Smart Document Q&A with Follow-ups - Implementation Plan

Project Overview

Build a RAG system that not only answers questions but intelligently decides when to ask clarifying questions to provide better, more targeted responses.

Architecture Components

1. Document Processing Pipeline

PDF/TXT Upload → Text Extraction → Chunking → Embeddings → Vector Store

2. Agent Decision Layer

User Query → Intent Analysis → Decide: Direct Answer OR Ask Clarification → Response

3. RAG Retrieval System

Query → Embedding → Vector Search → Chunk Ranking → Context Assembly

Phase 1: Basic RAG Foundation (Week 1)

Day 1-2: Environment Setup

Tech Stack:

- **Backend**: Python with FastAPI
- **Vector DB**: Pinecone (free tier) or ChromaDB (local)
- **LLM**: OpenAI GPT-4 or Claude
- **Frontend**: Streamlit for rapid prototyping
- **Document Processing**: PyPDF2, python-docx, langchain

Setup Steps:

1. Create virtual environment
2. Install dependencies: `openai`, `pinecone-client`, `streamlit`, `langchain`, `pypdf2`
3. Set up API keys (OpenAI, Pinecone)
4. Create basic Streamlit interface for file upload

Day 3-4: Document Processing

Core Functions to Build:

```python
def extract_text_from_pdf(file_path):
    # Extract text from PDF
    pass

def chunk_text(text, chunk_size=1000, overlap=200):
    # Split text into overlapping chunks
    pass

def create_embeddings(chunks):
    # Generate embeddings using OpenAI
    pass

def store_in_vector_db(chunks, embeddings, metadata):
    # Store in Pinecone/ChromaDB
    pass
```

Implementation Priority:

1. PDF text extraction
2. Text chunking with metadata (page numbers, section titles)
3. Embedding generation
4. Vector database storage

Day 5-7: Basic RAG Query

Core Functions:

```python
def retrieve_relevant_chunks(query, top_k=5):
    # Vector similarity search
    pass

def generate_answer(query, context_chunks):
    # LLM call with retrieved context
    pass
```

Streamlit Interface:

- File upload component
- Query input box
- Display retrieved chunks + final answer

Phase 2: Agent Decision Layer (Week 2)

Day 8-10: Query Analysis Agent

Intent Classification: Build a system that categorizes queries into:

- **Clear & Specific**: "What is the ROI calculation on page 15?"
- **Vague**: "Tell me about the financial performance"
- **Multi-faceted**: "How does this compare to industry standards?"
- **Out of scope**: "What's the weather today?"

Implementation:

```python
def analyze_query_intent(query, document_metadata):
    prompt = f"""
    Analyze this query: "{query}"
    Document contains: {document_metadata}

    Classify as:
    1. CLEAR - specific, answerable directly
    2. VAGUE - needs clarification
    3. MULTI_FACETED - complex, might need follow-up
    4. OUT_OF_SCOPE - not related to document

    If VAGUE or MULTI_FACETED, suggest 2-3 clarifying questions.
    """
    return llm_call(prompt)
```

Day 11-12: Follow-up Question Generator

Smart Questions Based on Context:

- Document type awareness: "Are you looking for technical implementation or business impact?"
- Ambiguity resolution: "Which quarter's data are you interested in?"

- Scope clarification: "Do you want a summary or detailed breakdown?"

Implementation:

```python
def generate_followup_questions(query, retrieved_chunks, intent_analysis):
    # Analyze what aspects could be clarified
    # Generate 2-3 specific follow-up questions
    pass
```

Day 13-14: Decision Engine Integration

Flow Control Logic:

```python
def handle_user_query(query):
    intent = analyze_query_intent(query)

    if intent['type'] == 'CLEAR':
        return direct_rag_answer(query)
    elif intent['type'] == 'VAGUE':
        return ask_clarification(intent['questions'])
    elif intent['type'] == 'MULTI_FACETED':
        initial_answer = direct_rag_answer(query)
        follow_ups = generate_followup_questions(query)
        return combine_answer_and_followups(initial_answer, follow_ups)
    else:
        return "I can only answer questions about the uploaded document."
```

Phase 3: Enhanced User Experience (Bonus)

Conversation Memory

- Track conversation history
- Remember user preferences ("focus on technical details")
- Build context across multiple questions

Confidence Scoring

- Rate answer confidence based on retrieval similarity scores

- Show uncertainty: "I found some relevant information, but want to make sure I understand what you're looking for..."

Interactive Clarification

- Multiple choice follow-ups
- Progressive disclosure (start broad, get specific)

Technical Implementation Details

Vector Database Schema

```python
# Pinecone index structure
{
    "id": "doc1_chunk_1",
    "values": [embedding_vector],
    "metadata": {
        "document_id": "doc1",
        "chunk_index": 1,
        "page_number": 1,
        "section": "Introduction",
        "text": "original_chunk_text"
    }
}
```

Prompt Engineering Templates

```python
```

```
RAG_PROMPT = """
Context: {retrieved_chunks}

User Question: {query}

Instructions: Answer based on the provided context. If the context doesn't contain enough information, say so clearly.

Answer:
"""

CLARIFICATION_PROMPT = """
The user asked: "{query}"

Based on the document contents, this query could be interpreted multiple ways.

Generate 2-3 clarifying questions that would help provide a better answer. Format as:
1. [Question focusing on scope]
2. [Question focusing on detail level]
3. [Question focusing on specific aspect]
"""
```

Error Handling & Edge Cases

- Empty documents
- Corrupted PDFs
- Very long documents (pagination)
- Multiple file uploads
- Query rate limiting

Testing Strategy

Unit Tests

- Document processing functions
- Embedding generation
- Vector retrieval accuracy

Integration Tests

- End-to-end query flow

- Follow-up question quality
- Response relevance

User Testing

- Upload various document types
- Test with different query styles
- Validate follow-up question usefulness

Deployment Considerations

Local Development

- Streamlit app for quick iteration
- Local ChromaDB for testing

Production Ready

- FastAPI backend
- Docker containerization
- Environment variable management
- Proper error handling and logging

Success Metrics

- **Accuracy**: Relevant answers to direct questions
- **Intelligence**: Appropriate follow-up questions generated
- **User Experience**: Smooth clarification flow
- **Edge Case Handling**: Graceful failure modes

Next Steps After Completion

1. Add support for multiple document types (Word, PowerPoint)
2. Implement conversation memory
3. Add document comparison features
4. Build more sophisticated intent classification

This project will give you hands-on experience with both RAG fundamentals and agentic decision-making, setting a strong foundation for more complex projects.