Vantaggi principali**
Architettura semplice e rapida che riduce le allucinazioni grazie all'uso di fonti esterne verificabili.
- **Limitazioni chiave**
Non supporta adattamento dinamico, iterazioni, integrazione API per gestire query complesse.
- **Applicazioni ideali**
Adatto a scenari medio/semplici e domande dirette, non per contesti con decision-making o necessità di orchestrazione complessa.

Agentic RAG: evoluzione e vantaggi



- **Evoluzione del paradigma RAG**
Agentic RAG introduce agenti AI autonomi per un approccio iterativo e dinamico nel retrieval e generazione di informazioni.
- **Workflow complesso e decision-making**
Gli agenti AI gestiscono workflow complessi, interagiscono con API esterne e prendono decisioni autonome.
- **Memoria e feedback loop**
Utilizzo di memoria e feedback loop per migliorare precisione e adattabilità alle esigenze contestuali.
- **Applicazioni avanzate e vantaggi**
Agentic RAG è ideale per query articolate e processi multi-step, offrendo maggiore adattabilità e accuratezza ma risulta più costoso in termini economici e di risorse.

Tabella comparativa



ASPETTO	RAG TRADIZIONALE	AGENTIC RAG
Pipeline	Lineare	Iterativa
Autonomia	Nessuna	Alta
Tool Integration	No	Sì
Adattabilità	Limitata	Elevata
Complessità	Limitata	Elevata

5. Ambienti di test e sviluppo

TEST AND DEVELOPMENT ENVIRONMENTS



Development: Perché e come

Perché testare LLM in locale

- **Privacy e sicurezza:** i dati rimangono all'interno della tua macchina.
- **Controllo e personalizzazione:** puoi gestire aggiornamenti, tuning e integrazione.
- **Riduzione dei costi:** niente abbonamenti o costi per token, sfrutti il tuo hardware
- **Scaricare modelli open source ottimizzati (GGUF) per inferenza locale.**
- **Usare quantizzazione per ridurre consumo di memoria.**
- **Testare con dataset realistici e metriche (accuratezza, coerenza, bias).**
- **Integrare API locali per simulare scenari di produzione.**



1.Container e ambienti virtualizzati

1. **Docker:** molti modelli offrono immagini pronte per ambienti isolati.
2. **WSL (Windows Subsystem for Linux):** per eseguire tool Linux su Windows.

2.Utilizzo di strumenti dedicati

1. **Ollama**: framework open source per eseguire modelli.
2. **LM Studio**: applicazione desktop multiplatforma con interfaccia grafica intuitiva.
3. **Jan**: soluzione “plug and play” con interfaccia pulita, utile per chi non vuole usare il terminale.

3.Integrazione Framework di sviluppo

1. **LangChain + RAG pipeline:** per testare modelli in scenari complessi (retrieval + generazione).
2. **FAISS:** per indicizzazione e ricerca semantica su dataset locali.

Development: Docker



Uno strumento che facilita lo sviluppo e il deploy di applicazioni basate su intelligenza artificiale

Docker è un software che permette di creare, gestire e distribuire container, ovvero ambienti isolati all'interno di un sistema operativo padre (definito host) in cui eseguire applicazioni con tutte le loro dipendenze in modo indipendente.

Un container è un ambiente dedicato che contiene tutto ciò che serve per far funzionare una specifica applicazione: il codice sorgente, le librerie, le variabili d'ambiente, i file di configurazione, ecc. Questo rende possibile spostare facilmente le applicazioni da un ambiente all'altro, senza dover modificare il codice o configurare nuovamente il sistema operativo.

consente di sfruttare al meglio le risorse hardware disponibili, riducendo i tempi di avvio e il consumo di memoria consentendo di distribuire le applicazioni in modo rapido, sicuro e scalabile

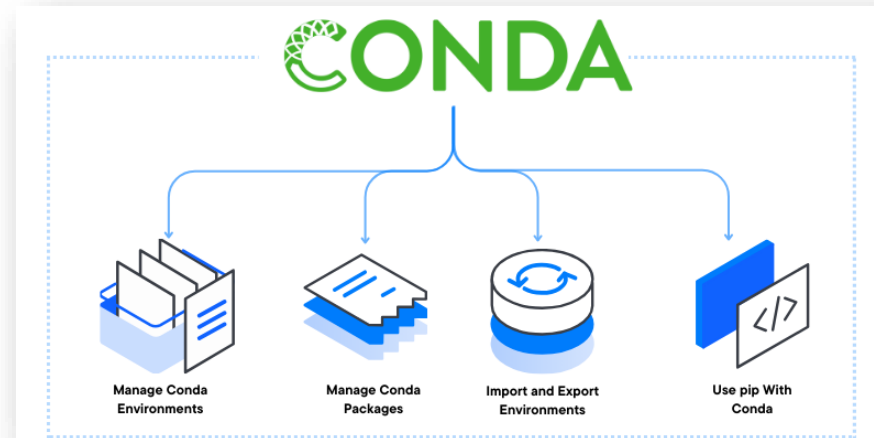
Development: Conda

Conda è un sistema di gestione dei pacchetti open source ed un sistema di gestione dell'ambiente per l'installazione di più versioni di pacchetti software e delle loro dipendenze.

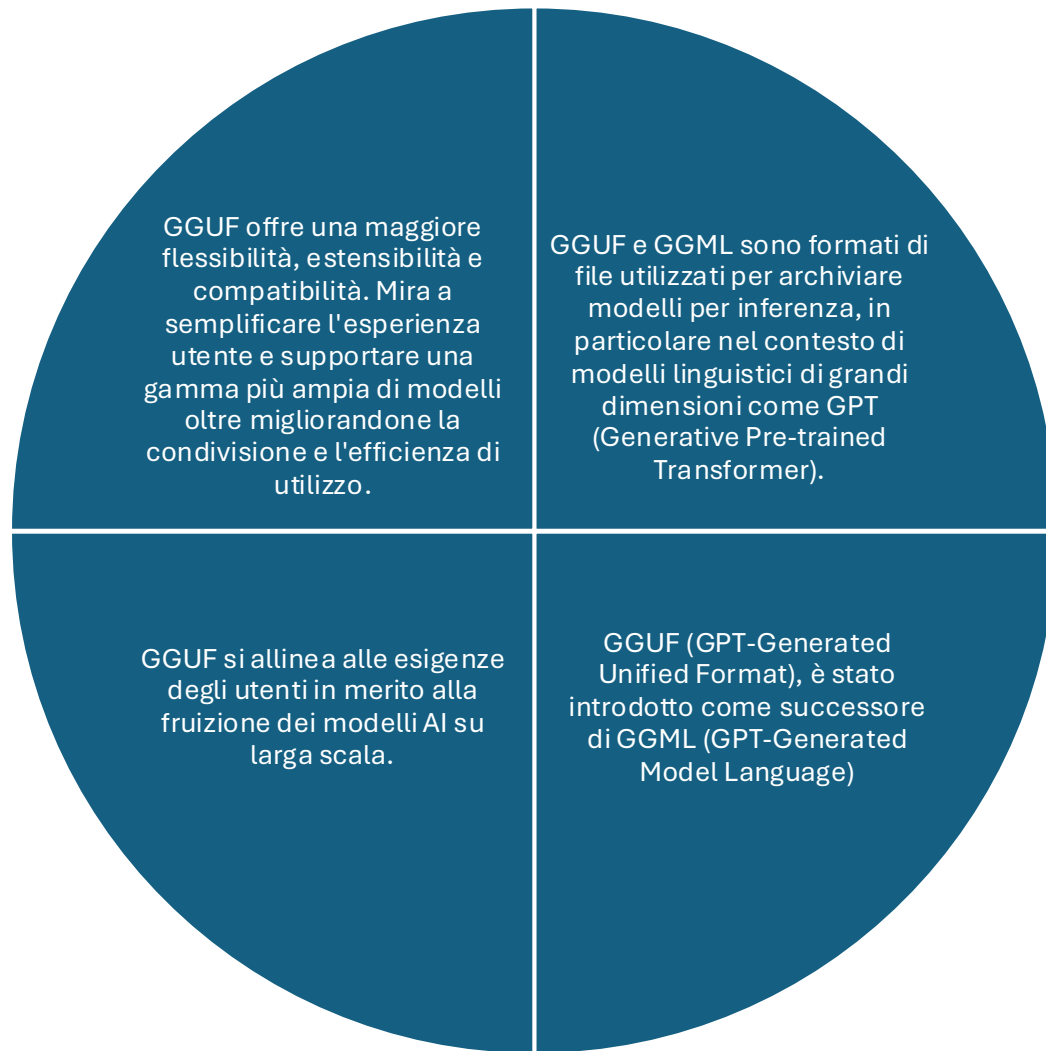
Esempio:

```
conda create --name mps python=3.13
```

```
conda activate mps
```



Development: gpt generated unified format (GGUF)



PRO:

- Risolve i limiti di GGML: GGUF è progettato per superare le carenze di GGML e migliorare l'esperienza utente.
- Estensibilità: consente l'aggiunta di nuove funzionalità mantenendo la compatibilità con i modelli precedenti.
- Stabilità: GGUF si concentra sull'eliminazione delle modifiche di interruzione, facilitando la transizione alle versioni più recenti dei modelli.
- Versatilità: supporta vari modelli, estendendosi oltre l'ambito dei modelli LLaMA.

CONTRO:

- Tempo di transizione: la conversione dei modelli esistenti in formato GGUF richiede molto tempo.
- Adattamento richiesto: utenti e sviluppatori devono abituarsi a questo formato.
- Framework che supportano il formato.

Development: Ollama



Progetto, costruito sul motore di esecuzione di modelli basati sull'architettura Transformers llama.cpp che permette di utilizzare modelli LLM offline attraverso una command line e mediante l'implementazione di API dedicate

Il catalogo dei modelli disponibili è reperibile online:

<https://ollama.com/library>



Supporto al
formato GGUF



Disponibile
come
container
docker

Development: Comandi Ollama

MODIFICA VARIABILI DI AMBIENTE:

```
export OLLAMA_ORIGINS="*"
export OLLAMA_HOST="127.0.0.1:11345"
export OLLAMA_MODELS=»custom_user_path»
```

ESEGUIRE IL SERVER

```
ollama serve
```

MOSTRARE MODELLI UTILIZZABILI

```
ollama list
```

Il funzionamento di ollama richiede un po' di familiarità con la riga di comando.

SCARICARE O AGGIORNARE UN MODELLO

```
ollama pull <model_name>
```

INIZIARE CHAT

```
ollama run <model_name>
```

ELIMINARE UN MODELLO

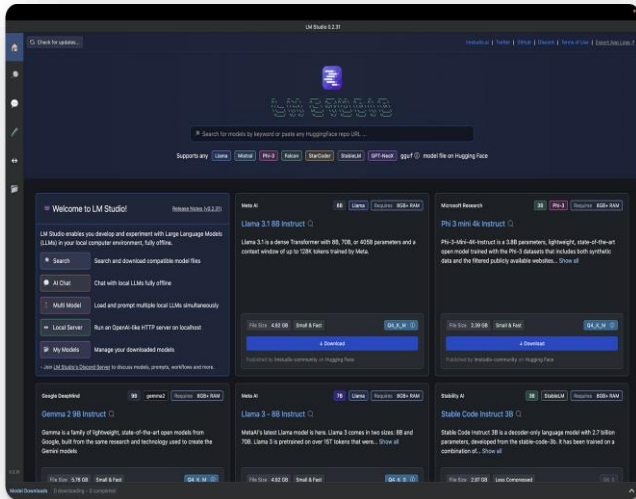
```
ollama rm <model_name>
```

MOSTRARE INFORMAZIONI SU UN MODELLO

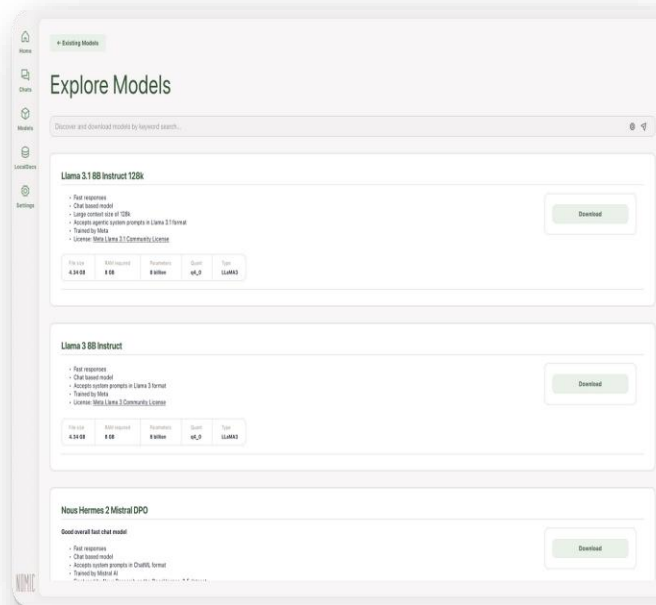
```
ollama show <model_name>
```



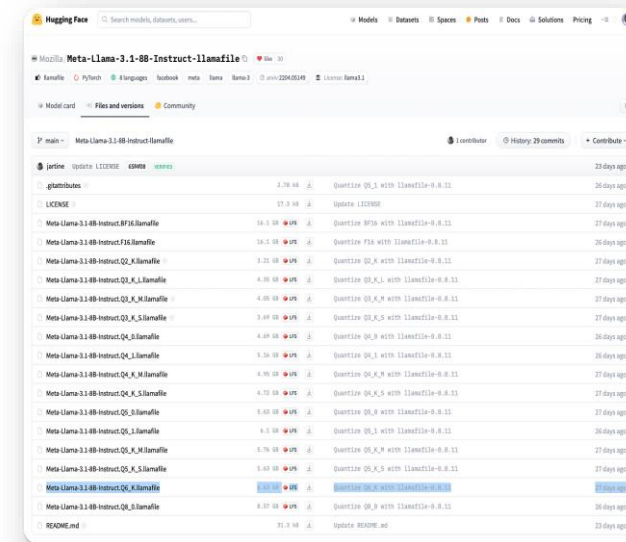
Development: Tools per eseguire modelli llm localmente



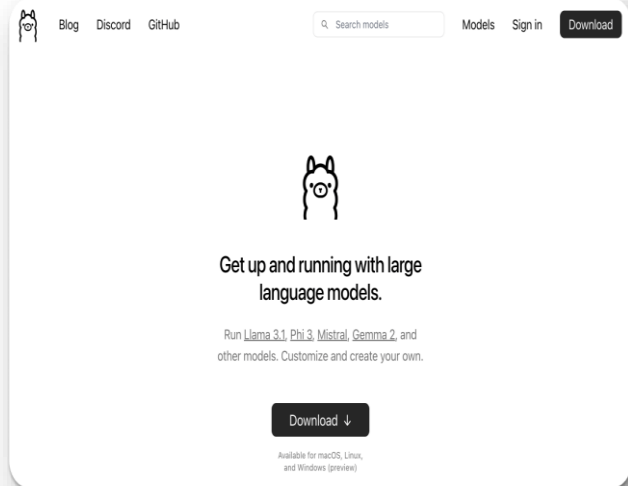
LM Studio



GPT4ALL



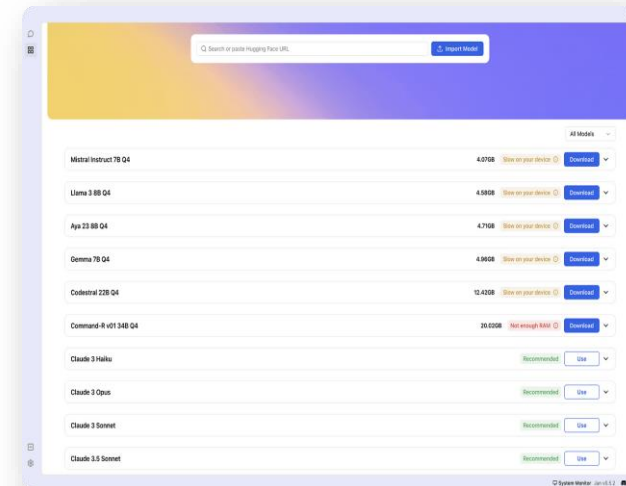
LLaMaFile



OLLAMA

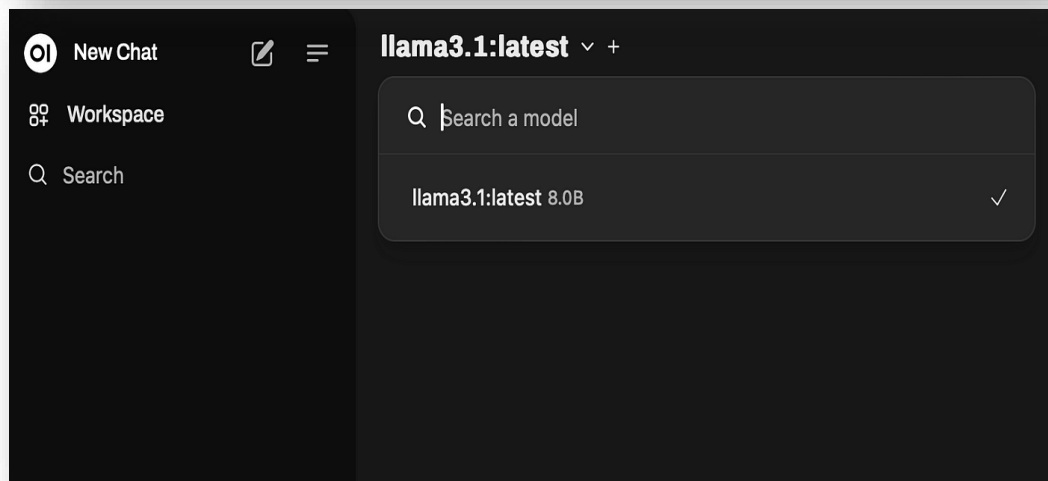
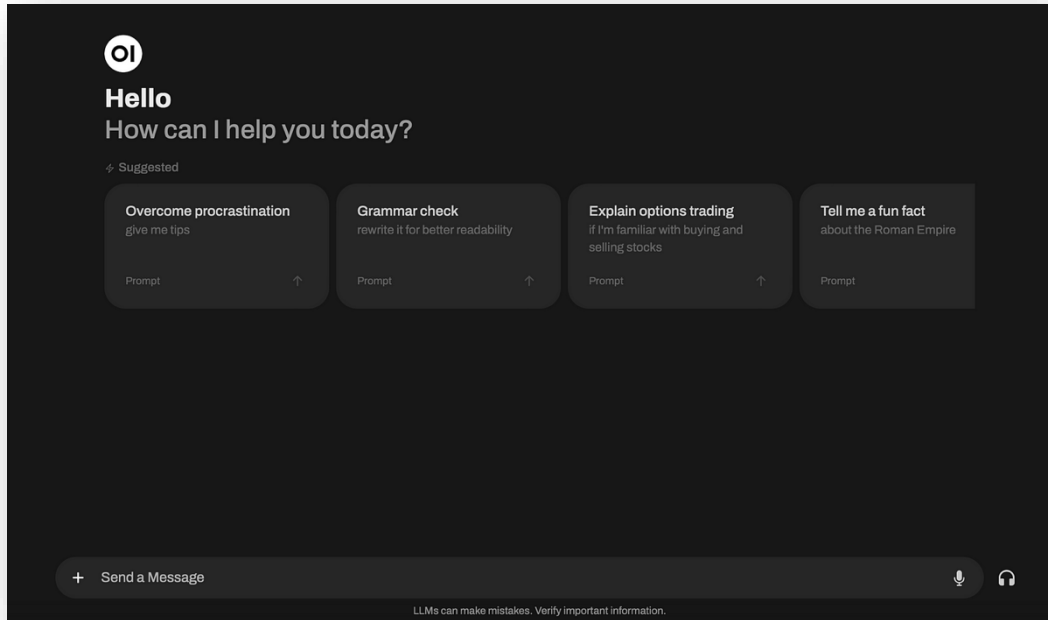


LLaMa.cpp

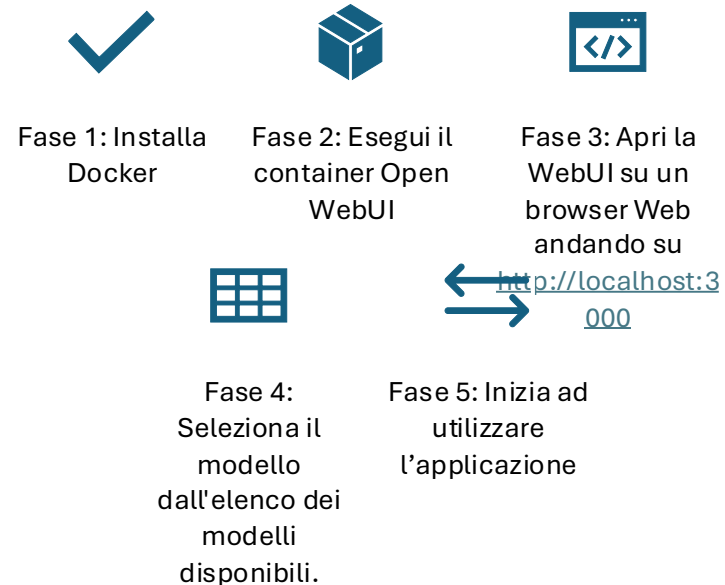


Jan

Development: Open webui



Open WebUI (precedentemente noto come Ollama WebUI) è un servizio eseguibile anche in locale che consente di interagire con differenti modelli tramite una interfaccia web dedicata



```
docker run -d -p 3000:8080 --add-host=host.docker.internal:host-gateway -v open-webui:/app/backend/data --name open-webui --restart always ghcr.io/open-webui/open-webui:main
```

Development: FAISS

FAISS (Facebook AI Similarity Search) è una libreria open-source sviluppata da Meta (Facebook AI Research) per eseguire ricerche di similarità e clustering su vettori ad alta dimensionalità in modo molto efficiente.

- ✓ Alta efficienza: ottimizzato in C++ con binding Python
- ✓ Scalabile: può gestire milioni o miliardi di vettori
- ✓ Supporto CPU e GPU: implementazioni CUDA per alte prestazioni
- ✓ Ricerca esatta e approssimata: trade-off tra accuratezza e velocità
- ✓ Libreria, non servizio: tipicamente usata embedded nelle applicazioni

Funzionalità in un'architettura RAG:

- memorizza gli embedding dei documenti
- indicizza i vettori
- recupera i k frammenti più semanticamente rilevanti
- passa questi frammenti come contesto



👉 La qualità del retrieval influisce direttamente sulla qualità della risposta del modello

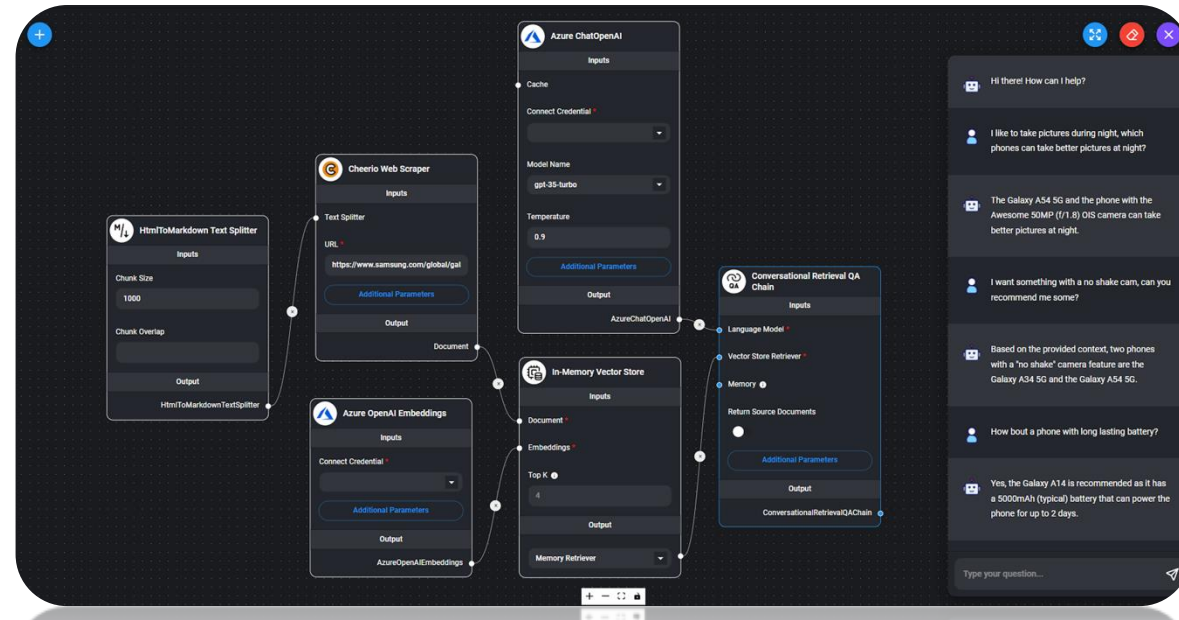
Development: Flowise



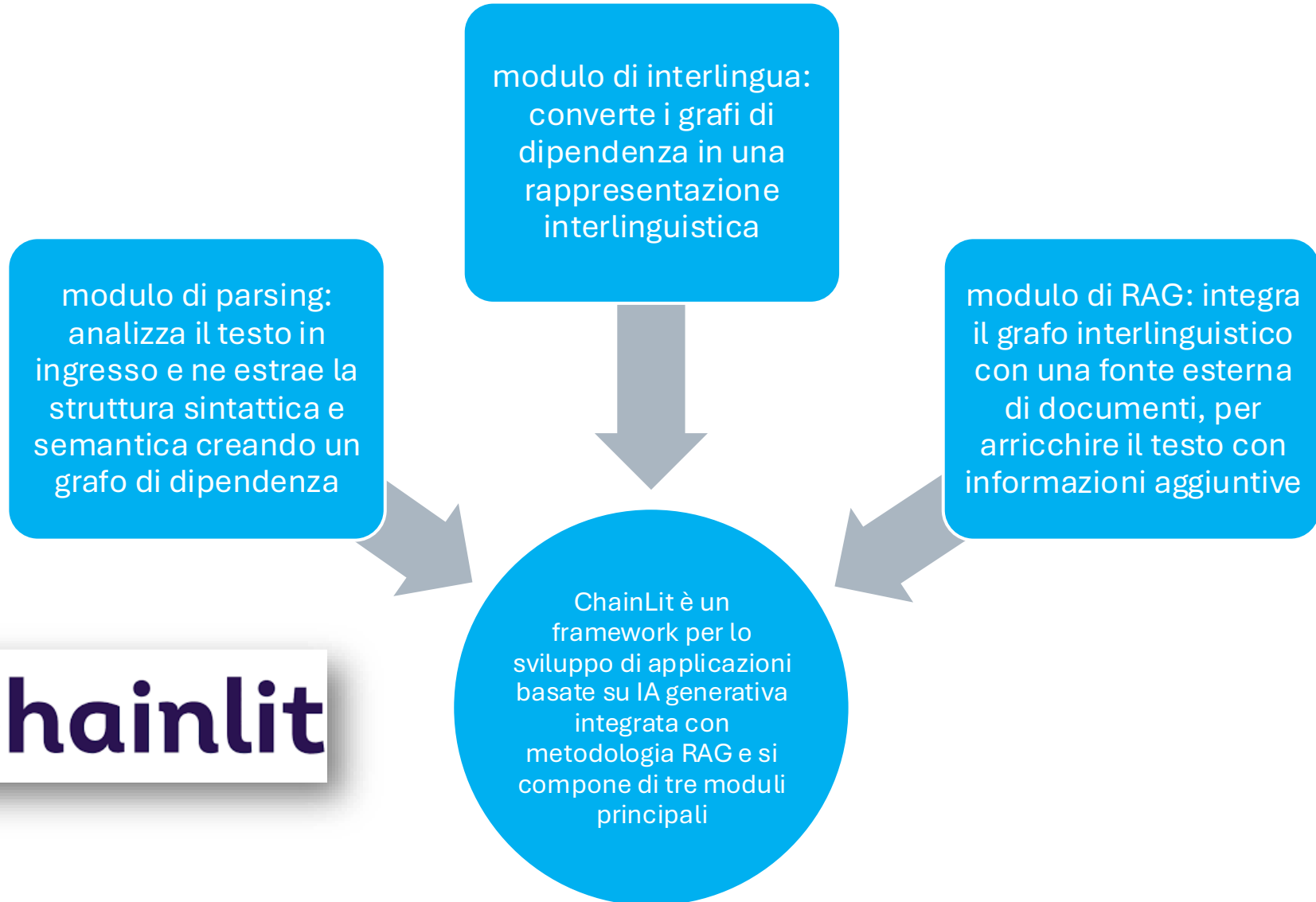
Funzionalità principali

- Generatore di app LLM senza scrittura di codice
- Creazione di agenti AI
- Orchestrazione LLM
- Integrazione di API e SDK
- Supporto per LLM open source

Flowise è uno strumento open source progettato per consentire agli sviluppatori di creare e gestire applicazioni basate su Large Language Model (LLM) tramite un approccio no/low-code tramite l'uso di tecnologie LLM, tra cui Langchain e LlamaIndex, e l'integrazione di oltre 100 servizi differenti.



Development: Chainlit



Development: Langchain

LangChain è un framework open source progettato per semplificare lo sviluppo di applicazioni basate su modelli linguistici di grandi dimensioni (LLM). Il suo obiettivo è orchestrare questi modelli integrandoli con dati esterni, API, strumenti e workflow complessi, così da creare applicazioni intelligenti come chatbot, sistemi di Q&A, assistenti virtuali e pipeline RAG.

Caratteristiche Chiave

- Open Source, disponibile in Python e JavaScript.
- Modularità: facile estensione e personalizzazione.
- Supporto multi-LLM: compatibile con modelli come GPT, LLaMA, Claude, ecc.
- Integrazione RAG: per risposte basate su dati reali.
- Scalabilità: progettato per applicazioni enterprise e workflow complessi.
- Ecosistema: include strumenti come LangGraph (per agenti persistenti) e LangSmith (debug e monitoraggio).

Architettura e Componenti Principali

- Chains (Catene): sequenze di passaggi che trasformano input in output complessi, orchestrando interazioni tra LLM e altre risorse.
- Agents: entità che scelgono dinamicamente quali strumenti o azioni eseguire, permettendo ragionamenti multi-step.
- Memory: gestione del contesto tra le interazioni, utile per conversazioni coerenti.
- Tools & Connectors: integrazione con database, API, knowledge base e servizi esterni.
- Document Loaders & Vector Stores: per implementare RAG, recuperando informazioni pertinenti da fonti esterne e fornendole al modello.

Development: LangGraph e fastMCP



LangGraph è una libreria open-source (dell'ecosistema LangChain) progettata per costruire workflow e agenti LLM stateful usando grafi invece di catene lineari. Permette di modellare il comportamento di un agente come una macchina a stati, con nodi, transizioni condizionali e cicli.

serve quando:

- un flusso non è lineare
- servono loop, retry, riflessione
- più agenti devono cooperare o competere
- lo stato deve persistere tra i passi

FastMCP è un framework Python-first per costruire server e client MCP. Implementa completamente lo standard MCP e ne astrae la complessità, permettendo di esporre tool, risorse e prompt a LLM e agenti.

serve quando si vuole:
collegare LLM o agenti a funzioni Python, dati o servizi
esporre capacità applicative come MCP server standard
evitare integrazioni custom e portare MCP dallo sviluppo alla produzione

Development: Langchain vs LangGraph

LangChain è ottimizzato per flussi di lavoro lineari e modulari in cui l'attenzione è rivolta a semplicità e velocità. Presenta un overhead di orchestrazione inferiore, rendendolo più veloce per attività leggere o semplici come pipeline di generazione con recupero aumentato (RAG) e chatbot semplici.

Aspect	LangChain AgentExecutor	LangGraph	Verdict
Workflow Type	Linear or modular simple tasks	Complex, branching, multi-agent workflows	LangChain better for simple; LangGraph for complex
State Management	External/manual memory	Native, built-in checkpointing and persistence	LangGraph wins
Error Handling	Custom logic	Built-in retries and fault tolerance	LangGraph more robust
Speed	Faster (less orchestration overhead)	Slightly slower for simple tasks	LangChain faster for lightweight
Multi-Agent Coordination	Possible but complex	Natively supported	LangGraph excels
Scalability	Moderate	High	LangGraph better for scale

LangGraph utilizza un modello di esecuzione basato su grafici progettato per un'elevata scalabilità e flussi di lavoro complessi, tra cui il coordinamento multi-agente e operazioni stateful di lunga durata. Offre checkpoint nativi, tolleranza agli errori con tentativi automatici e persistenza integrata di memoria/stato, che aggiungono un certo overhead ma migliorano l'affidabilità e la resilienza nei sistemi di produzione.

Development: Risorse Opensource per RAG

Fase RAG	Obiettivo	Strumenti Open Source consigliati
Framework / Orchestrazione	Comporre pipeline RAG	LangChain · LlamaIndex · Haystack
Orchestrazione avanzata / Agentic	Flussi stateful, branching, cicli	LangGraph
Ingestion dati	Caricare documenti e fonti	LangChain · LlamaIndex · Haystack · Firecrawl
Parsing / Document understanding	Estrarre testo e struttura	RAGFlow · LangChain · LlamaIndex
Chunking	Suddividere i documenti	LangChain · LlamaIndex · Haystack
Embedding	Testo → vettori	LangChain · LlamaIndex · Haystack
Vector Store / Index	Salvare e cercare embeddings	FAISS · Milvus · Weaviate · Qdrant · Chroma
Retrieval (top-k)	Recupero contesto rilevante	LangChain · LlamaIndex · Haystack
Reranking (opz.)	Migliorare i risultati	LangChain · LlamaIndex · Haystack
Prompt Augmentation	Costruire prompt contestuale	LangChain · LlamaIndex · Haystack
Generazione risposta	LLM + contesto	LangChain · LlamaIndex · Haystack
Controllo flusso / decisioni	Retry, fallback, multi-step	LangGraph
UI / Prototipazione	Interfaccia utente	Flowise · Dify · Chainlit
Evaluation & Monitoring	Qualità e groundedness	RAGAS · TruLens · DeepEval

Development: Esempio stack RAG Opensource

Uno stack di esempio per una RAG open source efficiente e controllabile è basato su

LangChain



LangChain

LangGraph



LangGraph

FAISS



RAGAS



LangChain costituisce il livello di orchestrazione applicativa, coordinando ingestion, chunking, embedding, retrieval e generazione

Quando il flusso RAG richiede logiche avanzate (branching, retry, routing, multi-step), LangChain viene esteso con LangGraph, che introduce un controllo stateful del processo

La ricerca semantica è affidata a FAISS, soluzione leggera e performante per similarity search, ideale in contesti on-prem o di ricerca

La qualità del sistema viene monitorata con RAGAS, che valuta retrieval e groundedness delle risposte, supportando test e miglioramento continuo



**THANK YOU
FOR YOUR
ATTENTION**