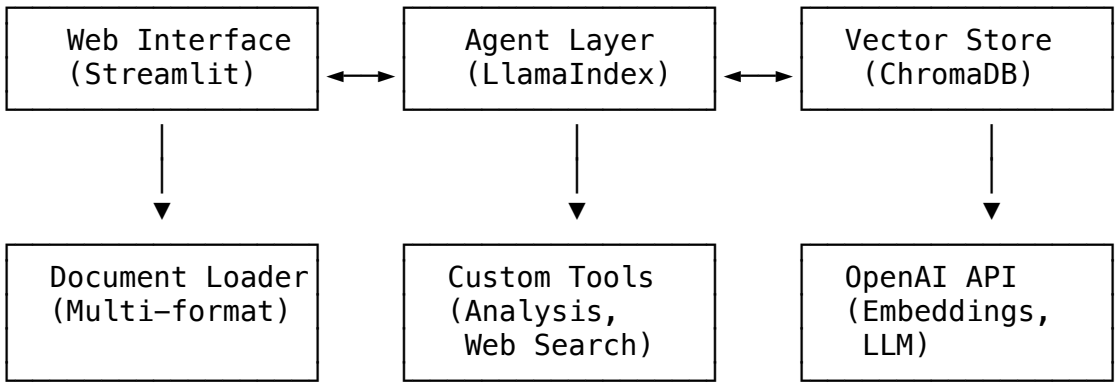
🤖 Agentic RAG with LlamaIndex

A sophisticated Retrieval-Augmented Generation (RAG) system that combines document processing, vector storage, and intelligent agents to provide context-aware question answering and document analysis capabilities.

🚀 Features

- **📚 Multi-format Document Support**: PDF, TXT, DOCX, and more
- **🔍 Intelligent Document Processing**: Automatic text extraction and chunking
- **🧠 Vector-based Retrieval**: ChromaDB-powered semantic search
- **🤖 Agentic Capabilities**: Custom tools for document analysis, web search, and more
- **🌐 Web Interface**: Beautiful Streamlit-based UI
- **📊 Analytics Dashboard**: System statistics and performance metrics
- **🛠️ Advanced Tools**: Document summarization, similarity search, and web research

🏗️ Architecture

```


```

🛠️ Tech Stack

Core Technologies
- **Python 3.11+**: Primary programming language
- **LlamaIndex 0.10.0**: RAG framework and agent orchestration
- **OpenAI API**: Language model and embeddings
- **ChromaDB**: Vector database for document storage
- **Streamlit**: Web application framework

Key Libraries
- **llama-index-core**: Core RAG functionality
- **llama-index-llms-openai**: OpenAI LLM integration

- **llama-index-agent-openai**: Agent framework
- **llama-index-vector-stores-chroma**: ChromaDB integration
- **llama-index-embeddings-openai**: OpenAI embeddings
- **chromadb**: Vector database
- **streamlit**: Web UI framework
- **pandas**: Data manipulation
- **python-dotenv**: Environment variable management

AI/ML Components
- **GPT-3.5-turbo**: Primary language model for reasoning
- **text-embedding-ada-002**: Text embedding model
- **Sentence Transformers**: Text chunking and processing

📁 Project Structure

```
Agentic RAG with LlamaIndex/
├── app.py                  # Streamlit web application
├── config.py               # Configuration management
├── requirements.txt        # Python dependencies
├── .env                    # Environment variables (API keys)
├── README.md              # This file
├── example_usage.py       # Command-line usage example
├── data/                  # Document storage
│   ├── sample_document.txt
│   └── AYUSH_CV.pdf
├── src/                   # Core application code
│   ├── __init__.py
│   ├── agent.py           # Main RAG agent
│   ├── document_loader.py # Document processing
│   ├── vector_store.py    # Vector database management
│   ├── tools.py           # Custom agent tools
│   └── utils.py           # Utility functions
├── chroma_db/             # ChromaDB storage
└── vector_store/          # Vector store cache
```

🔧 Installation

Prerequisites
- Python 3.11 or higher
- OpenAI API key
- Git

Setup Instructions

1. **Clone the repository**
   ```bash
   git clone <repository-url>
   cd "Agentic RAG with LlamaIndex"
   ```

2. **Install dependencies**
   ```bash
   ```

```
   pip install -r requirements.txt
   ```

3. **Configure environment variables**
   ```bash
   cp env_example.txt .env
   ```

   Edit `.env` file and add your OpenAI API key:
   ```
   OPENAI_API_KEY=your_openai_api_key_here
   OPENAI_MODEL=gpt-3.5-turbo
   OPENAI_EMBEDDING_MODEL=text-embedding-ada-002
   ```

🚀 Usage

Web Application (Recommended)

1. **Start the Streamlit app**
   ```bash
   streamlit run app.py
   ```

2. **Access the web interface**
   - Open your browser to `http://localhost:8501`
   - Navigate through different sections:
     - **🏠 Dashboard**: System overview
     - **📚 Document Management**: Upload and manage documents
     - **❓ Query System**: Ask questions about your documents
     - **🔧 System Tools**: Advanced analysis tools
     - **📊 Analytics**: System performance metrics

Command Line Usage

```bash
python example_usage.py
```

🔍 How It Works

1. Document Processing Pipeline

```
Document Upload → Text Extraction → Chunking → Embedding Generation
→ Vector Storage
```

- **Document Loader**: Supports multiple formats (PDF, TXT, DOCX)
- **Text Processing**: Cleans and normalizes text
- **Chunking**: Splits documents into manageable chunks (1024
tokens)

- **Embeddings**: Converts text chunks to vector representations
- **Storage**: Stores vectors in ChromaDB for fast retrieval

2. Query Processing

```
User Question → Query Embedding → Vector Search → Context Retrieval
→ LLM Response
```

- **Query Embedding**: Converts user question to vector
- **Semantic Search**: Finds most relevant document chunks
- **Context Assembly**: Combines retrieved chunks with query
- **LLM Generation**: Generates answer using GPT-3.5-turbo

3. Agentic Capabilities

The system includes custom tools for enhanced functionality:

- **Document Analysis**: Extract key information from documents
- **Similarity Search**: Find similar documents or passages
- **Web Research**: Search the internet for additional context
- **Document Summarization**: Create concise summaries
- **Metadata Extraction**: Extract structured information

⚙️ Configuration

Environment Variables

Variable	Description	Default
`OPENAI_API_KEY`	Your OpenAI API key	Required
`OPENAI_MODEL`	Language model to use	`gpt-3.5-turbo`
`OPENAI_EMBEDDING_MODEL`	Embedding model	`text-embedding-ada-002`
`CHROMA_PERSIST_DIRECTORY`	ChromaDB storage path	`./chroma_db`
`CHUNK_SIZE`	Document chunk size	`1024`
`CHUNK_OVERLAP`	Chunk overlap	`200`
`TEMPERATURE`	LLM creativity	`0.1`

Performance Tuning

- **Chunk Size**: Larger chunks (2048) for detailed analysis, smaller (512) for precise retrieval
- **Temperature**: Lower (0.1) for factual responses, higher (0.7) for creative responses
- **Top-K**: Number of similar documents to retrieve (default: 5)

🎯 Use Cases

Business Applications
- **Document Q&A**: Ask questions about company documents, policies, or reports

- **Research Assistant**: Analyze research papers and extract key insights
- **Customer Support**: Provide instant answers from knowledge bases
- **Legal Document Analysis**: Extract and summarize legal documents

Educational Applications
- **Study Assistant**: Answer questions about course materials
- **Research Tool**: Analyze academic papers and extract findings
- **Content Summarization**: Create summaries of long documents

Personal Applications
- **Personal Knowledge Base**: Organize and query personal documents
- **Reading Assistant**: Get insights from books and articles
- **Note Analysis**: Extract information from personal notes

🔒 Security & Privacy

- **API Key Management**: Secure storage using environment variables
- **Local Processing**: Document processing happens locally
- **No Data Persistence**: Documents are not stored permanently (configurable)
- **Rate Limiting**: Built-in protection against API quota exhaustion

🚨 Troubleshooting

Common Issues

1. **API Key Errors**
   - Ensure your OpenAI API key is correct and has sufficient credits
   - Check that the key is properly formatted in `.env` file

2. **Quota Exceeded**
   - Add billing information to your OpenAI account
   - Wait for quota reset (usually daily)
   - Consider using a different API key

3. **Document Processing Errors**
   - Ensure documents are in supported formats
   - Check file permissions and accessibility
   - Verify document size limits

4. **Streamlit Caching Issues**
   - Clear browser cache
   - Restart the Streamlit application
   - Check for conflicting cache decorators

Performance Optimization

- **Reduce Chunk Size**: For faster processing of large documents
- **Use GPU**: If available, configure for faster embedding generation
- **Optimize Retrieval**: Adjust Top-K and similarity thresholds

- **Cache Results**: Enable caching for frequently accessed documents

🤝 Contributing

1. Fork the repository
2. Create a feature branch
3. Make your changes
4. Add tests if applicable
5. Submit a pull request

📄 License

This project is licensed under the MIT License - see the LICENSE file for details.

🙏 Acknowledgments

- **LlamaIndex Team**: For the excellent RAG framework
- **OpenAI**: For providing the language models and embeddings
- **ChromaDB**: For the vector database solution
- **Streamlit**: For the web application framework

📞 Support

For issues, questions, or contributions:
- Create an issue on GitHub
- Check the troubleshooting section above
- Review the configuration options

Built with ❤️ using modern AI technologies