Agentic RAG System - System Design Document

Table of Contents

---

1. Executive Summary

This document describes the design of an **Agentic RAG (Retrieval-Augmented Generation) System** built for manufacturing document Q&A. The system intelligently processes user queries by routing them through specialized agents that handle intent classification, document retrieval, and response generation with source citations.

Key Features

Feature	Description
Agentic Routing	LLM-based intent classification with 4 query types
Multi-Format Ingestion	PDF, DOCX, Excel, PPTX, TXT support
Vector Storage	Milvus Lite with IVF_FLAT indexing
Citation System	Page-level source references
Dual LLM Support	Gemini (cloud) + Ollama (on-premise)

---

2. System Architecture

2.1 High-Level Architecture

```
┌─────────────────────────────────────────────────────────────────────┐
│                         PRESENTATION LAYER                          │
│  ┌────────────────────────────┐   ┌────────────────────────────────┐│
│  │       Streamlit UI         │   │        FastAPI REST            ││
│  │  app.py                    │   │  api.py                        ││
│  │  • Chat interface          │   │  • POST /query                 ││
│  │  • File upload             │   │  • POST /ingest                ││
│  │  • Source display          │   │  • GET /collection/stats       ││
│  └────────────────────────────┘   └────────────────────────────────┘│
└─────────────────────────────────────────────────────────────────────┘
                                   │
```

```
                                ▼
┌──────────────────────────────────────────────────────────────┐
│                        AGENTIC LAYER                          │
│                                                               │
│  ┌─────────────────────────────────────────────────────────┐  │
│  │                      ORCHESTRATOR                        │  │
│  │             agents/orchestrator.py                       │  │
│  │             process_query() - main pipeline              │  │
│  └─────────────────────────────────────────────────────────┘  │
│                               │                               │
│           ┌───────────────────┼───────────────────┐           │
│           ▼                   ▼                   ▼           │
│  ┌─────────────┐      ┌─────────────┐      ┌─────────────┐    │
│  │   ROUTER    │      │  RETRIEVER  │      │  GENERATOR  │    │
│  │ router.py   │      │ retriever.py│      │ generator.py│    │
│  │             │      │             │      │             │    │
│  │ • classify_ │      │ • retrieve_ │      │ • generate_ │    │
│  │   intent()  │      │   documents │      │   response  │    │
│  │ • route_    │      │ • build_    │      │ • format_   │    │
│  │   query()   │      │   context   │      │   citations │    │
│  └─────────────┘      └─────────────┘      └─────────────┘    │
│                                                               │
└──────────────────────────────────────────────────────────────┘
                                │
                                ▼
┌──────────────────────────────────────────────────────────────┐
│                         DATA LAYER                            │
│                                                               │
│  ┌───────────────────┐    ┌─────────────────────────────────┐ │
│  │    MILVUS LITE    │    │          LLM PROVIDERS          │ │
│  │ vectordb/         │    │                                 │ │
│  │ milvus_client.py  │    │  ┌──────────┐   ┌────────────┐  │ │
│  │                   │    │  │  Gemini  │   │   Ollama   │  │ │
│  │ • IVF_FLAT index  │    │  │ google.  │   │  (Local)   │  │ │
│  │ • COSINE metric   │    │  │ genai    │   │  qwen3:8b  │  │ │
│  │ • 384 dimensions  │    │  └──────────┘   └────────────┘  │ │
│  │ • ./milvus_data.db│    │                                 │ │
│  └───────────────────┘    └─────────────────────────────────┘ │
└──────────────────────────────────────────────────────────────┘
```

2.2 Project Structure

```
agentic-rag/
├── app.py                  # Streamlit UI
├── api.py                  # FastAPI REST backend
├── config.py               # Environment configuration
├── logger.py               # Colored console logging
│
├── agents/
│   ├── router.py           # Intent classification (4 types)
│   ├── retriever.py        # Vector search + context building
│   ├── generator.py        # LLM response generation
│   └── orchestrator.py     # Pipeline coordination
│
├── ingestion/
│   ├── loader.py           # Multi-format document extraction
│   └── chunker.py          # Smart text splitting
│
├── vectordb/
│   └── milvus_client.py    # Milvus Lite operations
│
└── data/samples/           # Sample manufacturing documents
```

2.3 Data Flow

```
Document Upload                    User Query
      │                                 │
      ▼                                 ▼
┌─────────────┐                  ┌─────────────┐
│   LOADER    │                  │   ROUTER    │
│ PDF/DOCX/   │                  │ Classify    │
│ Excel/PPTX  │                  │ Intent      │
└─────────────┘                  └─────────────┘
       │                                │
       ▼                                ▼
┌─────────────┐                  ┌─────────────┐
│   CHUNKER   │                  │ RETRIEVER   │
│ 512 chars   │                  │ Vector      │
│ 50 overlap  │                  │ Search      │
└─────────────┘                  └─────────────┘
       │                                │
       ▼                                ▼
┌─────────────┐                  ┌─────────────┐
│   EMBEDDER  │                  │  CONTEXT    │
│ all-MiniLM  │                  │  BUILDER    │
│ L6-v2       │                  │             │
└─────────────┘                  └─────────────┘
       │                                │
       ▼                                ▼
┌─────────────┐                  ┌─────────────┐
│  MILVUS     │◄─────────────────│ GENERATOR   │
│  LITE       │                  │ LLM + Cites │
└─────────────┘                  └─────────────┘
```

---

3. Agentic Workflow Design

3.1 What Makes This "Agentic"?

Unlike traditional RAG systems that follow a fixed pipeline, this system makes **intelligent decisions** at multiple points:

Traditional RAG	This Agentic RAG
Query → Retrieve → Generate	Query → **Route** → Retrieve → Generate
Fixed pipeline	**Decides** action based on intent
No self-correction	**Refines** if results are poor

3.2 Router Agent - Intent Classification

The Router Agent classifies queries into **4 distinct intents**:

```python
class QueryIntent(str, Enum):
    RETRIEVAL = "retrieval"     # Needs document search (DEFAULT)
    DIRECT = "direct"           # Greetings, meta-questions
    MULTI_PART = "multi_part"   # Complex queries → decompose
    CLARIFY = "clarify"         # Vague queries → ask for clarity
```

Classification Examples:

Query	Intent	Behavior
"What is the LOTO procedure?"	RETRIEVAL	Search documents
"Hello, how can you help?"	DIRECT	Respond without search
"What is LOTO and what PPE needed?"	MULTI_PART	Decompose → 2 searches
"Help me with the thing"	CLARIFY	Ask for specifics

3.3 Agentic Decision Flow

```
User Query: "What is LOTO and what PPE is required?"
         |
         ▼
┌──────────────────────────────────────────┐
│           ROUTER AGENT                   │
│                                          │
│  LLM classifies → MULTI_PART             │
│  Decompose into:                         │
│    • "What is LOTO procedure?"           │
│    • "What PPE is required?"             │
└──────────────────────────────────────────┘
                     |
                     ▼
┌──────────────────────────────────────────┐
│           RETRIEVER AGENT                │
│                                          │
│  Search Query 1 → 5 results              │
│  Search Query 2 → 5 results              │
│  Deduplicate → Merged results            │
└──────────────────────────────────────────┘
                     |
                     ▼
┌──────────────────────────────────────────┐
│      AGENTIC SELF-CORRECTION             │
│                                          │
│  IF >50% results are template files:     │
│    → Refine query                        │
│    → Search again                        │
│    → Merge new results                   │
└──────────────────────────────────────────┘
                     |
                     ▼
┌──────────────────────────────────────────┐
│           GENERATOR AGENT                │
│                                          │
│  Build context with [Source N] refs      │
│  Generate answer using LLM               │
│  Append formatted citations              │
└──────────────────────────────────────────┘
```

3.4 Self-Correction Mechanism

The orchestrator evaluates retrieval results and takes corrective action:

```python
# Agentic refinement: if >50% templates, refine search
template_count = sum(1 for r in results if 'template' in r.source_file.lower())
if template_count >= len(results) * 0.5:
```

```python
    refined_query = f"{query} in database specifications inventory"
    refined_results = retrieve_documents(refined_query)
    # Merge and deduplicate
```

This is "agentic" because the system: 1. **Evaluates** initial results 2. **Decides** they are not good enough 3. **Takes corrective action** autonomously

---

4. Context Construction Strategy

4.1 Document Processing Pipeline

```
┌─────────────┐    ┌─────────────┐    ┌─────────────┐    ┌─────────────┐
│   LOAD      │ →  │   CLEAN     │ →  │   CHUNK     │ →  │   EMBED     │
│             │    │             │    │             │    │             │
│ PDF/DOCX/   │    │ Remove      │    │ 512 chars   │    │ all-MiniLM  │
│ Excel/PPTX  │    │ noise       │    │ 50 overlap  │    │ L6-v2       │
└─────────────┘    └─────────────┘    └─────────────┘    └─────────────┘
```

4.2 Metadata Preservation

Each chunk preserves rich metadata for accurate citations:

```python
@dataclass
class TextChunk:
    content: str           # Actual text
    source_file: str       # Filename for citation
    file_type: str         # pdf, docx, excel, pptx, txt
    page_number: int       # Page/slide number
    section: str           # Section heading
    chunk_index: int       # Position in document
    content_type: str      # text, table, heading
```

4.3 Content-Type Aware Chunking

Content Type	Strategy	Rationale
Text	512 chars, sentence boundaries	Semantic coherence
Tables	Keep header in each chunk	Context for data rows
Headings	Don't split	Atomic section markers
Excel rows	Convert to natural language	Better semantic search

Excel Row Conversion Example:

```
Before (table cell): | Pump A | 150 PSI | Active |

After (semantic text):
"From equipment_data.xlsx, sheet 'Pumps':
 - Equipment ID: Pump A
 - Pressure: 150 PSI
 - Status: Active"
```

4.4 Context Building for LLM

```
def build_context(results, max_length=4000):
    context_parts = []
    citations = []

    for i, result in enumerate(results, 1):
        # Create source reference
        source_ref = f"[Source {i}: {result.source_file}"
        if result.page_number:
            source_ref += f", Page {result.page_number}"
        source_ref += "]"

        entry = f"{source_ref}\n{result.content}\n"
        context_parts.append(entry)

        # Track citation
        citations.append(Citation(
            source_file=result.source_file,
            page_number=result.page_number,
            section=result.section
        ))

    return "\n".join(context_parts), citations
```

---

5. Technology Choices & Rationale

5.1 Vector Database: Milvus Lite

Aspect	Choice	Rationale
Database	Milvus Lite	On-premise capable, no Docker required
Index	IVF_FLAT	Good balance of speed/recall for dataset size
Metric	COSINE	Standard for semantic similarity
Dimensions	384	Matches all-MiniLM-L6-v2 output

Why Milvus Lite over alternatives?

Option	Pros	Cons	Decision
Milvus Lite ✓	On-premise, same API as production	Limited scale	Best for demo + bonus points
ChromaDB	Simple	Limited features	-
Pinecone	Managed	Cloud-only, cost	-
Docker Milvus	Full features	Setup complexity	-

5.2 Embedding Model: all-MiniLM-L6-v2

Aspect	Value
Dimensions	384
Model Size	~80MB

Aspect	Value
Runs Locally	✅ Yes
Language	English optimized

Why this model? - Runs completely offline (air-gap compatible) - Fast inference on CPU - Good semantic quality for manufacturing domain - Widely validated performance

5.3 LLM Integration: Dual Provider Support

```
def get_llm_client():
    if config.llm_provider == "ollama":
        from ollama import Client
        return Client(), "ollama"
    else:
        from google import genai
        client = genai.Client(api_key=config.google_api_key)
        return client, "genai"
```

Provider	Use Case	Benefits
Gemini	Development, demos	Free tier, fast, reliable
Ollama	On-premise, air-gapped	Data stays local, no internet

5.4 Document Processing Libraries

Format	Library	Features Used
PDF	PyMuPDF (fitz)	Text extraction, table detection
DOCX	python-docx	Heading hierarchy, tables
Excel	pandas + openpyxl	Auto header detection, row semantics
PPTX	python-pptx	Slide content, speaker notes
TXT	Built-in	Basic text loading

---

6. Key Design Decisions

6.1 RETRIEVAL as Default Intent

Decision: When the router cannot confidently classify a query, it defaults to RETRIEVAL.

Rationale: - Manufacturing context: 90%+ queries need document search - Miss cost > Extra search cost - A failed search with "no results" is better than no response

```
except (json.JSONDecodeError, KeyError, ValueError):
    return RoutingResult(
        intent=QueryIntent.RETRIEVAL,
        sub_queries=[],
        reasoning="Defaulting to retrieval mode"
    )
```

6.2 Template File Penalty

Decision: Apply 30% score penalty to files with "template" in name.

Rationale: - Template files have headers that match queries but lack actual data - Pushes real data files higher in results

```
if 'template' in result.source_file.lower():
    result.score *= 0.7  # 30% penalty
```

6.3 Synchronous Architecture

Decision: All agents use synchronous function calls.

Rationale: - Simpler debugging and testing - Sufficient for demo scale - Easy to migrate to async later if needed

6.4 Milvus Schema Design

Decision: 7 fields with rich metadata.

```
schema.add_field("id", DataType.INT64, is_primary=True, auto_id=True)
schema.add_field("vector", DataType.FLOAT_VECTOR, dim=384)
schema.add_field("content", DataType.VARCHAR, max_length=65535)
schema.add_field("source_file", DataType.VARCHAR, max_length=512)
schema.add_field("file_type", DataType.VARCHAR, max_length=32)
schema.add_field("page_number", DataType.INT32)
schema.add_field("section", DataType.VARCHAR, max_length=512)
schema.add_field("chunk_index", DataType.INT32)
```

Rationale: - `source_file` + `page_number`: Essential for citations - `file_type`: Enable filtering by document type - `section`: Context for DOCX headings - `chunk_index`: Reconstruct document order if needed

---

7. Limitations & Future Work

7.1 Current Limitations

Limitation	Impact	Mitigation
No streaming	Full response wait	Acceptable for demo
Synchronous	Sequential processing	Sufficient for scale
No caching	Repeated embeddings	Low query volume
English only	Single language	Manufacturing domain focus
No hybrid search	Exact matches missed	Semantic search covers most cases
No MCP integration	No external tools	Self-contained by design

7.2 Design Choice: Self-Contained Architecture

Why No External Services or Tools?

This system is intentionally designed to be **self-contained** without external service dependencies:

Aspect	Our Approach	Rationale
MCP Servers	Not implemented	Focus on core RAG capabilities first
Web search	Not included	Manufacturing data is internal
External APIs	None required	Air-gap/on-premise deployment
Third-party tools	Avoided	Simpler deployment, no API costs

Benefits of Self-Contained Design:

1. **On-Premise Ready**: Can run entirely offline in secure environments
2. **No External Dependencies**: No API keys, no rate limits, no costs beyond LLM
3. **Predictable Behavior**: All data flows are internal and auditable
4. **Security**: No data leaves the system to external services
5. **Simpler Deployment**: Just pip install + .env configuration

Trade-offs Accepted:

- Cannot augment answers with real-time web data
- No automatic tool use (calculator, code execution)
- MCP integration would require additional infrastructure

7.3 Scalability Considerations

Current	At Scale
Milvus Lite (file-based)	Milvus Standalone/Cluster
Synchronous agents	Async with asyncio
No caching	Redis for embeddings
Single instance	Load-balanced API

7.4 Future Enhancements

1. **Hybrid Search**: Add BM25 for part numbers, equipment IDs
2. **Cross-Encoder Re-ranking**: Improve retrieval precision
3. **Streaming Responses**: Better UX for long answers
4. **Multi-language**: Support for regional manufacturing docs
5. **Evaluation Pipeline**: RAGAS metrics for continuous improvement
6. **MCP Servers**: Model Context Protocol for tool integration
7. **Conversation Memory**: Multi-turn context retention

Appendix A: Configuration

Environment Variables

```
# LLM Configuration
LLM_PROVIDER=gemini         # gemini or ollama
GOOGLE_API_KEY=your_key     # For Gemini
GEMINI_MODEL=gemini-3-flash-preview
```

```
OLLAMA_MODEL=qwen3:8b      # For Ollama

# Milvus Configuration
COLLECTION_NAME=manufacturing_docs
EMBEDDING_MODEL=all-MiniLM-L6-v2
EMBEDDING_DIM=384

# Chunking Configuration
CHUNK_SIZE=512
CHUNK_OVERLAP=50
```

Running the System

```
# Install dependencies
pip install -r requirements.txt

# Configure
cp .env.example .env
# Edit .env with your GOOGLE_API_KEY

# Run Streamlit UI
streamlit run app.py

# OR Run FastAPI
uvicorn api:app --reload
```

Appendix B: API Reference

POST /query

```
{
  "query": "What is the LOTO procedure?",
  "top_k": 5
}
```

Response:

```
{
  "query": "What is the LOTO procedure?",
  "intent": "retrieval",
  "response": "According to [Source 1]...",
  "citations": [
    {"source_file": "safety_manual.pdf", "page_number": 3}
  ]
}
```

POST /ingest

```
{
  "file_path": "/path/to/document.pdf"
}
```
