ContextIQ

Local RAG-Based Secure Document Q&A System

Technical Project Report

Retrieval-Augmented Generation • Document Intelligence • Privacy-First AI

Executive Summary

ContextIQ is a fully local Retrieval-Augmented Generation (RAG) system that enables users to upload documents and ask AI-powered questions about their content without relying on external APIs or cloud services. Built with privacy and security as core principles, the system processes all data locally on the user's machine, ensuring complete data sovereignty.

The system combines state-of-the-art natural language processing techniques including semantic search, vector embeddings, and local language model inference to deliver accurate, context-aware answers. It supports multiple document formats (PDF, Word, Excel, PowerPoint, and text files) and provides an intuitive Streamlit-based interface with conversation history and source attribution.

Key Features

- **100% Local Processing:** No external API calls, complete privacy
- **Multi-Format Support:** PDF, DOCX, XLSX, PPTX, TXT
- **Semantic Search:** Sentence transformers with FAISS vector database
- **Local LLM:** OpenVINO-optimized Phi-3 model
- **Interactive UI:** Streamlit-based interface with conversation history
- **Source Attribution:** Shows which document sections generated answers
- **Knowledge Graph:** RDF-based document relationship tracking

1. System Architecture

1.1 High-Level Architecture

ContextIQ follows a modular architecture with clear separation of concerns:

1. **Document Upload:** User uploads document through Streamlit interface
2. **Document Parsing:** Extract text from various file formats
3. **Text Chunking:** Split document into semantically meaningful chunks
4. **Embedding Generation:** Convert chunks to vector embeddings
5. **Vector Indexing:** Store embeddings in FAISS index
6. **Question Processing:** User asks question via interface
7. **Semantic Search:** Find relevant document chunks
8. **Context Assembly:** Compile retrieved chunks
9. **Answer Generation:** Local LLM generates response
10. **Response Display:** Show answer with source attribution

1.2 Component Overview

Component	Functionality
Document Parser	Extracts text from PDF, DOCX, XLSX, PPTX, TXT files
Text Processor	Chunks text and generates embeddings using sentence transformers
Vector Store	FAISS-based vector database for semantic search
RAG Engine	Coordinates retrieval and generation with OpenVINO LLM
Streamlit UI	Web-based user interface with conversation management

2. Document Processing Pipeline

2.1 Document Parser

The Document Parser component handles extraction of text content from multiple file formats using specialized libraries:

2.1.1 PDF Processing

- **Library:** PyMuPDF (fitz)
- **Method:** Page-by-page text extraction
- **Features:** Preserves formatting, handles multi-column layouts

2.1.2 Word Document Processing

- **Library:** python-docx
- **Content Types:** Paragraphs and tables
- **Table Handling:** Cell-by-cell extraction with pipe delimiter

2.1.3 Excel Processing

- **Library:** Pandas
- **Method:** DataFrame string conversion
- **Output:** Structured text representation of data

2.1.4 PowerPoint Processing

- **Library:** python-pptx
- **Method:** Slide-by-slide shape text extraction
- **Features:** Captures all text frames and paragraphs

2.2 Text Processing

2.2.1 Text Chunking Strategy

Uses LangChain's RecursiveCharacterTextSplitter with the following configuration:

- **Chunk Size:** 500 characters
- **Chunk Overlap:** 100 characters
- **Separators:** Double newline, single newline, space, empty string
- **Purpose:** Maintains semantic coherence while enabling granular retrieval

2.2.2 Embedding Generation

Model: all-MiniLM-L6-v2 from sentence-transformers

Key Characteristics:

- Embedding Dimension: 384
- Normalized embeddings for cosine similarity
- Fast inference on CPU
- Optimized for semantic similarity tasks
- Float32 precision for compatibility

3. Vector Store and Retrieval

3.1 FAISS Vector Database

ContextIQ uses Facebook AI Similarity Search (FAISS) for efficient vector similarity search:

- **Index Type:** IndexFlatL2 (exact search)
- **Distance Metric:** L2 (Euclidean distance)
- **Normalization:** L2 normalization before indexing and search
- **Persistence:** Save/load capability for reusability

3.2 Semantic Search Process

11. Convert user question to embedding vector
12. Normalize query embedding
13. Search FAISS index for k nearest neighbors
14. Return indices and distances of most similar chunks
15. Retrieve actual text chunks for context assembly

3.3 Knowledge Graph Integration

ContextIQ maintains an RDF-based knowledge graph using rdflib:

- Each chunk is represented as a node
- Nodes linked with RDF predicates (hasID, preview)
- Enables advanced querying and relationship tracking

- Future extensibility for complex document relationships

4. Language Model and Answer Generation

4.1 OpenVINO Phi-3 Model

The system uses Microsoft's Phi-3 model optimized for OpenVINO inference:

- **Model:** Phi-3-mini-4k-instruct
- **Quantization:** INT4 for reduced memory footprint
- **Context Window:** 4096 tokens
- **Inference Engine:** OpenVINO for CPU optimization
- **Device:** CPU (configurable)
- **Max Tokens:** 512 tokens per response

4.2 Prompt Engineering

The system uses carefully crafted prompts to ensure high-quality responses:

4.2.1 System Instructions

- Answer based ONLY on provided context
- Provide clear, well-structured answers
- Include ALL relevant items when listing
- Complete lists without truncation
- Use proper formatting (bullets, sections)
- Acknowledge when answer cannot be found
- Be concise but complete

4.2.2 Context Assembly

Retrieved chunks are formatted with clear labeling:

- Each chunk prefixed with [Context N] marker
- Chunks separated by double newlines
- Clear delineation between context and question

4.3 RAG Pipeline

The complete RAG query flow:

16. **Question Reception:** Receive user question from UI
17. **Document Retrieval:** Search vector store for k relevant chunks
18. **Context Assembly:** Format retrieved chunks into prompt
19. **LLM Inference:** Generate answer using local Phi-3 model
20. **Response Formatting:** Package answer with source attribution
21. **History Update:** Add to conversation history

5. User Interface

5.1 Streamlit Application

Built with Streamlit for rapid development and clean interface:

5.1.1 Layout Structure

- **Wide Layout:** Maximum screen utilization
- **Sidebar:** Document upload and settings
- **Main Area:** Question input and conversation history
- **Custom CSS:** Branded styling and improved UX

5.1.2 Key Features

- **File Upload:** Multi-format support with drag-and-drop
- **Retrieval Settings:** Adjustable k value (1-10 chunks)
- **Processing Indicator:** Spinner during document processing
- **Success Messages:** Feedback on chunk count
- **Clear History:** Reset conversation without reprocessing
- **Reset Document:** Complete system reset

5.2 Session State Management

Streamlit session state maintains:

- **chunks:** Processed document chunks
- **rag_engine:** Initialized RAG engine instance
- **history:** Complete Q&A conversation history
- **document_processed:** Flag to prevent reprocessing

5.3 Conversation History

Interactive history display:

- Reverse chronological order (newest first)
- Expandable question cards
- Answer display with formatting
- Retrieved context preview (500 chars)
- Source attribution for each chunk

6. Technical Stack

6.1 Core Dependencies

Library	Purpose
openvino-genai	Local LLM inference with OpenVINO optimization
streamlit	Web application framework and UI
faiss-cpu	Vector similarity search and indexing
sentence-transformers	Embedding model (all-MiniLM-L6-v2)
langchain	Text splitting and processing utilities
pymupdf	PDF text extraction
python-docx	Word document processing
python-pptx	PowerPoint text extraction
pandas	Excel spreadsheet processing
rdflib	Knowledge graph management

6.2 Model Download

The system includes an automated model downloader:

- **Source:** Hugging Face Hub
- **Model ID:** OpenVINO/Phi-3-mini-4k-instruct-int4-ov
- **Destination:** ./model/phi-3-openvino
- **Components:** Model weights, tokenizers, config files

7. Installation and Setup

7.1 System Requirements

- **Python:** 3.8 or higher
- **RAM:** Minimum 8GB (16GB recommended)
- **Storage:** ~5GB for model and dependencies
- **CPU:** Multi-core processor recommended
- **OS:** Windows, macOS, or Linux

7.2 Installation Steps

22. **Clone repository:** git clone https://github.com/satheeshbhukya/ContextIQ.git
23. **Create virtual environment:** python -m venv venv
24. **Activate environment:** source venv/bin/activate (Linux/Mac) or venv\Scripts\activate (Windows)
25. **Install dependencies:** pip install -r requirements.txt
26. **Download model:** python download_model.py
27. **Run application:** streamlit run app.py

7.3 Configuration Options

Adjustable parameters in the codebase:

- **Chunk Size:** Default 500 characters (text_processor.py)
- **Chunk Overlap:** Default 100 characters (text_processor.py)
- **Embedding Model:** all-MiniLM-L6-v2 (changeable in text_processor.py)
- **LLM Device:** CPU (changeable to GPU in rag_engine.py)
- **Max Tokens:** 512 tokens (rag_engine.py)

8. Use Cases and Applications

8.1 Research and Academia

- Query research papers for methodologies and findings
- Extract key information from literature reviews
- Compare and contrast different studies
- Search academic notes and lecture materials

8.2 Legal and Compliance

- Search contracts for specific clauses
- Query agreements and legal documents
- Extract policy information
- Privacy-compliant document analysis

8.3 Business Intelligence

- Analyze quarterly and annual reports
- Extract insights from business presentations
- Query financial data and metrics
- Search meeting minutes and documentation

8.4 Technical Documentation

- Search for specific procedures and configurations
- Extract troubleshooting steps
- Find API documentation and examples
- Query system manuals and guides

8.5 Personal Knowledge Management

- Search personal notes and documents
- Extract information from saved articles
- Organize and retrieve study materials
- Query recipe collections and cookbooks

9. Security and Privacy

9.1 Privacy-First Architecture

Complete Local Processing:

- No external API calls
- No cloud uploads
- No data telemetry
- All processing on user's machine

9.2 Data Sovereignty

- Users maintain complete control over their documents
- No third-party access to sensitive information
- Suitable for confidential business documents
- GDPR and compliance-friendly architecture

9.3 Session Management

- In-memory processing only
- Optional FAISS index persistence
- Session state clears on browser close
- No persistent logging of queries or documents

10. Performance Characteristics

10.1 Processing Speed

- **Document Parsing:** ~1-2 seconds per document
- **Chunking:** Milliseconds for typical documents
- **Embedding Generation:** ~2-5 seconds for 100 chunks
- **Vector Search:** Sub-second retrieval
- **Answer Generation:** 5-15 seconds (CPU-dependent)

10.2 Memory Usage

- **Base Application:** ~500MB
- **Phi-3 Model:** ~2GB (INT4 quantization)
- **Embedding Model:** ~100MB
- **FAISS Index:** Varies with document size
- **Total Peak:** ~3-4GB for typical usage

10.3 Scalability

- Suitable for documents up to 100+ pages
- FAISS scales to millions of vectors
- Performance degrades gracefully with size
- Can process multiple documents sequentially

11. Future Enhancements

11.1 Planned Features

- **Multi-Document Support:** Query across multiple uploaded files
- **GPU Acceleration:** Faster inference with CUDA support
- **Advanced Chunking:** Semantic and hierarchical strategies
- **Export Functionality:** Save Q&A history to file
- **Advanced Search:** Filter by metadata, date, document type

11.2 Potential Improvements

- **OCR Integration:** Handle scanned documents and images
- **Table Understanding:** Better extraction of structured data
- **Larger Models:** Support for Phi-3-medium or Llama models
- **Reranking:** Improve retrieval accuracy with cross-encoders
- **Hybrid Search:** Combine semantic and keyword search

11.3 Architecture Extensions

- **Vector Database:** Replace FAISS with Qdrant or Weaviate
- **Knowledge Graph:** Enhanced RDF schema for complex relationships
- **API Mode:** FastAPI backend for programmatic access
- **Batch Processing:** Command-line interface for automation

12. Conclusion

ContextIQ represents a significant advancement in privacy-preserving document question-answering systems. By combining state-of-the-art natural language processing techniques with a completely local architecture, it addresses the growing need for AI-powered document intelligence without compromising data security.

The system successfully demonstrates that sophisticated RAG capabilities can be achieved entirely on consumer hardware, making advanced AI accessible to users with strict privacy requirements. The modular architecture ensures maintainability and extensibility, while the Streamlit interface provides an intuitive user experience.

Key achievements include multi-format document support, efficient semantic search through FAISS, local LLM inference via OpenVINO, and an interactive conversation interface with source attribution. The integration of a knowledge graph adds a foundation for future enhancements in relationship tracking and complex querying.

ContextIQ serves as both a practical tool for document analysis and a reference implementation for building privacy-first AI applications. Its success validates the viability of local RAG systems and provides a foundation for further innovation in secure, on-device AI