Whitepaper #03: Private Knowledge Retrieval

Subtitle: *Architecting Local RAG Systems for Sensitive Data* **Author:** Dustin J. Ober, PMP

1. Executive Summary

The Bottom Line Up Front (BLUF): The most powerful application of Generative AI is not "creative writing," but **Retrieval-Augmented Generation (RAG)**—the ability to ground LLM responses in an organization's internal data. However, for sensitive sectors (Defense, Legal, Healthcare), the standard RAG playbook (uploading documents to a cloud vector store and querying OpenAI) is a non-starter due to data sovereignty risks.

The Solution: This brief outlines the architecture for a **Private RAG Pipeline**. This system runs entirely within a Closed System (air-gapped or on-premise), performing document ingestion, vector embedding, and semantic retrieval without a single byte crossing the network boundary.

The Outcome: By deploying local embedding models and self-hosted vector databases (e.g., Chroma, Qdrant), organizations can enable "ChatGPT-like" interrogation of classified manuals, proprietary research, and sensitive emails while maintaining absolute Zero Trust compliance.

2. The Operational Challenge: Data Gravity

In the commercial sector, RAG is often trivialized: "Just use the OpenAI Assistants API." This abstracts away the complexity of parsing, chunking, and indexing. In a sovereign environment, you must own the entire ETL (Extract, Transform, Load) pipeline.

2.1 The "Upload" Prohibition

The fundamental constraint is that internal documents—whether they are CUI (Controlled Unclassified Information), PII (Personal Identifiable Information), or IP—cannot be uploaded to an external embedding provider.

- **The Gap:** You cannot use `text-embedding-3-small` (OpenAI) or Pinecone (Cloud Vector DB).
- **The Fix:** You must run the embedding model on your own CPU/GPU and store the vectors on your own disk.

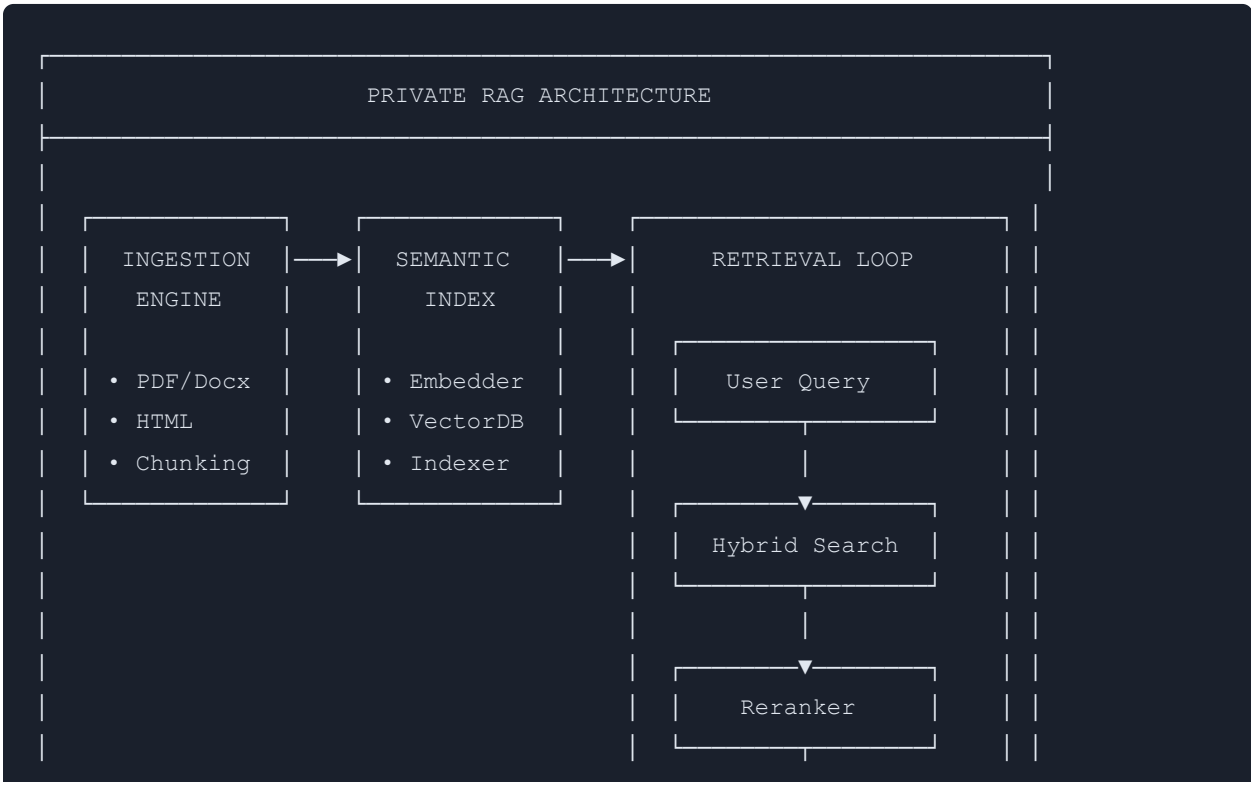
2.2 The "Black Box" Risk

When using cloud RAG, you trust the provider to retrieve the right context. In high-stakes environments, "trust" is insufficient. You need **explainability**. If the model claims "System X is safe," you need to know exactly which paragraph of the technical manual it cited, and you need to verify that the retrieval algorithm didn't miss a critical warning on the next page.

---

3. The Architecture: The Private RAG Triad

A Private RAG system consists of three distinct subsystems that replace their cloud counterparts.



3.1 The Ingestion Engine (ETL)

- **Role:** Converts raw files (PDF, Docx, HTML) into clean text.
- **Tooling: Unstructured** (local library) or **PyMuPDF**.
- **Constraint:** Must handle messy formatting (headers, footers, tables) without calling cloud OCR APIs.

3.2 The Semantic Index (The Brain)

- **Role:** Converts text into mathematical vectors (embeddings) and stores them.
- **Tooling: Sentence-Transformers** (Model) + **ChromaDB/Qdrant** (Database).
- **Constraint:** Must run efficiently on limited hardware (often CPU-only for indexing).

3.3 The Retrieval Loop (The Interface)

- **Role:** Finds the most relevant chunks and feeds them to the Local LLM (from Whitepaper #01).
- **Strategy: Hybrid Search** (combining Keyword search with Semantic search) for maximum accuracy in technical domains.

---

4. Implementation Strategy: Building the Engine

This section outlines the specific software stack required to build this capability in a disconnected environment.

4.1 Local Embeddings: The "Math" of Meaning

You need a model that turns text into numbers. Since you cannot call an API, you must host a small BERT-based model locally.

Model	Size	Speed	Quality	Best For
`all-MiniLM-L6-v2`	90MB	Fast	Good	General purpose, CPU
`bge-m3`	2.3GB	Medium	Excellent	Multi-lingual, high accuracy
`nomic-embed-text`	274MB	Fast	Very Good	Long context (8K tokens)
`e5-large-v2`	1.3GB	Slow	Excellent	Maximum retrieval quality

Recommendation: Start with `all-MiniLM-L6-v2` for prototyping, upgrade to `bge-m3` for production.

Deployment: Download the model weights on the Open Internet, transfer via Sneakernet (see Whitepaper #02), and load via Hugging Face `local_files_only=True`.

```
from sentence_transformers import SentenceTransformer

# Load model from local path inside the Closed System
model = SentenceTransformer('/mnt/models/all-MiniLM-L6-v2')
embeddings = model.encode(docs)
```

4.2 The Vector Database: File vs. Server

Where do you store the math?

Feature	ChromaDB	Qdrant	Milvus
Deployment	Embedded/File	Docker/Server	Docker/Cluster
Best For	Prototyping, Desktop	Teams, Production	Enterprise, Scale
Language	Python	Rust	Go
ACL Support	Manual	Native	Native

Feature	ChromaDB	Qdrant	Milvus
Filtering	Basic	Advanced	Advanced

Option A: ChromaDB (File-Based)

- *Best For:* Individual analysts, desktop apps, prototyping.
- *Why:* It runs "embedded" inside your Python script. It creates a `chroma.sqlite3` file on your disk. Zero infrastructure setup.

```
import chromadb

# Create persistent local database
client = chromadb.PersistentClient(path="/data/vectordb")
collection = client.get_or_create_collection("documents")

# Add documents with metadata
collection.add(
    documents=["Document text here..."],
    metadatas=[{"source": "manual_v1.pdf", "page": 42}],
    ids=["doc1"]
)
```

Option B: Qdrant (Server-Based)

- *Best For:* Enterprise Knowledge Bases, multi-user teams.
- *Why:* It runs as a Docker container. It is written in Rust, extremely fast, and supports "Role-Based Access Control" (filtering results so users only see documents they are allowed to see).

```
from qdrant_client import QdrantClient
from qdrant_client.models import PointStruct, Filter, FieldCondition, MatchValue

client = QdrantClient(host="localhost", port=6333)

# Search with access control filter
results = client.search(
    collection_name="documents",
    query_vector=query_embedding,
```

```
    query_filter=Filter(
        must=[FieldCondition(key="access_group",
match=MatchValue(value="engineering"))]
    ),
    limit=10
)
```

4.3 The "Reranker" Step (Critical for Accuracy)

In technical domains, standard similarity search often fails (e.g., confusing "Project A requirements" with "Project A budget").

- **The Fix:** Implement a **Cross-Encoder Reranker**.
- **Workflow:**
  1. The Vector DB retrieves the top 20 possible matches (fast but loose).
  2. The Reranker (a specialized local model) reads those 20 pairs and scores them carefully.
  3. The top 5 *actual* matches are sent to the LLM.
- **Impact:** This increases retrieval accuracy ("Recall") by 15–20% with minimal latency.

```
from sentence_transformers import CrossEncoder

reranker = CrossEncoder('/mnt/models/ms-marco-MiniLM-L-6-v2')

# Rerank candidates
pairs = [(query, doc.text) for doc in candidates]
scores = reranker.predict(pairs)

# Sort by score and take top 5
reranked = sorted(zip(candidates, scores), key=lambda x: x[1], reverse=True)[:5]
```

---

5. Security & Governance: Citations & ACLs

In a Closed System, security does not stop at the network border. You must enforce security *inside* the application.

5.1 Citation-Backed Generation

Hallucinations are the enemy. The system must be engineered to prove its work.

- **The Prompt Strategy:** Do not ask *"What is the policy?"* Ask *"Answer the question using ONLY the provided context. If the answer is not in the context, state 'I do not know'."*

Example System Prompt:

```
You are a technical assistant. Answer questions using ONLY the provided context.
For each claim you make, include a citation in the format [Source: filename, Page
X].
If the answer is not found in the context, respond: "I cannot find this
information in the provided documents."
Do not make up information.
```

- **The UI Requirement:** The interface must display the specific PDF page number and filename for every claim the model makes.

5.2 Document-Level Access Control (ACLs)

Not every user should see every document.

- **The Danger:** If a user asks "What are the CEO's salary details?", and that HR document is in the Vector DB, a standard RAG system will retrieve it and answer.
- **The Solution:** Tag every vector with metadata permissions (e.g., `group: ["hr", "admin"]`). When a user queries, the system must filter the search *before* the retrieval happens. Qdrant supports this natively.

```
# When indexing: tag with access groups
collection.add(
    documents=[doc_text],
    metadatas=[{
        "source": "hr_salaries.xlsx",
        "access_groups": ["hr", "executive"]
    }],
    ids=["hr_001"]
)

# When querying: filter by user's groups
```

```
user_groups = get_user_groups(current_user)
results = collection.query(
    query_embeddings=[query_embedding],
    where={"access_groups": {"$in": user_groups}},
    n_results=10
)
```

---

6. RAG Pipeline Checklist

Before deploying your Private RAG system, verify:

- ☐ **Embedding Model:** Downloaded and tested offline
- ☐ **Vector Database:** Deployed and accessible (Chroma/Qdrant)
- ☐ **Document Ingestion:** PDF/Docx parsing working without cloud dependencies
- ☐ **Chunking Strategy:** Configured (e.g., 512 tokens with 50 token overlap)
- ☐ **Reranker:** Deployed for high-accuracy retrieval
- ☐ **Access Controls:** Metadata tagging and filtering implemented
- ☐ **Citation Prompts:** System prompt enforces source citations
- ☐ **Local LLM:** Integrated with retrieval output (from Whitepaper #01)
- ☐ **UI:** Displays sources and page numbers with responses

---

7. Conclusion

Private Knowledge Retrieval is the bridge between "Raw Data" and "Actionable Intelligence." By architecting a RAG pipeline that relies on **Local Embeddings**, **Self-Hosted Vector Stores**, and **Verifiable Citations**, organizations can unlock the value of their internal archives without compromising data sovereignty.

This capability transforms the Local LLM from a generic chatbot into a subject-matter expert on *your* mission, *your* history, and *your* procedures—all without a single packet leaving the secure boundary.

Key Takeaways:

1. **Own the Stack:** Run embeddings locally; never upload documents externally.
2. **Choose the Right DB:** ChromaDB for prototypes, Qdrant for production.
3. **Rerank for Accuracy:** A Cross-Encoder dramatically improves retrieval quality.
4. **Enforce ACLs:** Filter queries by user permissions before retrieval.
5. **Cite Everything:** Design the system to prove every claim with a source.

Next in this Series:

- **Whitepaper #04:** *Verifiable Intelligence: DSPy, Governance, and Hallucination Control.*

About the Author

Dustin J. Ober, PMP, M.Ed. *AI Developer & Technical Instructional Systems Designer*

Dustin J. Ober is a specialist in the intersection of Artificial Intelligence, Instructional Strategy, and secure systems architecture. With a background spanning over two decades in the United States Air Force and defense contracting, he focuses on deploying high-impact technical solutions within mission-critical environments.

Unlike traditional developers who focus solely on code, Dustin bridges the gap between **technical capability** and **operational reality**. His expertise lies in architecting "Sovereign AI" systems—designing offline, air-gapped inference pipelines that allow organizations to leverage state-of-the-art intelligence without compromising data security or compliance.

He holds a Master of Education in Instructional Design & Technology and is a certified Project Management Professional (PMP). He actively develops open-source tools for the AI community, focusing on DSPy implementation, neuro-symbolic logic, and verifiable agentic workflows.

Connect:

- **Web:** aiober.com
- **LinkedIn:** linkedin.com/in/dustinober
- **Email:** dustinober@me.com

Suggested Citation: Ober, D. J. (2025). *Private Knowledge Retrieval: Architecting Local RAG Systems for Sensitive Data* (Whitepaper No. 03). AIOber Technical Insights.