Build Your First RAG Chatbot - Complete Beginner Guide

What You'll Learn

By the end of this tutorial, you'll understand:

- What RAG (Retrieval Augmented Generation) means
- How to build a chatbot that answers from your documents
- Vector databases and embeddings (simplified!)
- How to prevent AI hallucination

Prerequisites

- Basic Python knowledge (variables, functions, loops)
- Know how to run Python scripts
- Understand what APIs are

Key Libraries & Tools We'll Use

🧠 Google Generative AI (genai)

- **What it does**: The "brain" that generates human-like responses
- **Why we use it**: Fast, accurate, and follows instructions well
- **Think of it as**: A very smart assistant that can read and write

🔢 Sentence Transformers

- **What it does**: Converts text into numbers (vectors) that computers can compare
- **Why we use it**: Finds similar meanings even with different words
- **Think of it as**: A translator that turns words into math
- **Example**: "cat" and "kitten" get similar numbers because they're related

💾 ChromaDB

- **What it does**: Stores and searches through millions of number vectors super fast
- **Why we use it**: Like Google search but for your document chunks
- **Think of it as**: A super-smart filing cabinet that finds related documents instantly
- **Cool feature**: Remembers everything even after you restart the program

📄 PyPDF2

- **What it does**: Reads text from PDF files
- **Why we use it**: Most documents are in PDF format
- **Think of it as**: A tool that copies text from PDFs so our AI can read it

🌐 Streamlit

- **What it does**: Creates web apps with just Python (no HTML/CSS needed!)
- **Why we use it**: Makes a beautiful chat interface in minutes
- **Think of it as**: Magic that turns Python scripts into websites

🔑 Python-dotenv

- **What it does**: Safely stores secret keys (like API keys)
- **Why we use it**: Keeps your API keys secure and separate from code
- **Think of it as**: A secure vault for passwords

🏷️ Key Terms You'll Learn

Vector/Embedding: A list of numbers that represents the "meaning" of text

```
"I love cats" → [0.2, 0.8, 0.1, 0.9, ...] # 384 numbers!
```

Similarity Search: Finding text with similar meanings by comparing their numbers

```
"What are cats?" finds "Cats are pets" because their numbers are similar
```

Chunking: Breaking long documents into smaller pieces

```
"Long document..." → ["Piece 1", "Piece 2", "Piece 3"]
```

Context: The relevant document pieces we show to the AI

```
User asks: "What do cats eat?"
Context: "Cats eat fish. Cats like milk. Cats hunt mice."
AI answers using only this context
```

RAG Pipeline: The complete process

```
Documents → Chunks → Vectors → Database → Search → Context → AI →
Answer
```

Step-by-Step Learning Path

Phase 1: Understanding the Concepts (15 minutes)

What is RAG?

Think of RAG like a smart student taking an open-book exam:

- **Regular AI**: Answers from memory (might make mistakes)
- **RAG AI**: Looks up answers in provided books first, then responds

The Magic Behind RAG

```
Your Documents → Convert to Numbers → Store in Database → Search →
Answer
```

1. **Documents**: Your PDFs, text files
2. **Convert to Numbers**: Each piece of text becomes a list of numbers (vectors)
3. **Store**: Save these numbers in a searchable database
4. **Search**: Find similar numbers when you ask a question
5. **Answer**: AI reads only the found text and responds

Phase 2: Setting Up Your Environment (10 minutes)

Install Python Tools

```
# Create a new folder for your project
mkdir my-rag-chatbot
cd my-rag-chatbot
```

```
# Install required packages (this might take a few minutes)
pip install streamlit sentence-transformers chromadb google-generativeai
```

What each package does:

- `streamlit` → Web interface magic ✨
- `sentence-transformers` → Text-to-numbers converter 🔢
- `chromadb` → Super-fast search database 💾
- `google-generativeai` → The AI brain 🧠
- `python-dotenv` → Secret key manager 🔐
- `pypdf2` → PDF text extractor 📄

Get Your AI API Key

1. Go to Google AI Studio
2. Create a free account
3. Generate an API key
4. Save it somewhere safe

Phase 3: Build Step by Step

Step 1: Create the Brain (15 minutes)

Create `simple_rag.py`:

```
# This is like the brain of our chatbot
import google.generativeai as genai            # 🧠 Google's AI
from sentence_transformers import SentenceTransformer  # 🔢 Text → Numbe
import chromadb                                # 💾 Vector database

class SimpleChatbot:
    def __init__(self, api_key):
        # Set up the AI (Google Gemini)
        genai.configure(api_key=api_key)
        self.ai = genai.GenerativeModel('gemini-pro')

        # Set up the "memory" (converts text to numbers)
        # This model is small, fast, and good quality
        self.memory = SentenceTransformer('all-MiniLM-L6-v2')

        # Set up the "filing cabinet" (stores the numbers)
        self.cabinet = chromadb.Client()
        self.files = self.cabinet.create_collection("my_docs")

    def learn_document(self, text, doc_name):
        """Teach the chatbot about a document"""
        # Break text into small pieces
        pieces = [text[i:i+500] for i in range(0, len(text), 400)]

        # Convert pieces to numbers
        numbers = self.memory.encode(pieces)

        # Store in filing cabinet
        ids = [f"{doc_name}_{i}" for i in range(len(pieces))]
        self.files.add(
            embeddings=numbers.tolist(),
            documents=pieces,
            ids=ids
        )
        print(f"✅ Learned about {doc_name}")

    def answer_question(self, question):
        """Answer a question using learned documents"""
        # Convert question to numbers
        question_numbers = self.memory.encode([question])

        # Search filing cabinet for similar pieces
        results = self.files.query(
            query_embeddings=question_numbers.tolist(),
            n_results=3
        )

        # Combine found pieces
        context = "\n".join(results['documents'][0])

        # Ask AI to answer using only these pieces
        prompt = f"""
        Answer this question using ONLY the provided context.
        If the answer isn't in the context, say "I don't know."

        Context: {context}
        Question: {question}
        Answer:
        """

        response = self.ai.generate_content(prompt)
```

```python
        return response.text

# Test it!
if __name__ == "__main__":
    # Replace with your API key
    bot = SimpleChatbot("your-api-key-here")

    # Teach it something
    bot.learn_document("Python is a programming language. It's easy to le

    # Ask it something
    answer = bot.answer_question("What is Python?")
    print(answer)
```

Try this first! Run it and see if it works.

Step 2: Add File Reading (10 minutes)

Create `file_reader.py`:

```python
import PyPDF2  # 📄 The PDF text extractor

def read_pdf(file_path):
    """Read text from a PDF file using PyPDF2"""
    with open(file_path, 'rb') as file:  # 'rb' = read binary
        reader = PyPDF2.PdfReader(file)   # Create PDF reader
        text = ""
        # Loop through each page and extract text
        for page in reader.pages:
            text += page.extract_text()
    return text

def read_txt(file_path):
    """Read text from a TXT file"""
    with open(file_path, 'r') as file:
        return file.read()

# Test it!
if __name__ == "__main__":
    # Create a test file
    with open("test.txt", "w") as f:
        f.write("This is a test document about cats. Cats are amazing pet
    text = read_txt("test.txt")
    print(f"Read: {text}")
```

Step 3: Combine Everything (15 minutes)

Create `complete_chatbot.py`:

```
from simple_rag import SimpleChatbot
from file_reader import read_pdf, read_txt
import os

def main():
    # Get API key
    api_key = input("Enter your Google AI API key: ")

    # Create chatbot
    bot = SimpleChatbot(api_key)

    # Load documents from a folder
    docs_folder = "documents"
    if not os.path.exists(docs_folder):
        os.makedirs(docs_folder)
        print(f"Created {docs_folder} folder. Add your PDF/TXT files ther
        return

    # Read all files in the folder
    for filename in os.listdir(docs_folder):
        file_path = os.path.join(docs_folder, filename)

        if filename.endswith('.pdf'):
            text = read_pdf(file_path)
            bot.learn_document(text, filename)
        elif filename.endswith('.txt'):
            text = read_txt(file_path)
            bot.learn_document(text, filename)

    # Chat loop
    print("\n🤖 Chatbot ready! Ask me about your documents (type 'quit' t
    while True:
        question = input("\nYou: ")
        if question.lower() == 'quit':
            break
        answer = bot.answer_question(question)
        print(f"Bot: {answer}")

if __name__ == "__main__":
    main()
```

Step 4: Add Web Interface (20 minutes)

Create `web_app.py`:

```
import streamlit as st
from complete_chatbot import SimpleChatbot
import os

st.title("🤖 My First RAG Chatbot")

# Get API key
api_key = st.text_input("Enter your Google AI API key:", type="password")

if api_key:
    # Create chatbot
    if 'bot' not in st.session_state:
        st.session_state.bot = SimpleChatbot(api_key)

        # Load documents
        docs_folder = "documents"
        if os.path.exists(docs_folder):
            for filename in os.listdir(docs_folder):
                if filename.endswith(('.pdf', '.txt')):
                    # Load document logic here
                    st.success(f"Loaded {filename}")

    # Chat interface
    question = st.text_input("Ask me about your documents:")

    if question:
        answer = st.session_state.bot.answer_question(question)
        st.write(f"**Answer:** {answer}")
```

Phase 4: Understanding What You Built

The Vector Magic Explained

```
# When you add "Cats are pets"
text = "Cats are pets"
vector = [0.1, 0.5, 0.8, 0.2, ...]  # 384 numbers!

# When you ask "What are cats?"
question = "What are cats?"
q_vector = [0.2, 0.4, 0.9, 0.1, ...]  # Similar numbers!

# The system finds similar vectors and returns the text
```

Why This Prevents Hallucination

- AI only sees the text pieces you found
- No access to its training data
- Must say "I don't know" if answer isn't in your documents

Phase 5: Common Beginner Mistakes & Solutions

Mistake 1: "My chatbot gives weird answers"

Problem: Chunks are too small or too big
Solution: Adjust chunk size (try 500-1000 characters)

Mistake 2: "It says 'I don't know' for everything"

Problem: Search isn't finding relevant pieces
Solution: Check if documents loaded correctly, try different search terms

Mistake 3: "It's too slow"

Problem: Processing large documents
Solution: Use smaller documents for testing first

Phase 6: Next Steps

Make It Better

1. **Add more file types**: Word docs, web pages
2. **Better chunking**: Split by sentences, not characters
3. **Multiple languages**: Use different embedding models
4. **Better UI**: Add file upload, chat history
5. **Deploy it**: Put it online with Heroku or Streamlit Cloud

Advanced Concepts to Learn Later

- **Hybrid search**: Combine keyword + vector search
- **Reranking**: Improve search results
- **Fine-tuning**: Train models on your specific data
- **Evaluation**: Measure how good your answers are

Quick Reference

Essential Commands

```
# Install everything
pip install streamlit sentence-transformers chromadb google-generativeai

# Run web app
streamlit run web_app.py

# Run command line version
python complete_chatbot.py
```

Key Concepts & Libraries Summary

Core Libraries

- `google-generativeai` : Google's AI that generates responses
- `sentence-transformers` : Converts text to searchable numbers
- `chromadb` : Fast vector database for storing document chunks
- `pypdf2` : Reads text from PDF files
- `streamlit` : Creates web interfaces with Python
- `python-dotenv` : Manages environment variables securely

Essential Concepts

- **Embedding/Vector**: Text converted to numbers for similarity comparison
- **Vector Database**: Storage system optimized for finding similar vectors
- **Similarity Search**: Finding related content by comparing number patterns
- **Context Window**: Amount of text the AI can process at once
- **Chunking**: Splitting documents into manageable pieces
- **RAG Pipeline**: Document → Vector → Search → Context → AI Response

Troubleshooting

"ModuleNotFoundError"

```
pip install [missing-module-name]
```

"API Key Error"

- Check your Google AI Studio account
- Make sure you copied the key correctly
- Try generating a new key

"No documents found"

- Make sure files are in the `documents/` folder
- Check file extensions (.pdf, .txt)
- Try with a simple .txt file first

Congratulations! 🎉

You've built a RAG chatbot that:

- ✅ Reads your documents
- ✅ Converts them to searchable vectors
- ✅ Finds relevant information
- ✅ Answers questions without hallucination
- ✅ Has a web interface

You now understand the core concepts behind modern AI applications!