RAG On-Demand: Come interrogare un archivio di 300.000+ email senza fondere il server

RAG On-Demand: Come interrogare un archivio di 300.000+ email senza fondere il server

Avete presente l'incubo di dover cercare una specifica informazione, una password dimenticata, una delibera o il dettaglio di un vecchio server sperduto all'interno di un archivio storico di **oltre 300.000 mail (circa 30 GB in formato Maildir)** accumulato in anni di onorato servizio?

I client tradizionali come Outlook o Thunderbird muoiono miseramente o impiegano minuti preziosi. La soluzione moderna e "di moda" sarebbe quella di dare in pasto tutto ad un LLM (Large Language Model) locale tramite RAG (Retrieval-Augmented Generation). Ma fare il chunking e l'embedding vettoriale di 300.000 email richiede giorni di computazione CPU, database vettoriali pesanti e GPU costantemente al 100% solo per mantenere aggiornato l'indice.

Abbiamo progettato un'architettura **RAG On-Demand** ibrida e leggerissima che risolve questo problema in modo geniale: combina un'indicizzazione SQLite FTS5 locale ultra-veloce, una sincronizzazione incrementale tramite Docker e l'inferenza di Ollama via GPU remota solo quando serve, riducendo il tempo di risposta da oltre 80 a soli **11 secondi**.

Ecco come abbiamo costruito questo sistema e come potete farlo anche voi.

L'Architettura Generale del Sistema

Invece di usare costosi database vettoriali residenti in memoria (come ChromaDB o PGVector) che richiederebbero di calcolare gli embedding per 30 GB di testo ad ogni minima sincronizzazione, abbiamo optato per

un approccio ibrido basato su metadati strutturati e Full-Text Search (FTS5):

```
flowchart TD
    subgraph LocalHost ["🖥️ Host Locale (Laptop / Server)"]
        IMAP["IMAP Server\n(imap.example.com)"] -->|mbsync / ISYNC| Maildir["./maildir/\n(300.000+ file .eml)"]
        Maildir -->|Fase 1: Quick Index| SQLite["./data/emails.db\n(SQLite + FTS5 Index)"]

        UserQuery["👤 Utente: python3 ask_emails.py 'domanda'"] -->|1. Ricerca FTS5 0.01s| SQLite
        SQLite -->|2. Restituisce Top 20 Candidates| AskScript["ask_emails.py Engine"]
        Maildir -->|3. Lettura Testo Integrale via IMAP UID| AskScript
    end

    subgraph RemoteServer ["⚡ Server Remoto GPU (NVIDIA RTX 3060 12GB)"]
        AskScript -->|4. HTTP REST API (Payload Context 16k tokens)| Ollama["Ollama Server @ OLLAMA_URL"]
        Ollama -->|5. Modello: ornith:9b| Inference["GPU Inference (10-15 sec)"]
        Inference -->|6. Stream Response| UserQuery
    end
```

Componenti Principali

1. Sincronizzazione Incrementale IMAP (`mail-sync`)

Per avere le mail locali sempre aggiornate senza impatti sulla banda, utilizziamo un container Docker che fa girare `isync` (`mbsync`). Le mail vengono salvate sul disco in formato Maildir (un file `.eml` per ogni singola mail).

La particolarità di Maildir è che i file cambiano nome quando cambia lo stato di lettura (es. da `:2,RS` a `:2,S`). Per questo motivo, non possiamo mappare le mail nel database usando il nome del file fisico, ma dobbiamo usare l'**IMAP UID**, che rimane costante e unico nel tempo.

2. Demone di Indicizzazione Veloce (`mail-analyzer`)

Un demone Python gira in background ogni 5 minuti controllando i nuovi file nella directory `./maildir/`. Invece di leggere tutto il testo (che richiederebbe troppo tempo), il demone esegue un parsing ultra-veloce dei soli metadati principali:

- Mittente (`From`)
- Destinatario (`To`)
- Oggetto (`Subject`)
- Data (`Date`)
- IMAP UID (`UID`)

Il parser elabora **500 mail in soli 0.5 secondi**, scrivendo i dati in una tabella **SQLite abilitata con FTS5 (Full-Text Search 5)**.

La Magia del Motore RAG (`ask_emails.py`)

Quando l'utente fa una domanda digitando nel terminale `python3 ask_emails.py "Quali credenziali ha configurato Matteo per il server Antares nel 2025?"`, lo script esegue questi passaggi intelligenti:

1. Stemming e Normalizzazione

La query viene normalizzata per supportare la grammatica italiana e inglese (es. *incidente/incidenti* vengono ricondotti alla radice *incident*). Questo garantisce che la ricerca FTS5 trovi riscontri anche con flessioni diverse delle parole.

2. Iniezione dell'Identità Utente

Se nella domanda l'utente usa pronomi di prima persona (es. "*Cosa **devo** fare...*" o "*Cosa ho inviato **io**...*"), lo script legge la variabile d'ambiente `USER_NAME` e inietta automaticamente il nome/cognome reale dell'utente all'interno della query di ricerca su SQLite, limitando i risultati alle mail effettivamente inviate o ricevute dall'utente stesso.

3. Filtro Temporale Automatico

Se l'algoritmo rileva un anno nella query (es. "*nel 2025*"), aggiunge istantaneamente una clausola SQL restrittiva: `AND date_iso LIKE '2025%'`. Questo evita che la memoria di contesto del modello LLM venga saturata da mail irrilevanti di altri anni.

4. Risoluzione Dinamica via UID

Una volta trovate le 20 mail più rilevanti su SQLite tramite FTS5, lo script deve leggerne il testo completo per darlo in pasto all'LLM. Poiché i file fisici sul disco possono aver cambiato nome a causa del cambio di flag IMAP, lo script implementa una risoluzione dinamica tramite `glob` cercando l'UID univoco sul file system:

```
def resolve_maildir_path(uid, maildir_root):
    # Cerca il file che contiene "U=109655" (l'UID) a prescindere dalle flag finali
    pattern = os.path.join(maildir_root, "**", f"*U={uid}*")
    matches = glob.glob(pattern, recursive=True)
    return matches[0] if matches else None
```

5. Conversione HTML ed Estrazione del Testo

Molti messaggi (specialmente quelli provenienti da Outlook Exchange) sono composti esclusivamente da codice HTML pesante e ridondante. Lo script ne pulisce i tag strutturali ed estrae solo il testo nudo, riducendo del 90% il numero di token inviati alla GPU.

6. Contesto Allargato su GPU Dedicata

Le 20 mail più rilevanti così ripulite vengono formattate come contesto e inviate a un server remoto provvisto di GPU dedicata (NVIDIA RTX 3060 12GB) che esegue **Ollama**. Grazie alla VRAM generosa, possiamo utilizzare un contesto di ben **16.384 token** con il modello `ornith:9b`, ottenendo risposte ricche e dettagliate in modalità streaming direttamente sul terminale dell'utente.

Prestazioni a Confronto

Parametro	RAG Batch Continuo (Tradizionale)	RAG On-Demand (Nostro approccio)
Strategia	Embedding vettoriale h24 di tutte le 300.000 mail	Ricerca FTS5 veloce + LLM sul testo grezzo delle top 20
Tempo di Risposta	> 80 secondi (o giorni per calcolare gli indici)	**11,6 secondi**
Consumo GPU/ Banda	Continuo e costante (100% H24)	Solo durante la domanda dell'utente
Accuratezza	Media (Rischio di allucinazioni su riassunti vecchi)	**Altissima (Testo integrale delle mail reali)**

Conclusioni

Questo approccio dimostra che non serve sempre ricorrere a complessi e pesanti database vettoriali per fare RAG su grandi moli di dati. Spesso, combinare strumenti tradizionali e super ottimizzati del mondo sistemistico (come SQLite FTS5 e `mbsync`) con la flessibilità dei moderni LLM locali via Ollama permette di ottenere risultati nettamente superiori, risparmiando risorse computazionali e garantendo una privacy assoluta dei propri archivi.

Fluxer Icon

Canale di discussione e supporto

Canale di discussione e supporto. Per confrontarti su configurazioni o altro entra su fluxer.dreadful.work (https://fluxer.dreadful.work/invite/dreadful).