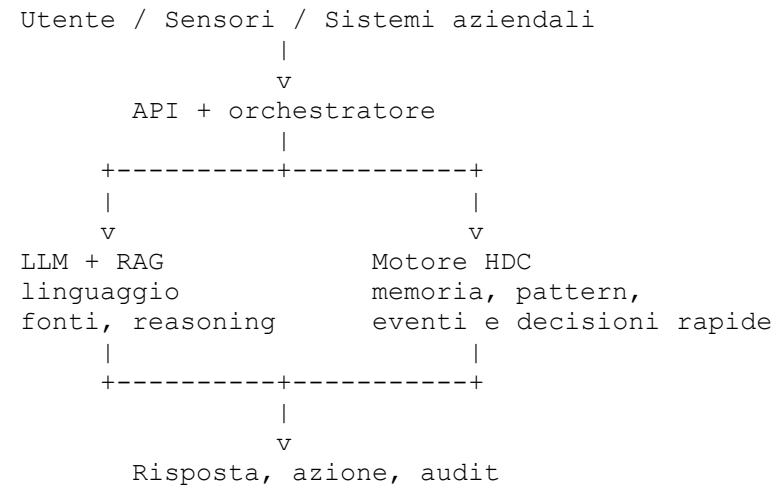
Un progetto **LLM + RAG + Vector DB + HDC** può essere costruito come un sistema a livelli: l'LLM parla, interpreta e ragiona; RAG recupera fonti verificabili; il database vettoriale trova contenuti semanticamente simili; HDC crea una memoria associativa veloce e robusta per segnali, eventi, pattern e decisioni locali. RAG collega un generatore a conoscenza esterna tramite una knowledge base, un retriever e un livello di integrazione/orchestrazione.

Architettura generale



L'obiettivo non è sostituire l'LLM con HDC: è assegnare a ciascun componente il compito per cui è più efficace. L'LLM non dovrebbe diventare il database aziendale né il processore di ogni segnale operativo; HDC non dovrebbe essere usato per produrre testi complessi o interpretare contratti lunghi in linguaggio naturale.

LLM

Un **Large Language Model** è il livello conversazionale e semantico del sistema. Riceve la domanda, comprende l'intenzione, usa gli strumenti disponibili e produce una risposta in linguaggio naturale.

Componenti interne

Componente	Funzione
Tokenizer	Divide testo, numeri e simboli in token gestibili dal modello
Token embedding	Trasforma ciascun token in un vettore numerico
Positional encoding	Comunica al modello l'ordine dei token
Transformer blocks	Elaborano il contesto attraverso attenzione e reti feed-forward
Self-attention	Pesa le parti più rilevanti della sequenza rispetto a ogni token
MLP / feed-forward	Trasforma e raffina le rappresentazioni prodotte dall'attenzione
Layer normalization	Stabilizza calcoli e addestramento
Residual connections	Conservano informazione e facilitano il flusso nei molti layer
Output head	Trasforma lo stato finale in probabilità sul token successivo

I Transformer moderni usano embedding, attenzione multi-head, reti feed-forward, normalizzazione e connessioni residue per costruire rappresentazioni contestuali del testo.[

Ruolo nel progetto

L'LLM dovrebbe:

- comprendere richieste in linguaggio naturale;
- classificare l'intento dell'utente;
- decidere se interrogare RAG, HDC, knowledge graph o sistemi esterni;
- combinare i risultati in una risposta utile;
- spiegare il risultato e citare le fonti recuperate;
- generare procedure, sintesi, report e istruzioni operative;
- mantenere il dialogo con l'utente.

Esempio:
"Perché la linea 3 ha avuto tre fermi negli ultimi due turni?"
L'LLM scompone la domanda, interroga la base documentale per procedure e manuali, chiede a HDC di confrontare il pattern degli allarmi con casi passati e restituisce una spiegazione verificabile.

RAG

RAG, Retrieval-Augmented Generation, è un'architettura che permette all'LLM di rispondere usando dati esterni aggiornabili, invece di affidarsi solo alla conoscenza contenuta nei suoi pesi. Le componenti principali sono knowledge base, retriever, livello di integrazione e generatore.

Componenti RAG

```
Documenti e dati
       |
       v
Ingestion e normalizzazione
       |
       v
Chunking + metadata
       |
       v
Embedding
       |
       v
Vector database / indice
       |
       v
Retriever + reranker
       |
       v
Prompt con fonti
       |
       v
LLM + risposta citabile
```

Componente	Funzione
Data connectors	Collegano file, cartelle, SharePoint, ERP, CRM, ticket, wiki, e-mail o database
Ingestion	Importa, pulisce e standardizza i contenuti
Chunking	Divide i documenti in parti brevi e semanticamente coerenti

Componente	Funzione
Metadata	Aggiunge autore, data, reparto, versione, permessi, macchina, sito e tipo di documento
Embedding model	Converte ogni chunk in un vettore semantico
Vector index	Indicizza gli embedding per ricerca rapida
Retriever	Seleziona i chunk più rilevanti alla domanda
Reranker	Riordina i risultati con un modello più preciso
Prompt builder	Inserisce domanda, istruzioni e fonti nel contesto dell'LLM
Generator	L'LLM formula la risposta
Citation layer	Mostra documento, pagina, estratto e livello di confidenza
Evaluation layer	Misura qualità del retrieval, correttezza, citazioni e tasso di allucinazioni

Ruolo nel progetto

RAG deve essere la **memoria documentale verificabile**:

- manuali tecnici;
- procedure operative;
- normative e documentazione compliance;
- ticket di assistenza;
- schede macchina;
- contratti;
- report storici;
- knowledge base aziendale.

La sequenza tipica è: la domanda viene trasformata in embedding, il motore cerca i chunk più vicini, i risultati migliori vengono aggiunti al prompt e l'LLM genera la risposta condizionata dalle fonti recuperate.

Vector database

Il **Vector DB** è il motore di ricerca semantica del RAG. Non salva solo parole chiave: salva vettori che rappresentano il significato di porzioni di testo, immagini, audio o dati strutturati.

Componenti

Componente	Funzione
Embedding vectors	Rappresentazioni numeriche di documenti e query
Index ANN	Indice di approximate nearest neighbors per trovare rapidamente vettori simili
Metadata store	Filtri per data, cliente, autorizzazione, reparto, tipo file e asset
Similarity metric	Misura la vicinanza tra query e documenti, ad esempio cosine similarity o dot product
Hybrid search	Combina ricerca vettoriale con keyword search
Filtering	Riduce risultati a documenti consentiti o pertinenti

Componente	Funzione
Reranking	Migliora la precisione sui risultati top-K
Versioning	Gestisce aggiornamenti, rimozioni e nuove versioni dei documenti
Access control	Applica i permessi utente prima del retrieval

Come funziona

Se un documento contiene: “Allarme di sovratemperatura del motore: verificare il circuito di raffreddamento”, il sistema produce un vettore, per esempio:

```
Manuale manutenzione → embedding [0.12, -0.44, 0.87, ...]
```

Se l’utente domanda: “Come intervenire quando il motore scalda troppo?”, anche la domanda viene trasformata in vettore. Il database confronta i due vettori e recupera il documento semanticamente vicino, anche se le parole non coincidono perfettamente.

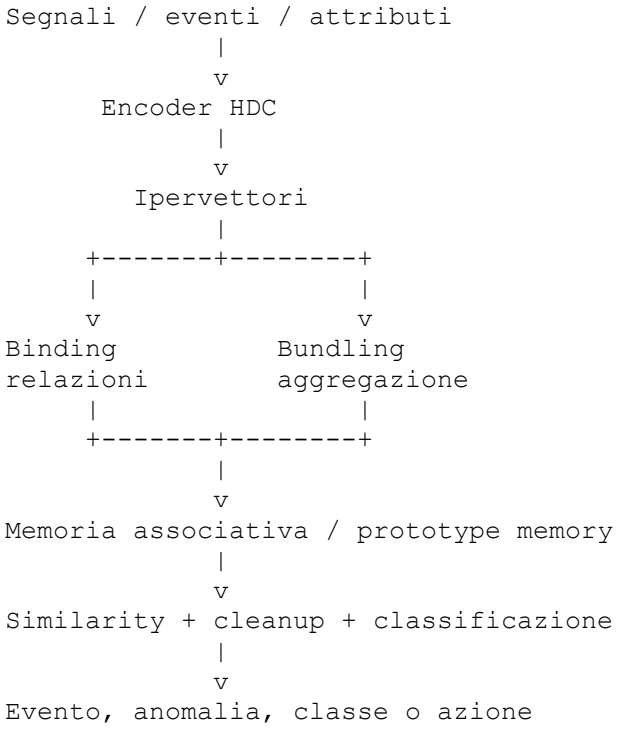
Il retrieval vettoriale normalmente usa una query embedding e una misura di similarità come cosine similarity o dot product per restituire i chunk top-K.

HDC

Hyperdimensional Computing è un paradigma computazionale che rappresenta elementi, relazioni ed eventi mediante ipervettori ad alta dimensionalità. È utile quando bisogna confrontare pattern, associare segnali, accumulare evidenze, classificare rapidamente e operare con tolleranza al rumore.

Le operazioni fondamentali ricorrenti nelle Vector Symbolic Architectures e nell’HDC includono encoding, binding, bundling, similarity e cleanup/associative memory.

Componenti HDC



Componente	Ruolo
Item memory	Dizionario di ipervettori casuali o appresi per simboli, attributi e categorie
Encoder	Converte dati grezzi in ipervettori
Continuous encoder	Trasforma valori numerici e segnali continui in rappresentazioni HDC
Spatial encoder	Codifica posizione, topologia, coordinate o relazione tra componenti
Temporal encoder	Rappresenta sequenze e ordine temporale degli eventi
Binding	Collega due concetti in una relazione, ad esempio “temperatura” + “alta”
Unbinding	Recupera una parte di una relazione codificata
Bundling	Aggrega molte evidenze o eventi in una singola memoria composita
Permutation	Mantiene ordine e posizione in sequenze o strutture
Prototype memory	Memorizza il profilo medio di una classe o condizione
Associative memory	Recupera il prototipo più simile a un input
Similarity engine	Calcola somiglianza, distanza di Hamming o cosine similarity
Cleanup memory	Corregge rappresentazioni rumorose verso il simbolo o prototipo più vicino
Classifier	Decide classe, anomalia, stato o azione
Online learning	Aggiorna prototipi progressivamente con nuovi esempi validati

Operazioni essenziali

Encoding

Trasforma un elemento in un ipervettore. Ad esempio:

```
Sensore temperatura → H_temperatura
Valore elevato      → H_alto
Motore 3            → H_motore3
```

Binding

Unisce ruolo e valore per preservare una relazione:

```
H_temperatura ⊗ H_alto
```

Il risultato rappresenta il fatto: **temperatura = alta**.

Bundling

Somma o aggrega più fatti:

```
H_evento =
bundle(
  H_motore3 ⊗ H_temperatura ⊗ H_alto,
  H_motore3 ⊗ H_vibrazione ⊗ H_alta,
  H_turno_notte
)
```

Il risultato è un’impronta compatta dell’evento operativo.

Permutation

Introduce l'ordine nella sequenza:

```
permute(H_evento_t-2)
+ permute²(H_evento_t-1)
+ H_evento_t
```

In questo modo HDC può distinguere "temperatura alta, poi vibrazione alta" da "vibrazione alta, poi temperatura alta".

Similarity e classificazione

L'ipervettore dell'evento corrente viene confrontato con prototipi memorizzati:

```
similarity(H_evento_corrente, H_prototipo_guasto_cuscinetto)
```

Se la somiglianza supera una soglia, il motore può segnalare un possibile guasto, classificare l'evento o inviare il risultato all'LLM per una spiegazione leggibile.

L'implementazione hardware HDC può sfruttare operazioni altamente parallele di binding, bundling, permutation e similarity, con potenziale vantaggio di latenza e consumo.

Ruolo di HDC nel progetto

HDC è più utile quando i dati sono dinamici, rumorosi, multimodali o localizzati sull'edge:

- classificazione rapida di eventi industriali;
- rilevamento anomalie;
- analisi di sequenze di allarmi;
- riconoscimento di pattern di guasto;
- fusione di sensori;
- memoria operativa di eventi recenti;
- personalizzazione locale;
- inferenza su FPGA o acceleratore dedicato;
- selezione preventiva di ciò che merita una chiamata costosa all'LLM.

Non è necessario inviare ogni misura a un LLM. HDC può filtrare e sintetizzare migliaia di eventi in un numero ridotto di stati, alert o pattern significativi.

Orchestratore

L'**orchestratore** è il componente che coordina i moduli. Può essere un workflow engine, un layer software con regole e strumenti, oppure un agente governato da policy.

Funzioni

- autentica l'utente e applica i permessi;
- comprende il tipo di richiesta;
- sceglie il percorso: RAG, HDC, database, API o combinazione;
- definisce soglie di confidenza;
- limita l'accesso ai dati sensibili;
- prepara il prompt per l'LLM;

- conserva traccia delle fonti e delle azioni;
- gestisce fallback in caso di risposta incerta;
- registra feedback e correzioni per migliorare il sistema.

Esempio di routing

Tipo di richiesta	Modulo primario	Modulo di supporto
“Qual è la procedura per sostituire il filtro?”	RAG	LLM
“Questo pattern indica un guasto?”	HDC	LLM
“Quali macchine hanno avuto questo pattern?”	HDC + database eventi	RAG
“Prepara un report sugli ultimi fermi”	Database + HDC	LLM
“Quale norma si applica a questo processo?”	RAG	LLM
“Spiega perché è stato generato l’allarme”	HDC	RAG + LLM

Flusso operativo completo

Caso 1: domanda documentale

```
Utente: “Come si calibra il sensore di pressione?”
        |
        v
LLM interpreta la domanda
        |
        v
RAG ricerca manuali e procedure autorizzate
        |
        v
Vector DB recupera i chunk più pertinenti
        |
        v
Reranker seleziona le fonti migliori
        |
        v
LLM formula istruzioni con fonti e avvertenze
```

Caso 2: anomalia industriale

```
Sensori: temperatura, vibrazione, corrente, stato macchina
        |
        v
Encoder HDC costruisce ipervettori per ciascun evento
        |
        v
Binding + bundling + temporal encoding
        |
        v
Confronto con prototipi di funzionamento e guasto
        |
        v
HDC segnala: “pattern compatibile con usura cuscinetto”
        |
        v
RAG recupera manuale e storico manutenzioni
        |
        v
```

```
LLM spiega causa, evidenze e azione suggerita
```

Componenti software

Per una prima versione realistica, il sistema può includere:

- API gateway e gestione identità;
- interfaccia chat o dashboard;
- LLM locale, in private cloud o tramite API controllata;
- embedding model;
- pipeline ETL e document ingestion;
- vector database;
- database relazionale per utenti, log e metadata;
- object storage per documenti;
- motore HDC software;
- stream processor per sensori ed eventi;
- orchestratore dei workflow;
- sistema di valutazione e osservabilità;
- audit log e gestione dei permessi.

Componenti hardware

La roadmap hardware può procedere gradualmente:

Fase	Implementazione	Obiettivo
Proof of concept	CPU/GPU + HDC software	Validare use case e benchmark
Edge prototype	Mini PC industriale / GPU compatta / FPGA	Ridurre latenza e invio dati al cloud
HDC accelerator	FPGA ottimizzata	Accelerare encoding, binding, bundling e similarity
IP core	Blocco integrabile in SoC/OEM	Licensing a produttori e integratori
Architettura avanzata	Compute-in-memory o fotonica, se giustificata	Massimizzare parallelismo ed efficienza energetica

L’HDC è particolarmente adatto a una futura accelerazione perché le sue primitive possono essere rese parallele e integrate in architetture associative o in-memory.

Separazione dei ruoli

Livello	Fa bene	Non dovrebbe fare
LLM	Linguaggio, sintesi, ragionamento, orchestrazione, spiegazione	Essere l’unico database o processare ogni evento raw
RAG	Risposte basate su documenti aggiornati e citabili	Classificazione real-time di segnali
Vector DB	Ricerca semantica su documenti e contenuti	Ragionamento complesso o decisione finale

Livello	Fa bene	Non dovrebbe fare
HDC	Pattern, sequenze, anomalie, memoria associativa, edge inference	Generazione linguistica lunga e interpretazione di documenti complessi
Knowledge graph	Relazioni esplicite, vincoli, tracciabilità	Retrieval semantico ad alta scala da solo
Orchestratore	Routing, policy, sicurezza, workflow e audit	Sostituire i modelli specialistici

Come presentarlo

:

“L’LLM rende il sistema comprensibile e utilizzabile in linguaggio naturale. Il RAG collega le risposte a fonti aziendali aggiornate e verificabili. Il database vettoriale consente di ritrovare rapidamente conoscenza per significato. HDC opera come memoria associativa e motore di decisione efficiente per eventi, pattern e inferenza locale. L’orchestratore integra i moduli, applica sicurezza e produce risposte spiegabili.”

Il vantaggio competitivo non è possedere singolarmente un LLM, un vector database o un algoritmo HDC. È costruire un’architettura proprietaria in cui questi elementi cooperano, con metriche misurabili su accuratezza, latenza, costo per decisione, consumo energetico e qualità delle risposte.