Readme

Sazid Ali

Overview

This project implements a **conflict-aware Retrieval-Augmented Generation (RAG)** system for NebulaGears. The goal is to answer employee policy questions accurately even when the company's documents contain contradictory guidance.

The system uses:

- **Google Gemini Flash 2.0** (primary LLM),
- **ChromaDB** (local vector store),
- **Google Embeddings v4 (text-embedding-004)**,
- **Metadata-aware reranking** to enforce policy precedence.

Dataset

Three internal policy documents are ingested:

- `employee_handbook_v1.txt` — Effective 2024-01-15
- `manager_updates_2024.txt` — Effective 2024-06-01
- `intern_onboarding_faq.txt` — Effective 2024-06-01

They contain conflicting statements such as:

- Handbook: "Remote work requires no approval."
- Manager Update: "Remote work requires manager approval and is capped at 3 days/week."
- Intern FAQ: "Interns cannot work remotely at all."

System Architecture

1. Ingestion

Each document is processed as follows:

1. Chunked with `RecursiveCharacterTextSplitter`,
2. Embedded with `text-embedding-004`,

3. Stored in ChromaDB with metadata:
    - **filename**
    - **effective_date**
    - **role_scope** (interns / all_employees)
    - **doc_type** (handbook / policy_update / role_specific)

   Metadata is extracted dynamically based on filenames.

2. Retrieval

ChromaDB returns the top vector matches based on cosine similarity.

The retrieved chunks are then **reranked** using metadata:

$$(-role\_match, -effective\_date, distance)$$

Interpretation

- **Role Match**: If the user is an intern, intern-specific chunks are boosted.
- **Recency**: Newer documents (larger date value) come before older ones.
- **Similarity**: Used only if earlier factors tie.

Note: **Role-specific documents are not penalized when the role is unknown**. They are simply treated neutrally.

Only the **top 3 reranked chunks** are passed to the LLM.

3. LLM Reasoning

Gemini receives:

- The retrieved text chunks,
- The metadata for each chunk,
- A set of conflict-resolution rules.

Conflict Rules Given to the LLM

1. Role-specific policies override general policies.
2. Newer documents override older ones (using effective_date).
3. Newer restrictions override older permissions.
4. If ambiguity remains, choose the most restrictive policy.
5. Cite the exact filenames used in the final answer.

The LLM uses metadata + rules to resolve conflicts reliably.

Conflict Resolution Logic

The system applies a layered strategy:

1. Reranking (Deterministic)

Chunks are sorted by:

1. Whether the role matches the user,
2. The recency of the document,
3. Cosine similarity from vector search.

This ensures:

- Intern queries prefer intern documents,
- Newer policies override older ones,
- Only the most relevant text is shown to the LLM.

2. LLM-Level Rules

The prompt explicitly instructs Gemini to:

- Compare dates,
- Consider role scopes,
- Resolve contradictions based on metadata,
- Cite the correct source filenames.

This removes guesswork and forces deterministic policy reasoning.

Did We Use a Prompt to Force Date/Specificity Reasoning?

Yes. The prompt explicitly instructs Gemini to analyze:

- `effective_date`
- `role_scope`
- `doc_type`

and enforce the conflict-resolution rules strictly.
Gemini is not allowed to "infer" policy; it must follow metadata.

Output for Intern Query

Cost Analysis (Approximate)

This section provides a rough, order-of-magnitude estimate of the cost of running the RAG system with Gemini Flash at scale.

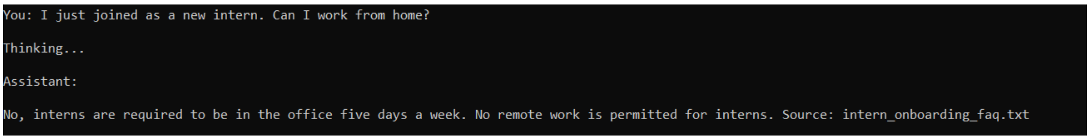

Figure 1: Output for Intern Query

Embedding Cost (One-Time)

Assume 10,000 documents, with an average of 500 tokens each. At typical embedding pricing (on the order of a few dollars per million tokens), the total embedding cost is only a few tens of cents:

$$One - timeembeddingcost \approx \$0.10-0.30.$$

LLM Query Cost (Recurring)

Each user query triggers **two** Gemini calls:

- role detection (short prompt)
- final answer generation (longer prompt)

Assuming a combined prompt size of $\sim 600-800$ input tokens and ~ 150 output tokens per query, and 5,000 queries per day, the daily cost is on the order of a few dollars:

$$Dailycost \approx \$1-2.$$

Monthly Estimate

Approx. monthly cost: $30−60.

This makes the architecture highly cost-effective for enterprise-scale RAG applications, even at 5,000 queries per day.

How to Run

1. Install Dependencies

```
pip install -r requirements.txt
```

2. Add API Key

```
GEMINI_API_KEY=your_key_here
```

3. Ingest Data

```
python src/ingest.py
```

4. Start Chat Assistant

```
python -m src.chat_loop
```

Open-Source LLM Extension (Bonus)

In addition to the Gemini Flash 2.0 pipeline, this project includes an open-source alternative powered by the **Meta Llama 3–8B Instruct** model. This satisfies the optional requirement to use an open-source model alongside (or instead of) Gemini Flash.

Model and Configuration

The open-source pipeline uses:

- **Model:** meta-llama/Meta-Llama-3-8B-Instruct
- **Framework:** HuggingFace Transformers
- **Hardware:** Google Colab T4 GPU
- **Quantization:** 4-bit NF4 quantization via BitsAndBytes

The quantization configuration is:

```
BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16
)
```

This reduces the model memory footprint from ~ 14 GB to ~ 4 GB, enabling smooth inference on free-tier GPUs.

Architecture Consistency

The open-source version uses the **same RAG architecture** as the Gemini version:

- Same document ingestion,
- Same metadata schema (effective date, role scope, document type),
- Same ChromaDB vector store,
- Same chunking strategy,
- Same hybrid retrieval,
- Same metadata-aware reranking:

$$(-role\_match, -effective\_date, distance)$$

- Same conflict-resolution rules inside the LLM prompt.

The only difference is the **LLM that interprets the retrieved context**. Gemini Flash is replaced with a local Llama 3 inference pipeline:

```
text_pipe = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    max_new_tokens=512,
    temperature=0.1,
    top_p=0.9,
    do_sample=True
)
```

Output for Intern Query

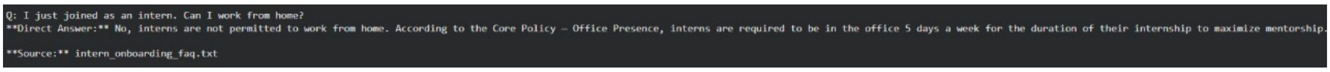

Figure 2: Output for Intern Query

Running the Open-Source Version

To run the Llama-based version:

1. Open the Colab notebook,
2. Insert your HuggingFace token,
3. Run the ingestion cell,
4. Run the hybrid RAG cell,
5. Query using:

   ```
   ask_nebula_llama_clean("your question")
   ```

This demonstrates a complete RAG pipeline running entirely with open-source language models.