Document-Aware Conversational Assistant

A RAG-Based NLP System
Natural Language Processing
Ishan Srivastava
14th Dec 2025

1. Abstract

This project implements a Document-Aware Conversational Assistant using Retrieval-Augmented Generation (RAG). Users can upload PDF documents and ask questions in natural language. The system retrieves relevant content using semantic embeddings and generates accurate, grounded answers using a large language model. The project demonstrates core NLP concepts including text preprocessing, sentence embeddings, cosine similarity search, and context-aware generation.

2. Problem Statement

Finding specific information in documents is challenging. Traditional keyword search fails when users use different words than the document (vocabulary mismatch problem). For example, searching "vacation policy" won't find content about "annual leave." This project solves this using semantic search that understands meaning, not just keywords.

3. Solution: RAG Architecture

RAG = Retrieval-Augmented Generation

Instead of asking an LLM to guess from memory (which causes hallucination), we:

1. **Retrieve** relevant chunks from the user's document
2. **Augment** the prompt with retrieved content
3. **Generate** answers grounded in actual sources

This reduces hallucination and provides source attribution for transparency.

4. Project Structure & Files

```
document-assistant/
├── config.py            # Configuration constants
├── utils.py             # Helper functions
├── nlp_core.py          # Core NLP logic
├── app.py               # Streamlit UI
├── requirements.txt     # Dependencies
├── .env                 # API keys (not in git)
└── README.md            # Documentation
```

File Descriptions:

File	Purpose	Key Contents
config.py	Centralized settings	API keys, model names, chunk size (800), overlap (200), top-k (4), temperature (0.7)
utils.py	Helper functions	truncate_text(), format_percentage(), parse_key_value_string(), safe_get()
nlp_core.py	All NLP logic	Text extraction, cleaning, chunking, embedding generation, similarity search, RAG answer generation
app.py	User interface	Streamlit UI with chat, infographic, and image prompt tabs
requirements.txt	Dependencies	streamlit, sentence-transformers, pymupdf, numpy, openai, python-dotenv
.env	Secrets	OPENAI_API_KEY=sk-xxxxx (excluded from git)

5. System Architecture

INDEXING PHASE (Document Upload):
PDF → Extract (PyMuPDF) → Clean → Chunk (800c/200o) → Embed (384-dim) → Store in RAM

QUERY PHASE (User Question):
Question → Embed → Cosine Similarity → Top-4 Chunks → Build Prompt → GPT-4o-mini → Answer + Sources

6. Key Implementation Details

6.1 Text Processing

- **Extraction:** PyMuPDF extracts text from PDF pages
- **Cleaning:** Removes noise (extra newlines, spaces, special characters)
- **Chunking:** 800 character chunks with 200 character overlap and smart sentence boundary detection

6.2 Embeddings & Search

- **Model:** all-MiniLM-L6-v2 (sentence-transformers)
- **Output:** 384 dimensional vectors per chunk
- **Search:** Cosine similarity (measures angle between vectors)
- **Retrieval:** Top 4 most similar chunks returned

6.3 Answer Generation

- **LLM:** OpenAI GPT-4o-mini
- **Prompt:** Retrieved chunks + last 3 conversation turns + current question
- **Constraint:** "Answer ONLY from document content" (prevents hallucination)

6.4 Memory Solution

- **Problem:** LLMs have no built-in memory between API calls
- **Solution:** Store chat history in `st.session_state.chat_history` and inject last 3 turns into every prompt
- **Technique:** Prompt injection / context stuffing

6.5 Infographic Generation

- **Method:** LLM extracts structured data → Parse to dictionary → Inject into HTML template
- **Why HTML:** Fast, free, no image API needed, downloadable, customizable

7. Technologies Used

Component	Technology	Why Chosen
UI	Streamlit	Python-native, built-in chat components
PDF Extraction	PyMuPDF	Fast, reliable
Embeddings	sentence-transformers	Free, local, high quality
LLM	OpenAI GPT-4o-mini	Reliable, affordable, easy setup
Vector Storage	NumPy (RAM)	Simple, sufficient for demo scale
Similarity	Cosine similarity	Measures meaning direction, not length

Why NOT Alternatives:

- **React/Flask:** Requires separate frontend, more complex
- **Word2Vec:** Word-level only, no sentence context
- **Local LLM:** Needs GPU, complex setup
- **FAISS/Pinecone:** Overkill for project scale

8. Features

- PDF upload and text extraction
- Natural language Q&A
- Source attribution (shows retrieved chunks)
- Conversation memory (follow-up questions work)
- Multi-document support
- Infographic generation (HTML-based)
- AI image prompt generator

9. Key Parameters

Parameter	Value	Reason
Chunk size	800 chars	~1 paragraph, balances context vs precision
Overlap	200 chars	25% prevents info loss at boundaries
Top-k	4 chunks	Balance of coverage vs noise
Embedding dims	384	MiniLM model output
Memory turns	3	Balance of context vs API cost
Temperature	0.7	Balanced creativity

10. Limitations

Limitation	Reason	Future Fix
Data lost on refresh	In-memory storage (RAM)	Add database
Needs internet	OpenAI API dependency	Local LLM (Llama)
Text only	No vision processing	Multi-modal RAG
Small scale	NumPy brute-force search	FAISS/Pinecone
PDF noise	Complex layouts extract poorly	Better extraction tools

11. Future Scope

- Vector database (FAISS/Pinecone) for scale and persistence
- Local LLM (Llama/Mistral) for offline use
- Multi-modal RAG for images and tables
- Hybrid search (semantic + keyword)
- User authentication for personal libraries

12. How to Run

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

# 2. Create .env file
echo "OPENAI_API_KEY=key is already in code" > .env

# 3. Run application
streamlit run app.py
```

13. Conclusion

This project successfully implements a RAG-based document assistant demonstrating core NLP concepts: text preprocessing, semantic embeddings, similarity-based retrieval, and context-grounded generation. The system solves the vocabulary mismatch problem through semantic search and reduces hallucination by constraining the LLM to retrieved content. The same architecture powers commercial products like ChatGPT with file uploads and Google NotebookLM.

14. References

1. Lewis, P., et al. (2020). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." NeurIPS.
2. Reimers, N., & Gurevych, I. (2019). "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks."
3. Sentence-Transformers Documentation: https://www.sbert.net/
4. Streamlit Documentation: https://docs.streamlit.io/
5. OpenAI API Documentation: https://platform.openai.com/docs/

15. Appendix: Complete File Contents

requirements.txt

```
streamlit>=1.28.0
sentence-transformers>=2.2.0
pymupdf>=1.23.0
numpy>=1.24.0
openai>=1.0.0
python-dotenv>=1.0.0
```

.env (sample)

```
OPENAI_API_KEY=sk-your-api-key-here
```

.gitignore

```
.env
__pycache__/
*.pyc
.streamlit/
```

Total Files: 6 (config.py, utils.py, nlp_core.py, app.py, requirements.txt, .env)
Total Lines of Code: ~600
Key NLP Techniques: Text preprocessing, Embeddings, Cosine similarity, RAG, Prompt engineering