Exercise 1: Intelligent Document Q&A System with Memory

Duration: 2-3 hours
Difficulty: Medium
Focus: RAG Architecture with Long-term Memory using Google Gemini

Objective

Build a production-ready document question-answering system that learns from user interactions and improves over time. The system should handle multiple document formats, maintain conversation history, and adapt based on user feedback.

Datasets to Use

1. **Stanford Question Answering Dataset (SQuAD 2.0)**
   - Source: https://rajpurkar.github.io/SQuAD-explorer/
   - 150,000+ questions on 500+ Wikipedia articles
   - Includes unanswerable questions for robustness testing
2. **CoQA - Conversational Question Answering**
   - Source: https://stanfordnlp.github.io/coqa/
   - 127,000+ questions with answers from 8,000+ conversations
   - Multi-turn conversational structure
3. **Natural Questions (NQ) Dataset**
   - Source: https://ai.google.com/research/NaturalQuestions/download
   - Real Google search queries with Wikipedia answers
   - Long-form and short-form answer annotations

System Architecture Requirements

Phase 1: Document Processing Pipeline (30-40 minutes)

1. **Multi-format Document Ingestion**
   - Support PDF, DOCX, TXT, HTML, and Markdown files
   - Implement intelligent chunking strategies:
     - Semantic chunking based on paragraph boundaries
     - Sliding window with overlap for context preservation
     - Dynamic chunk sizing based on content type
2. **Embedding Generation**
   - Use Gemini's embedding model for text chunks

- Implement hierarchical embeddings:
  - Document-level embeddings
  - Section-level embeddings
  - Chunk-level embeddings
- Store metadata: source, page numbers, timestamps

3. **Vector Database Setup**
   - Choose between ChromaDB, Pinecone, or Weaviate
   - Design schema for:
     - Document chunks with embeddings
     - User interaction history
     - Feedback data
     - Query-answer pairs

Phase 2: Q&A Engine with Memory (40-50 minutes)

1. **Context-Aware Query Processing**
   - Implement query expansion using previous conversations
   - Use Gemini Pro to reformulate queries based on context
   - Maintain conversation threads with session management
2. **Retrieval-Augmented Generation (RAG)**
   - Hybrid search: semantic + keyword matching
   - Re-ranking based on:
     - Relevance scores
     - User interaction history
     - Temporal relevance
   - Dynamic context window adjustment
3. **Memory Implementation**
   - **Short-term Memory**: Current session context (last 20 exchanges)
   - **Long-term Memory**:
     - Successful Q&A pairs indexed by topic
     - User-specific preferences and patterns
     - Document-specific learnings
   - **Episodic Memory**: Complete interaction histories with outcomes

Phase 3: Learning and Adaptation System (30-40 minutes)

1. **Feedback Collection Mechanism**
    - Explicit feedback: thumbs up/down, corrections
    - Implicit feedback: dwell time, follow-up questions
    - Answer quality ratings (1-5 scale)
2. **Learning Pipeline**
    - Store corrected answers with original queries
    - Build correction patterns database
    - Fine-tune retrieval based on feedback
    - Implement A/B testing for answer strategies
3. **Performance Optimization**
    - Cache frequently asked questions
    - Pre-compute embeddings for common query patterns
    - Implement query result caching with TTL

Phase 4: Evaluation and Testing (30-40 minutes)

1. **Automated Testing Suite**
    - Use SQuAD evaluation metrics (F1, Exact Match)
    - Test on CoQA for conversational coherence
    - Measure retrieval precision/recall
2. **User Experience Metrics**
    - Response time benchmarking
    - Memory usage profiling
    - Conversation coherence scoring
3. **Production Readiness**
    - API documentation
    - Error handling and logging
    - Rate limiting implementation

Deliverables

1. **Working System Components**
    - Document upload and processing API
    - Q&A interface with conversation history
    - Admin dashboard showing learning metrics
    - Memory visualization interface

2. **Demonstration Requirements**
   - Process 10+ diverse documents
   - Handle multi-turn conversations
   - Show learning from corrections
   - Display performance improvements over time
3. **Technical Documentation**
   - System architecture diagram
   - API endpoint documentation
   - Memory schema design
   - Performance benchmarks

Evaluation Criteria

- **Accuracy**: F1 score > 0.7 on test queries
- **Learning**: Demonstrable improvement from feedback
- **Memory**: Effective context retention across sessions
- **Performance**: < 2 second response time
- **Scalability**: Handle 100+ documents efficiently
- **Code Quality**: Modular, well-documented code

Technical Stack Recommendations

- **LLM**: Google Gemini Pro
- **Embeddings**: Gemini Embedding Model
- **Vector DB**: ChromaDB or Pinecone
- **Backend**: FastAPI or Flask
- **Frontend**: Streamlit or React
- **Monitoring**: Weights & Biases or MLflow

Advanced Features (Bonus)

- Multi-language support
- Source attribution with confidence scores
- Query suggestion based on document content
- Automatic document summarization
- Cross-document knowledge synthesis
- User role-based access control

Common Pitfalls to Avoid

- Over-chunking documents losing context
- Not handling embedding failures gracefully
- Ignoring conversation context in follow-ups
- Poor feedback loop implementation
- Inadequate error handling for Gemini API limits

Resources

- Gemini API Documentation
- LangChain RAG tutorials
- Vector database best practices
- Evaluation metrics implementations