Costruzione sperimentale di un sistema RAG locale con Ollama, PostgreSQL e pgvector

Marco Barbato

24 maggio 2026

Sommario

Questo articolo descrive un esperimento di utilizzo di LLM per effettuare una ricerca su una knowledge base documentale. Qui si chiarisce molto bene cosa significhi ricerca nello spazio degli embedding: passiamo dalla ricerca lessicale tipica dei DBMS attraverso l'operatore LIKE alla ricerca per vicinanza semantica ottenuta come una metrica di uno spezio vettoriale.

RAG – Retrieval-Augmented Generation – utilizza gli LLM senza doverli esplicitamente istruire sui contenuti del documento ma dandogli degli spunti del tipo: "questi frammenti sono corrispondenti a vettori che sono vicini al prompt, prendili in considerazione nella produzione dell'output".

Il lavoro mostra inoltre come un sistema RAG possa essere interpretato come un'evoluzione dei classici sistemi di *information retrieval* documentale, nei quali l'indicizzazione lessicale viene sostituita da una rappresentazione geometrica del significato del testo.

1 Introduzione

Nella mia ricerca sul funzionamento degli LLM ho voluto provare a immaginare un utilizzo di un modello di linguaggio per aiutarmi a cercare informazioni in un archivio di file locale.

Qeusta tecnica è detta RAG (Retrieval-Augmented Generation), che è un'architettura ibrida che combina:

- un database vettoriale contenente embedding semantici;
- un Large Language Model (LLM);
- una pipeline di retrieval semantico.

L'idea fondamentale è relativamente semplice: invece di affidarsi esclusivamente alla conoscenza appresa durante l'addestramento del modello linguistico, si fornisce dinamicamente al modello un contesto recuperato da documenti esterni.

L'alternativa consiste nell'addestrare da zero una rete neurale su un corpus molto ampio di documenti. È ciò che hanno fatto aziende come OpenAI e Anthropic, operando però su scala gigantesca.

Anche aziende italiane come Engineering Group hanno sviluppato modelli proprietari addestrati internamente, ma un progetto di questo tipo va considerato di livello enterprise, poiché richiede il coinvolgimento di numerose competenze specialistiche (ML Engineers, Data Engineers, DevOps Engineers, esperti infrastrutturali, ecc.).

L'architettura RAG adottata in questo lavoro, invece, non prevede alcun addestramento della rete neurale. Essa utilizza un LLM già addestrato, arricchendo dinamicamente il prompt con informazioni e citazioni ottenute tramite una ricerca semantica preliminare su un database vettoriale.

Nel presente lavoro viene descritto un piccolo prototipo sperimentale sviluppato in ambiente Linux utilizzando Ubuntu Linux, PostgreSQL con estensione pgvector; Ollama come runtime locale per l'esecuzione dei modelli (`phi3:mini` per l'inferenza e `nomic-embed-text` come modello di embedding per la vettorizzazione dei documenti; Python per la realizzazione della pipeline.

L'intero sistema è stato eseguito su una macchina relativamente modesta: un laptotp HO con CPU Intel Core i5 con 16 GB RAM, *senza* GPU CUDA.

L'obiettivo dell'esperimento è quello di comprendere il funzionamento interno di un sistema RAG ma nello stesso tempo individuare i colli di bottiglia computazionali, le problematiche pratiche di retrieval e generazione e, sempre molto interessante, gli effetti della limitazione hardware. Per questo ho inserito anche diverse misurazioni dei tempi di esecuzione dei vari step.

2 Architettura generale del sistema

Il sistema realizzato può essere schematizzato come in figura 1.

Figura 1: Pipeline RAG ingestion e inferenza

3 Ingestione dei documenti

La pipeline di ingestione svolge i seguenti passi:

1. lettura dei documenti PDF;

2. estrazione del testo;

3. suddivisione in chunk;

4. generazione degli embedding;

5. salvataggio nel database PostgreSQL.

La generazione degli embeddings è la parte cruciale: utilizziamo un altro modello, che non phi3:mini ma `nomic-embed-text` che ho installato "a fianco" di phi3-mini

```
ollama pull nomic-embed-text
```

Questa è la parte cruciale del processo. Utilizziamo una rete neurale per indicizzare documenti. Mi sovviene l'utilizzo che feci di Lucene molti anni fa.

3.1 Dal retrieval lessicale al retrieval semantico

L'idea di effettuare una ricerca preliminare su una collezione di documenti prima di produrre una risposta non è nuova. Già da molti anni esistono motori di indicizzazione documentale come Apache Lucene, Solr ed Elasticsearch, utilizzati per realizzare motori di ricerca testuale ad alte prestazioni.

Questi sistemi tradizionali costruiscono tipicamente un indice invertito, associando ad ogni termine la lista dei documenti nei quali esso compare: un indice analitico classico. La ricerca avviene quindi prevalentemente tramite similarità lessicale, utilizzando tecniche come tokenizzazione, stemming, TF-IDF o BM25.

L'approccio RAG può essere visto come una naturale evoluzione di questi sistemi: l'indicizzazione non avviene più nello spazio discreto delle parole, ma in uno spazio vettoriale continuo costruito tramite embedding neurali.

In questo contesto ogni frammento di testo (*chunk*) viene rappresentato da un vettore numerico ad alta dimensionalità. La ricerca non avviene più chiedendosi:

> "quali documenti contengono queste parole?"

ma piuttosto:

> "quali documenti hanno significato semanticamente vicino alla richiesta dell'utente?"

Il database vettoriale svolge quindi un ruolo concettualmente simile a quello degli storici motori di ricerca documentale, ma utilizzando una misura di similarità geometrica nello spazio degli embedding invece della sola corrispondenza lessicale.

Il DBMS utilizzato nel progetto è PostgreSQL. Si tratta di una piattaforma estremamente versatile che avevo già impiegato in passato anche in ambito GIS e nell'elaborazione di immagini satellitari.

Per molti anni PostgreSQL è rimasto in secondo piano – nei progetti che ho seguito – rispetto ad altre soluzioni molto diffuse come MySQL e Oracle, ma nel tempo ha acquisito un ruolo sempre più importante grazie alla ricchezza delle estensioni disponibili e alla sua notevole flessibilità architetturale.

In particolare, estensioni come PostGIS ne hanno fatto uno strumento di riferimento per l'elaborazione geografica, mentre oggi pgvector consente di utilizzarlo efficacemente anche come database vettoriale per applicazioni AI e sistemi RAG.

La struttura semplificata del database è la seguente:

```
CREATE TABLE documents (
   id SERIAL PRIMARY KEY,
   path TEXT
);

CREATE TABLE chunks (
   id SERIAL PRIMARY KEY,
   document_id INTEGER,
   chunk_index INTEGER,
   content TEXT,
   embedding VECTOR
);
```

L'estensione `pgvector` permette di memorizzare direttamente vettori numerici all'interno di PostgreSQL.

Ogni chunk viene trasformato in un vettore numerico tramite il modello embedding locale.

Il retrieval successivo utilizza operatori di distanza vettoriale come:

```
embedding <-> query_embedding
```

che rappresenta la distanza semantica tra due vettori. La similarità vettoriale svolge un ruolo analogo a quello dell'operatore `LIKE` nei database relazionali, ma operando sul significato semantico del testo anziché sulla semplice corrispondenza lessicale.

4 Embedding e retrieval semantico

Il cuore concettuale del sistema è costituito dagli embedding. Premetto che i pochi documenti indicizzati parlavano di numeri complessi e di fisica delle alte energie, si tratta di alcuni articoli e libri digitalizzati che ho utilizzato come banco di prova.

Durante le prime sperimentazioni è emerso chiaramente che la fase di ingestione documentale rappresenta una componente computazionalmente significativa dell'intera architettura RAG.

La generazione degli embedding per migliaia di chunk testuali richiede infatti numerose operazioni di inferenza neurale e può comportare tempi di elaborazione non trascurabili anche su corpus documentali relativamente piccoli, soprattutto in presenza di hardware privo di GPU dedicate.

In una prima implementazione il processo di ingestione prevedeva una ricostruzione completa del database vettoriale ad ogni esecuzione. L'aumento progressivo della base documentale ha però reso necessario riprogettare il software in modo *incrementale*, evitando la rielaborazione dei documenti già indicizzati.

Questa ottimizzazione ha ridotto drasticamente i tempi di aggiornamento della knowledge base, avvicinando l'architettura a problematiche tipiche dei sistemi enterprise di information retrieval.

Un'ulteriore criticità emersa durante le sperimentazioni riguarda l'ingestione di documenti ottenuti tramite *scansione OCR* (tipicamente le trascrizioni in lettering moderno delle opere originali di Bombelli e Tartaglia sull'algebra).

Molti file PDF non contengono infatti testo nativo, ma semplicemente immagini delle pagine. In questi casi il sistema deve eseguire preventivamente un riconoscimento ottico dei caratteri (OCR) prima di poter effettuare il chunking e il calcolo degli embedding.

La qualità dell'OCR influisce direttamente sulla qualità della ricerca semantica successiva. Errori di riconoscimento, caratteri corrotti o impaginazioni particolarmente complesse possono infatti degradare la rappresentazione vettoriale del testo e ridurre l'efficacia del retrieval.

Questo aspetto evidenzia come, nei sistemi RAG reali, la qualità della pipeline di acquisizione documentale sia cruciale tanto quanto la qualità del modello linguistico utilizzato.

Per avere un'idea di quanto la fase di *ingestion* sia preponderante in tutto il processo basti pensare che per indicizzare 73 documenti, producendo 16242 chuncks, il mio HP ha lavorato per tre ore esclusivamente su questo task facendo lavorare costantemente tutte i 10 core dell'Intel i5.

Ora veniamo a come viene matematicamente trattata l'*ingestion*. Il testo:

```
"numeri complessi"
```

non viene confrontato lessicalmente con i documenti.

Viene invece trasformato in un vettore:

$$\vec{x} = (x_1, x_2, \ldots, x_n)$$

all'interno di uno spazio vettoriale ad alta dimensionalità.

La ricerca non avviene quindi tramite parole identiche, ma tramite vicinanza geometrica nello spazio degli embedding.

In questo senso il database vettoriale si comporta come una memoria semantica.

5 Costruzione del prompt

Una volta recuperati i chunk più simili semanticamente alla domanda, il sistema costruisce un prompt arricchito.

Un esempio semplificato è il seguente:

```
Sei un assistente tecnico che risponde usando solo il contesto fornito.

CONTESTO:
[FONTE 1]
...

[FONTE 2]
...

DOMANDA:
Quali argomenti tratta il documento sui numeri complessi?

RISPOSTA:
```

Il modello linguistico non interroga direttamente il database.

Riceve invece:

- il contesto recuperato;
- la domanda originale;
- alcune istruzioni.

Questo è precisamente il significato di Retrieval-Augmented Generation.

6 Prestazioni osservate

Sono stati effettuati vari test sperimentali.

Il database conteneva:

- 73 documenti;

- 16242 chunk indicizzati.

La temporizzazione del retrieval ha fornito risultati molto interessanti:

```
[START] embedding domanda
[END] embedding domanda: 0.18 s

[START] query vettoriale PostgreSQL
[END] query vettoriale PostgreSQL: 0.17 s
```

Il retrieval semantico tramite PostgreSQL e pgvector è quindi risultato estremamente rapido.

Il vero collo di bottiglia si è invece manifestato nella generazione finale della risposta da parte del modello locale.

7 Problemi incontrati

7.1 Lentezza della generazione

La generazione finale della risposta è risultata molto lenta.

La causa principale non era PostgreSQL, bensì il modello linguistico eseguito localmente su CPU.

L'elaborazione di prompt lunghi richiede infatti:

- tokenizzazione;
- moltiplicazioni matriciali;
- calcolo dell'attenzione;
- generazione autoregressiva token per token.

In assenza di GPU CUDA i tempi crescono rapidamente.

8 Ottimizzazioni introdotte

8.1 Riduzione del contesto

Per ridurre il carico computazionale è stato ridotto il numero di chunk recuperati:

```
TOP_K = 3
```

invece di:

```
TOP_K = 5
```

Questo riduce:

- la lunghezza del prompt;
- il numero di token elaborati;
- il tempo di inferenza.

8.2 Limitazione dell'output

È stato inoltre limitato il numero massimo di token generabili:

```
"num_predict": 300
```

per evitare:

- risposte eccessivamente lunghe;
- loop generativi;
- timeout.

8.3 Retry automatico

Sono stati introdotti meccanismi di retry automatico in caso di timeout HTTP verso Ollama.

9 Allucinazioni e qualità delle risposte

Le prime risposte ottenute presentavano:

- allucinazioni;
- contaminazioni semantiche;
- riutilizzo improprio di frammenti del prompt.

In particolare una richiesta troppo generica come:

```
"riassumi cosa contiene questa knowledge base"
```

ha prodotto risultati incoerenti.

Questo è comprensibile, poiché il sistema RAG recupera soltanto pochi chunk semanticamente vicini alla domanda.

Domande più specifiche hanno invece prodotto risultati significativamente migliori.

Ad esempio:

```
Quali argomenti tratta il documento sui numeri complessi?
```

ha restituito una risposta coerente con i documenti realmente presenti nella knowledge base.

10 Output di un paio di esperimenti

10.1 Qual è il ruolo di Bombelli nella scoperta dei numeri complessi?

Qui vediamo che la ricerca nel db individua alcuni documenti che rispettano il contesto della domanda:

```
python ask.py "Qual è il ruolo di Bombelli nella scoperta dei numeri ocmplessi?"
[START] embedding domanda
[END] embedding domanda: 0.38 s
[START] query vettoriale PostgreSQL
[END] query vettoriale PostgreSQL: 0.07 s
```

Chunk recuperati:

[FONTE 1] /home/marcob/complessi/.tex/complessi.pdf | chunk 23 | distanza 11.6222
[FONTE 2] /home/marcob/complessi/tmp/complessi_con_note.pdf | chunk 80 | distanza 11.7777
[FONTE 3] /home/marcob/complessi/tmp/complessi.pdf | chunk 80 | distanza 11.7777
[FONTE 4] /home/marcob/complessi/.tex/complessi.pdf | chunk 80 | distanza 11.8213
[FONTE 5] /home/marcob/complessi/tmp/complessi_con_note.pdf | chunk 84 | distanza 12.0363

Risposta:

[START] Passo il prompt a Ollama
[WARN] Timeout Ollama tentativo 1
[END] Passo il prompt a Ollama: 386.64 s
[DEBUG] Lunghezza prompt: 7099 caratteri
Sei un assistente tecnico che risponde usando solo il contesto fornito.

Regole:
- Se il contesto non contiene abbastanza informazioni, dillo chiaramente.
- Non inventare informazioni.
- Cita sempre le fonti usando [FONTE 1], [FONTE 2], ecc.
- Rispondi in italiano.

CONTESTO:
[FONTE 1]
File: /home/marcob/complessi/.tex/complessi.pdf
Chunk: 23
Distanza: 11.622213050088208

do

1.1. HISTORIA MAGISTRA VITÆ?
11
furono scoperti e succede in continuazione , non è dovuta a ignoranza, ma a
un ostacolo epistemologico radicato nell'intuizione legata ai numeri razionali e
irrazionali.
Si può fare anche un esempio speculare in Fisica: la transizione dal mo-
dello geocentrico di Tolomeo a quello eliocentrico di Copernico fu ostacolata
da credenze radicate nelle osservazioni sensoriali e nelle concezioni filosofiche
dell'epoca.
Durante il Rinascimento, i numeri complessi non erano ancora riconosciuti
come entità matematiche a sé stanti. I principali ostacol
Nel Cinquecento, Girolamo Cardano ha introdotto i numeri complessi come soluzioni
alle equazioni cubiche. Tuttavia, nonostante avesse conosciuto la loro esistenza e
le potenzialità che offrivano per risolvere problemi matematici precedentemente
insolubili in R (il campo dei numeri reali), ha ignorato i risultati di Rafael
Bombelli, un matematico italiano del XVI secolo. In particolare, nel suo lavoro
"Introductio ad Analysin Infinitas", pubblicato postumo nel 1572, Bombelli
aveva già affrontato l'ipotesi che il quadrato negativo potesse avere
radici reali e avrebbe trovato soluzioni complesse. Nonostante questo
contributo significativo alla matematica dei numeri complessi, Cardano non ha

```
riconosciuto la validità delle sue scoperte di Bombelli o le applicazioni pratiche
che potevano avere per il calcolo e l'ottimizzazione in vari campi. Questo
atteggiamento contraddittorio dell'epoca è stato un ostacolo epistemologico
radicato nell'intuizione legata ai numeri razionali e irrazionali, come descritto
nel contesto fornito.
```

L'esempio mostra bene come il modello possa produrre concatenazioni narrative apparentemente plausibili ma storicamente scorrette. Il fatto che Cardano "ignori" Bombelli è evidentemente impossibile sul piano cronologico, ma statisticamente plausibile dal punto di vista linguistico. Poi Bombelli non ha mai scritto *Introductio ad Analysin Infinitas* ma ha scritto *L'Algebra*.

Questo mette in evidenza una caratteristica fondamentale dei LLM: il loro "ragionamento" non è storico o semantico nel senso umano del termine, ma consiste nella costruzione probabilistica della sequenza di token più plausibile dato il contesto.

10.2 Quale contributo è dovuto a Bruno Rossi nello studio dei raggi cosmici?

```
$ python ask.py "Quale contributo è dovuto a Bruno Rossi nello studio
dei ragggi cosmici?"
[START] embedding domanda
[END] embedding domanda: 0.38 s
[START] query vettoriale PostgreSQL
[END] query vettoriale PostgreSQL: 0.07 s

Chunk recuperati:

[FONTE 1] /home/marcob/Fisica/HEP/muons/muons (1).pdf | chunk 1 | distanza 11.7438
[FONTE 2] /home/marcob/Fisica/HEP/muons/muons.pdf | chunk 1 | distanza 11.8629
[FONTE 3] /home/marcob/Fisica/HEP/muons/muons (1).pdf | chunk 18 | distanza 12.3548
[FONTE 4] /home/marcob/Fisica/HEP/muons/muons.pdf | chunk 22 | distanza 12.6827
[FONTE 5] /home/marcob/complessi/.tex/complessi.pdf | chunk 14 | distanza 12.8954

Risposta:

[START] Passo il prompt a Ollama
[WARN] Timeout Ollama tentativo 1
[END] Passo il prompt a Ollama: 382.56 s
[DEBUG] Lunghezza prompt: 7019 caratteri
Bruno Rossi ha apportato un significativo contributo allo studio dei raggi cosmici,
in particolare attraverso la sua osservazione e il successivo sviluppo di una tecnica
innovativa per rilevare particelle secondarie. Conosceva che i contatori Geiger-Müller
potevano rilevare le radiazioni ionizzanti come quelli prodotti dai raggi cosmici,
ma notò un'anomalia: quando venivano visualizzate due o più particelle
contemporaneamente da lati opposti del rivelatore, suggeriva che non si trattava solo
di protoni. Rossi ha quindi ideato e costruito un contatore a coincidenze modificato
per rilevare in modo affidabile queste particelle secondarie con energie molto
elevate (PeV). Il suo lavoro è stato fondamentale perché ha evidenziato che i raggi
cosmici non sono limitati ai protoni, ma includono una vasta gamma di altre particelle.
Questa scoperta ha avuto un impatto profondo sullo studio dei raggi cosmici e sulla
nostra comprensione delle loro origini e della fisica ad alta energia coinvolte
nei processi che producono tali particelle. Il suo contatore a coincidenze,
```

```
conservato al Museo Poleni di Padova, è un prezioso artefatto stor
```

Anche in questo caso il modello incorre in varie allucinazioni. Ad esempio vengono attribuite a Bruno Rossi osservazioni di particelle nell'intervallo dei PeV, nonostante nel documento originale tali energie siano riferite ad altri esperimenti e ad altri contesti fisici.

È interessante osservare che la query contenente "Bruno Rossi" e "raggi cosmici" abbia prodotto un retrieval proveniente dal documento `complessi.pdf`, pur non contenendo esplicitamente tali espressioni.

Questo comportamento evidenzia una caratteristica fondamentale della ricerca semantica basata su embedding: il recupero dei documenti non avviene tramite semplice corrispondenza lessicale di parole chiave, ma attraverso la vicinanza geometrica tra rappresentazioni vettoriali ad alta dimensionalità generate dal modello di embedding.

In altre parole, il sistema non ricerca "parole", ma regioni semanticamente vicine nello spazio vettoriale.

11 Considerazioni finali

L'esperimento ha mostrato chiaramente che:

- PostgreSQL + pgvector costituiscono una soluzione estremamente efficace per il retrieval semantico;
- il vero collo di bottiglia computazionale è l'inferenza del modello linguistico;
- l'assenza di GPU CUDA limita fortemente le prestazioni;
- anche con hardware modesto è comunque possibile realizzare un piccolo sistema RAG funzionante.

Dal punto di vista didattico e sperimentale, il progetto si è rivelato estremamente istruttivo. Mi ha infatti permesso di osservare concretamente:

- il ruolo degli embedding;
- la differenza tra retrieval e generazione;
- il comportamento di un database vettoriale;
- le problematiche pratiche dei sistemi LLM locali.
- l'importanza di un hardware adeguato

In prospettiva futura sarebbe interessante:

- utilizzare GPU CUDA;
- introdurre streaming token-by-token;
- implementare ranking più sofisticati;
- migliorare il chunking dei PDF;
- utilizzare modelli embedding più evoluti.

Un sistema RAG non può essere considerato semplicemente come un "LLM con memoria", ma come una pipeline articolata nella quale estrazione del testo, normalizzazione documentale, indicizzazione semantica e inferenza neurale contribuiscono tutte alla qualità del risultato finale.