

LLM-Assisted RAG in Microsoft Fabric

Workspace: Test Fabric FSKU.DACD.DDAE.DG DAIM.DSG L4FPWS

1. Introduction

Purpose

This project implements a **RAG-style chatbot in Microsoft Fabric** to answer questions using **internal DACD documentation** (Azure DevOps Wiki). It enables **semantic question answering** directly within Fabric notebooks, with answers grounded in source content and links back to documentation.

Constraints & Assumptions

- **Model constraint:** Only GPT-4o is available; no embedding models or custom deployments.
- **Environment:** Execution is in a Fabric notebook using Python (PySpark + REST calls).
- **Data scale:** Chunk volume is expected to remain in the hundreds, allowing in-memory semantic retrieval.
- **Refresh model:** Source content changes periodically; chunking and semantic enrichment are designed to be fully re-runnable with overwrite semantics.

2. High-Level Architecture

Architecture Pattern

The system uses **LLM-assisted RAG without embeddings**.

GPT-4o is used for semantic understanding and ranking instead of vector similarity search.

Rationale

This approach was chosen because:

- Embedding models are unavailable
- Dataset size is manageable
- GPT-4o can reliably handle synonym resolution and semantic ranking
- It minimizes infrastructure and operational complexity

Core Components

- **Ingestion:** Pulls documentation content and metadata
- **Chunking:** Splits content into deterministic chunks
- **Semantic Index:** GPT-generated summaries and concept labels
- **Retrieval:** GPT selects relevant chunks using semantic metadata
- **Answer Generation:** GPT produces grounded answers from selected chunks
- **Link Resolution:** Maps answers back to authoritative documentation URLs

3. Tables

3.1 dim_wiki_pages

Purpose

Stores **wiki page-level metadata** for all wiki pages.

Acts as the **source-of-truth** for mapping chunks back to documentation pages.

Populated By

- Recursive flattening of the Azure DevOps Wiki page tree

Key Columns

- PageID – Generated UUID
- Title – Derived from last segment of Path
- Path – Canonical wiki path (used as join key downstream)
- ApiUrl – API endpoint for the page
- Url – Browser-accessible wiki URL
- LastLoadedUtc – Ingestion timestamp

3.2 raw_wikipedia_contents

Purpose

Stores **raw markdown content** for each wiki page.

Population Logic

- Iterates over all rows in dim_wiki_pages
- Calls the Wiki API per page with:
 - includeContent=true
 - recursionLevel=OneLevel

Key Columns

- PageID
- Path
- Title
- Content – Raw markdown text
- RetrievedUtc

3.3 factchunks

Purpose

Stores **chunked representations** of wiki content used downstream for semantic indexing and retrieval. This is the **primary content table** used by the RAG system.

4. Chunking Strategy

The chunking process is **two-stage**:

Stage 1: Heading-Aware Sectioning

- Markdown headings (# to #####) are detected using regex

- Content is split into **logical sections**, keeping each heading with its associated text

This ensures:

- Context is preserved
- Large documents are not split arbitrarily

Stage 2: Overlapping Text Chunks

Each section is further split into overlapping chunks:

- `max_chars = 800`
- `overlap = 120`
- Whitespace normalized before splitting

This overlap:

- Preserves continuity across chunks
- Improves retrieval for boundary-spanning concepts

These chunks are stored in the table: **factchunks**

5. Semantic Indexing (LLM Summaries + Concept Labels)

Purpose

This stage builds a **semantic index** over factchunks to support **LLM-assisted retrieval** without embeddings.

For each chunk of documentation text, GPT-4o generates:

- ChunkSummary: 1–2 sentence semantic summary of the chunk
- ConceptLabels: 6–10 comma-separated technical labels (acronyms, tools, field names, system nouns)

The resulting table (factchunks_semantic) is used downstream to select relevant chunks based on meaning (synonyms/abbreviations), rather than keyword filtering.

	ChunkID	Path	Title	ChunkSummary	ConceptLabels	GeneratedUtc
1	24f1dd53-0...	/DACD D...	Request P...	The documentation describes compone...	Work item Id, devopstype, Power BI, Gateway requ...	2025-12-16 14:45:1...
2	0a56e80c-4a...	/DACD D...	Request P...	The flow utilizes a Scope component wit...	Scope component, branching logic, Power BI suppo...	2025-12-16 14:45:1...
3	4063d562-e...	/DACD D...	Request P...	The 'Other' condition in Power Automat...	Other condition, sub-conditions, Power Automate li...	2025-12-16 14:45:1...
4	09b63e2e-2...	/DACD D...	Request P...	The documentation outlines best practic...	Other condition, Scope block, Intake Form Request...	2025-12-16 14:45:1...
5	132b81b3-f...	/DACD D...	Request P...	The process involves using a Delay actio...	Delay action, Scope, parallel conditions, Power Aut...	2025-12-16 14:45:1...

How Notebook Works

1. Build the Concept Labelling Prompt

A single prompt template (`build_concept_prompt`) is used for consistent semantic enrichment across all chunks.

Prompt requirements enforced:

- Summary must be concise (1–2 sentences)
- Labels must be technical and specific
- Include exact identifiers when present (e.g., `WorkItemCreated`, `devopstype`)
- Include platform/tool nouns (e.g., Power Automate, DevOps work item, Teams)
- No invented facts
- Return strict JSON

2. Model Call Wrapper (GPT-4o via Azure OpenAI)

The notebook defines a `call_gpt4o()` function to call the deployed **gpt-4o** model through Azure OpenAI

3. Full Semantic Index Build

For the full dataset, the notebook:

Loads factchunks into pandas (current scale assumption: small enough)

Iterates row-by-row

Calls GPT-4o per chunk

Throttles requests (`sleep(0.2)`) to reduce rate-limit risk

Writes the full output once (overwrite) to `factchunks_semantic`

6. Semantic Retrieval & Answer Generation

Purpose

Retrieval

Uses GPT-4o to **semantically rank documentation chunks** based on pre-generated summaries and concept labels, allowing synonym- and acronym-aware selection without embeddings or keyword filtering.

Answer Generation

Uses GPT-4o to generate a **grounded natural-language answer** using only the selected chunk content, ensuring responses are traceable to source documentation.

Link Resolution

Maps selected content back to **documentation URLs** stored in metadata tables; URLs are never generated by the model to avoid broken or hallucinated links.

How Notebook Works

1. Load Semantic Index and Source Content

2. Semantic Understanding and Retrieval (Where “Understanding” Happens)

When a user asks a question, GPT-4o first performs semantic interpretation of the question.

This happens during retrieval.

GPT-4o:

- Reads the question
- Compares it against:
 - Chunk summaries
 - Concept labels
- Resolves:
 - Synonyms (e.g., *RLS* \leftrightarrow *Row Level Security*)
 - Acronyms
 - Conceptual similarity even when wording differs

Based on this, GPT-4o selects the most relevant ChunkIDs.

3. Retrieve Full Chunk Text for Grounding

After relevant chunks are identified:

- The system retrieves the full ChunkText for the selected ChunkIDs
- This text becomes the authoritative grounding context for answer generation

This design ensures:

- Retrieval is based on meaning
- Answers are based on exact documentation content

4. Deduplicate Sources by Documentation Page

Multiple chunks may originate from the same documentation page.

The notebook deduplicates these sources to:

- Avoid redundant links
- Present a clean “read more” experience
- Ensure one documentation page appears only once, even if multiple chunks were relevant

This step affects output presentation only, not answer correctness.

5. Answer Generation

GPT-4o is then asked to generate an answer using:

- Only the selected chunk text
- No external knowledge
- No citations or links

The model:

- Synthesizes information across chunks if needed

- Produces a clear, human-readable response
- Does not generate URLs or references
- Consistent with source documentation

6. Source Selection for “Read More”

Instead of emitting links, GPT-4o:

- Selects which of the provided sources are useful for further reading
- Returns indexes pointing to those sources

This keeps the model focused on content relevance, not URL construction.

7. URL Resolution

Finally, the notebook:

- Maps selected source indexes back to the wiki’s URLs stored in dim_wiki_pages
- Returns raw URLs only (no formatting)

8. Final Output

The notebook returns:

- A grounded answer string
- A deduplicated list of documentation URLs for further reading

```

1 ask("what are the Cyber Security Trainings provided")
[6] ✓ 22 sec - Command executed in 22 sec 817 ms by Veera K@ADM(DS) DDAE@Defence365 on 11/17/25 PySpark (Python) ▾
> └─ Spark jobs (6 of 6 succeeded) └─ Resources ...
{'question': 'what are the Cyber Security Trainings provided',
 'answer': 'The cyber security trainings provided include platform training, security training, and O&D processes training. The Canadian Centre for Cyber Security (CCCS) offers technical reference documentation and training, including cloud security courses and software development security courses.',
 'read_more_urls': ['https://dev.azure.com/DIA-DAI/36bef981-1b6c-4cc8-8459-d100334fb285/_wiki/wikis/c711e9f6-2d84-4b88-9dab-ca4ebac8a851?pagePath=%2FDACD%20Documentation%2FGeneral%20Platform%20Administration%2FCyber%20Security%20Training']}

```