Project Report: RAG Chatbot (FLAN-T5-Small)

1. Introduction

The goal of this project was to build a Retrieval-Augmented Generation (RAG) chatbot that can answer user queries based on a provided PDF document. By combining semantic search over document chunks with a lightweight, instruction-tuned language model (FLAN-T5-Small), the chatbot delivers grounded, context-aware responses in real time via a simple Gradio interface.

Key components:

- **Document Chunking:** Split the PDF into overlapping text segments.
- **Embedding & Indexing:** Convert chunks into normalized vector embeddings using All-MiniLM-L6-v2, and store in-memory.
- **Retrieval:** Compute cosine similarity between the query embedding and chunk embeddings to retrieve top-k relevant passages.
- **Generation:** Use FLAN-T5-Small to produce answers conditioned on retrieved contexts.
- **Interface:** A Gradio web app that accepts questions and displays responses interactively.

2. Architecture Overview

```
User Query
  ↓
Query Embedding (sentence-transformers)
  ↓     ↘
  ↓   Top-k Retrieval (cosine similarity)
  ↓     ↘
Retrieved Chunks ──────────────────────▶ Prompt Construction
                         ↓
               Generation (T5ForConditionalGeneration)
                         ↓
                 Final Answer
                         ↓
                Gradio Interface
```

3. Data Preparation & Chunking

- **Library:** pdfplumber
- **Strategy:** Extract text from each PDF page, concatenate, and split into 300-word chunks with 50-word overlap.
- **Rationale:** Overlap ensures that context boundaries don't omit critical sentences.

```
chunks = []
for i in range(0, len(words), step):
    chunk = words[i:i+300]
    chunks.append(" ".join(chunk))
```

Result: *N* chunks covering the entire document (e.g. ~40–50 for a 10,000-word PDF).

4. Embedding & Indexing

- **Embedding Model:** all-MiniLM-L6-v2 (384-dimensional embeddings).
- **Normalization:** Each embedding is L2-normalized for cosine similarity.
- **Index:** In-memory NumPy array of shape (num_chunks, 384).

```
embs = embedder.encode(chunks)
norms = np.linalg.norm(embs, axis=1, keepdims=True)
chunk_embs = embs / norms
```

Advantages: No external vector database dependency; simple and fast for moderate document sizes.

5. Retrieval Mechanism

- **Query Embedding:** Also generated with the same embedder and normalized.
- **Similarity Computation:** Dot product between normalized embeddings yields cosine similarity.
- **Top-k Selection:** np.argsort(-sims)[:k] to pick the k most relevant chunks.

```
def retrieve(query, k=3):
    q_emb = embedder.encode([query]) / np.linalg.norm(...)
    sims = chunk_embs @ q_emb.T
    top_indices = np.argsort(-sims)[:k]
    return [chunks[i] for i in top_indices]
```

Parameter k can be tuned; default is 3 to balance relevance versus context volume.

6. Generation via FLAN-T5-Small

- **Model:** google/flan-t5-small (~80M parameters), instruction-tuned on diverse tasks.
- **Prompt Template:** Prepends retrieved contexts as bullet points, then appends the user question:

- Answer based on contexts:
- - <chunk1>
- - <chunk2>
- ...
- Question: <user_query>
- Answer:
- **Decoding:** Greedy/beam generation up to 128 new tokens.

inputs = tokenizer(prompt, return_tensors="pt")

out = model.generate(**inputs, max_new_tokens=128)

answer = tokenizer.decode(out[0])

GPU Support: Automatically moves model and inputs to CUDA if available.

7. Gradio Interface

- **Components:** Textbox for query, slider for k, and output textbox for the answer.
- **Launch:** Runs on localhost:7860 by default.
- **User Flow:** Enter question → click Submit → view grounded answer in seconds.

demo = gr.Interface(fn=answer_fn, inputs=[inp, slider], outputs=out, title="RAG Chatbot")

demo.launch()

8. Setup & Execution

1. **Install dependencies:**
2. pip install sentence-transformers pdfplumber transformers torch gradio numpy
3. **Place PDF:** Ensure AI Training Document.pdf is in the same folder.
4. **Run script:**
5. python rag_chatbot.py

9. Example Queries

Query	k	Sample Response Summary
"What is the main purpose of this document?"	3	Summarizes RAG pipeline goals
"How are text chunks generated?"	2	Describes 300-word chunks with 50-word overlap
"Which model is used for embeddings?"	1	Mentions all-MiniLM-L6-v2
"Explain the retrieval process."	4	Details cosine similarity and top-k selection

10. Limitations & Future Work

- **Model Capacity:** FLAN-T5-Small may struggle with very long contexts or complex reasoning.
- **Scalability:** In-memory retrieval suffices for small docs but would need FAISS or Chroma for large corpora (>10K chunks).
- **Evaluation:** Could integrate metrics like retrieval accuracy and response fidelity.
- **Extensions:** Add streaming token-by-token UI, citation links back to chunk pages, or allow multi-document ingestion.

Conclusion: This RAG chatbot demonstrates a lightweight yet effective pipeline to ground LLM responses in document content, using off-the-shelf open-source models and simple in-memory retrieval for rapid prototyping and user interaction.

SCREENSHOTS: