**Knowledge Extraction from Engineering PDFs: A RAG Approach**

Here's a step-by-step approach to implement this project:

**1. Project Setup & Data Collection**

* Set up your development environment with necessary Python libraries
* Collect a dataset of engineering PDFs (textbooks, research papers, technical manuals)
* Organize PDFs by engineering domain (mechanical, electrical, civil, etc.)
* Create a validation set for testing your system

**2. PDF Processing Pipeline**

* Implement PDF extraction using tools like PyPDF2, PDFMiner, or PyMuPDF
* Extract text content while preserving structure (paragraphs, sections)
* Extract tables using specialized tools (Camelot, Tabula)
* Extract images, diagrams, and figures
* Process mathematical equations (using MathPix API or similar tools)

**3. Text Preprocessing**

* Clean extracted text (remove headers/footers, normalize whitespace)
* Split documents into chunks of appropriate size (paragraphs or sections)
* Implement special handling for engineering terminology
* Develop metadata extraction (titles, authors, sections)

**4. Vector Database Implementation**

* Choose an embedding model appropriate for engineering text (consider models trained on technical content)
* Generate embeddings for each document chunk
* Set up a vector database (Pinecone, Weaviate, Chroma, etc.)
* Implement indexing and retrieval functions

**5. RAG System Architecture**

* Integrate an LLM (e.g., Claude, GPT, Llama) via API or local deployment
* Implement query processing and reformulation
* Build the retrieval component to fetch relevant chunks
* Design a generation component that synthesizes information from retrieved chunks
* Create a feedback mechanism to improve retrieval quality

**6. Engineering Domain Adaptation**

* Fine-tune embedding models on engineering documents (if resources allow)
* Create domain-specific prompting templates for different engineering disciplines
* Implement specialized entity extraction for engineering concepts
* Build knowledge validation modules using engineering principles

**7. User Interface Development**

* Create a simple web interface using Flask/Django/Streamlit
* Implement PDF upload functionality
* Build query input and result display components
* Add visualization tools for document exploration
* Include citation tracking to original source pages

**8. Evaluation Framework**

* Design evaluation metrics relevant to engineering knowledge extraction
* Implement automated evaluation using test queries with known answers
* Create a human evaluation protocol for assessing factual correctness
* Measure performance against baseline methods (e.g., basic keyword search)

**9. Optimization & Enhancement**

* Optimize retrieval parameters (chunk size, similarity thresholds)
* Implement caching mechanisms for frequent queries
* Add multi-hop reasoning capabilities for complex engineering questions
* Incorporate domain-specific knowledge bases or ontologies

**10. Deployment & Documentation**

* Package the application for easy deployment
* Create comprehensive documentation for users
* Build a demonstration dataset of sample engineering PDFs
* Prepare presentation materials showing the system's capabilities

This project would demonstrate your ability to apply RAG techniques to specialized technical content while addressing the unique challenges of engineering documents, like technical terminology, equations, and diagrams.

**How to Use the Python Script-readme file**

1. **Save the code** to a file named rag.py
2. **Install required packages** (the script will attempt to do this automatically):

*pip install PyPDF2 langchain sentence-transformers chromadb torch transformers openai pypdf faiss-cpu tiktoken*

1. **Prepare your PDFs**:
   * Create a folder named pdf\_data in the same directory as the script
   * Place your engineering PDFs in this folder
2. **Run the script**: Basic usage:

*python rag.py*

This will:

* + Process all PDFs in the pdf\_data directory
  + Create a vector database
  + Launch an interactive query mode

1. **Command-line options**:
   * Load existing database:

*python rag.py --load*

* + Run a specific query:

*python rag.py --query "How does a centrifugal pump work?"*

* + Use OpenAI directly instead of LangChain:

*python rag.py --openai --query "What is finite element analysis?"*

* + Interactive mode (default):

*python rag.py --interactive*

1. **In interactive mode**:
   * Type 'exit' to quit
   * Type 'openai' to switch to direct OpenAI querying
   * Type 'langchain' to switch to LangChain QA

The script will generate two directories:

* pdf\_data: Where you should place your engineering PDFs
* vector\_db: Where the vector database is stored

The script will automatically ask for your OpenAI API key if it's not set as an environment variable.

Retry

Claude does not have the ability to run the code it generates yet.

How to get Open API Key

Get an OpenAI API key:

* Go to <https://platform.openai.com/>
* Sign up or log into your OpenAI account
* Navigate to "API Keys" or "Settings" section
* Create a new API key

 When the script prompts "Please enter your OpenAI API key:", paste your API key and press Enter.

**For interview preparation**

Introduction to RAG (Retrieval-Augmented Generation) and Its Uses

Retrieval-Augmented Generation (RAG) is a powerful AI architecture that combines the strengths of large language models (LLMs) with external knowledge retrieval systems. Unlike standard LLMs that rely solely on their pre-trained knowledge, RAG systems can access, retrieve, and incorporate relevant information from external data sources before generating responses.

How RAG Works

RAG operates in three main stages:

1. Indexing: Documents are processed, chunked, and converted into vector embeddings that capture their semantic meaning. These embeddings are stored in a vector database.
2. Retrieval: When a query is received, it's converted into the same embedding space, and similar documents are retrieved from the vector database using semantic similarity search.
3. Generation: The LLM generates a response based on both the user query and the retrieved relevant context.

Key Uses and Applications

1. Enterprise Knowledge Management

* Internal Documentation: Making corporate wikis, manuals, and SOPs accessible through natural language queries
* Expertise Location: Finding subject matter experts based on authored documents
* Compliance Support: Providing up-to-date regulatory information and guidance

2. Enhanced Customer Support

* Self-Service Support: Letting customers find answers from knowledge bases
* Agent Assistance: Providing service agents with relevant information during customer interactions
* Troubleshooting: Retrieving specific technical solutions based on described problems

3. Research and Analysis

* Literature Review: Finding and synthesizing information across large document collections
* Competitive Intelligence: Analyzing market trends and competitor information
* Patent Analysis: Exploring technical innovations and IP landscapes

4. Education and Training

* Personalized Learning: Creating tailored educational content from course materials
* Question Answering: Providing specific answers from textbooks and learning resources
* Course Development: Assembling relevant materials for curriculum creation

5. Technical Documentation

* Code Documentation: Answering questions about codebase functionality
* API Knowledge: Retrieving specific endpoint information and usage examples
* Maintenance Support: Finding relevant technical diagrams and procedures

6. Healthcare and Medicine

* Clinical Decision Support: Retrieving relevant medical literature and guidelines
* Research Synthesis: Combining information from multiple medical studies
* Patient Information: Providing accurate health information from trusted sources

Advantages of RAG

* Up-to-date Information: Access to the latest documents and data beyond the LLM's training cutoff
* Reduced Hallucinations: Grounding responses in retrieved facts rather than generated content
* Transparency: Ability to cite specific sources for provided information
* Domain Adaptation: Easily adapting general models to specialized knowledge domains
* Cost Efficiency: Using smaller models with external knowledge rather than larger models

RAG represents a significant advancement in making AI systems more reliable, trustworthy, and useful for knowledge-intensive tasks across virtually every industry and domain.

**Linkedin description reference**

"Developed an Engineering Knowledge Extraction System using Retrieval-Augmented Generation (RAG) technology. The project processes technical PDFs, creates a searchable knowledge base, and uses AI to answer domain-specific engineering questions with source citations. Built with Python, leveraging vector databases and large language models to provide accurate, context-aware responses from engineering documentation."

You could also add a brief bullet point about specific technologies if you want to highlight your technical stack:

* Tech stack: Python, LangChain, HuggingFace embeddings, ChromaDB, OpenAI API