🧠 RAG Pipeline - Retrieval-Augmented Generation System

A comprehensive command-line RAG (Retrieval-Augmented Generation) system for document-based question answering using ChromaDB, LangChain, and GROQ.

🚀 Features

- 🌐 **Web Interface**: Modern Streamlit-based web UI for all operations
- 📄 **Document Ingestion**: Support for PDF, DOCX, TXT, and JSON files
- 🔍 **Vector Search**: ChromaDB for efficient similarity search
- 🤖 **AI Integration**: GROQ API for text generation
- 💬 **Interactive Queries**: Web chat interface and CLI modes
- 📊 **Statistics**: Real-time database analytics and monitoring
- 🎯 **Dual Interface**: Both web UI and comprehensive CLI tools
- ⚙️ **Configurable**: YAML-based configuration system
- 📝 **Timestamped Logs**: Detailed logging with unique timestamps

📋 Prerequisites

- Python 3.8+
- GROQ API Key

🛠️ Installation

1. **Clone the repository**

```
git clone <repository-url>
cd rag-v1
```

2. **Install dependencies**

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

3. **Set up environment variables**

```
export GROQ_API_KEY="your_groq_api_key"
```

4. **Optional: Install as package**

```
pip install -e .
```

Environment Variables and API Keys

This project uses a `.env` file to manage secrets and API keys securely. A template file named `.env.example` is provided in the project root.

Setup Instructions:

1. Copy `.env.example` to `.env` in the project root:

   ```
   cp .env.example .env
   ```

2. Open `.env` and fill in your API keys and any other required secrets. For example:

   ```
   GROQ_API_KEY=your_actual_groq_api_key_here
   # Add other keys as needed
   ```

3. **Do not commit your `.env` file to version control.**

The application will automatically load environment variables from `.env` at startup.

🎯 CLI Usage

Quick Start

```
# Initialize and test the system
python main.py init

# Show help
python main.py --help

# Show available commands
python main.py list
```

Document Ingestion

```
# Ingest documents from a directory
python main.py ingest -d ./docs

# Ingest a single file
python main.py ingest -f document.pdf

# Ingest with verbose output
python main.py ingest -d ./docs --verbose
```

Querying

```
# Ask a question
python main.py query "What is machine learning?"

# Query with verbose output (shows source details)
python main.py query "Explain neural networks" --verbose

# Interactive mode
python main.py interactive
```

Database Management

```
# Show database statistics
python main.py stats

# Clear all documents (with confirmation)
python main.py clear

# Clear without confirmation
python main.py clear --confirm
```

Configuration

```
# Use custom config file
python main.py init --config /path/to/config.yaml

# Skip test query during initialization
python main.py init --no-test
```

🌐 Web Interface (Streamlit)

Launch the comprehensive web-based interface for a user-friendly experience:

```
# Start the Streamlit web app
streamlit run app.py

# Or with custom port
streamlit run app.py --server.port 8502
```

Web Interface Features

- 🏠 **Dashboard**: System overview and quick actions
- 🚀 **Initialize**: Web-based system initialization

- 📑 **Ingest Documents**:
  - Upload files directly through the browser
  - Specify directory paths
  - Drag-and-drop support for multiple files
- 💬 **Chat Interface**: Interactive conversational AI with chat history
- ❓ **Single Query**: Detailed query interface with source analysis
- 📊 **Statistics**: Real-time database analytics and visualizations
- 🗑️ **Clear Database**: Safe database clearing with confirmations
- 📋 **System Info**: Configuration and system status overview

The web interface provides all CLI functionality through an intuitive, modern UI accessible at `http://localhost:8501`.

🎮 Interactive Mode (CLI)

Start CLI interactive mode for conversational queries:

```
python main.py interactive
```

Interactive commands:

- `/stats` - Show database statistics
- `/help` - Show help
- `/quit` - Exit interactive mode

📊 Examples

Basic Workflow

```
# 1. Initialize the system
python main.py init

# 2. Add documents
python main.py ingest -d ./data/raw

# 3. Query the system
python main.py query "What are the main topics in the documents?"

# 4. Check statistics
python main.py stats
```

Advanced Usage

```
# Verbose ingestion with timing
python main.py ingest -d ./research_papers --verbose
```

```
# Query with source details
python main.py query "Explain the methodology" --verbose --max-results 10

# Interactive session
python main.py interactive
```

🔧 Configuration

Edit `config/config.yaml` to customize:

```
# Logging Configuration
logging:
  level: "INFO"
  format: "%(asctime)s - %(levelname)s - %(name)s:%(lineno)d - %(message)s"
  path: "./logs"

# LLM Configuration
llm:
  model: "llama-3.1-8b-instant"
  temperature: 0.7
  max_tokens: 1000

# Vector Database Configuration
vector_db:
  path: "./data/vectors"
  collection_name: "documents"
```

🗀 Project Structure

```
rag-v1/
├── main.py                  # Enhanced CLI entry point
├── app.py                   # Streamlit web interface
├── setup.py                # Package setup (legacy)
├── pyproject.toml          # Modern package configuration
├── requirements.txt        # Dependencies
├── rag.bat                # Windows CLI launcher
├── rag.sh                 # Linux/Mac CLI launcher
├── src/
│   ├── utils/
│   │   ├── init_manager.py    # Logging initialization
│   │   ├── log_manager.py     # Log management utilities
│   │   └── config_loader.py   # Configuration management
│   ├── ingestion/
│   │   └── document_loader.py # Document processing
│   └── rag_pipeline.py        # Core RAG functionality
├── config/
│   └── config.yaml            # System configuration
├── data/
│   ├── raw/                   # Input documents
│   ├── processed/          # Processed documents
│   └── vectors/            # Vector database
├── logs/                   # Timestamped log files
└── docs/                   # Documentation
```

🔍 Supported File Formats

- **PDF** (.pdf) - Extracted using PyPDF2
- **Word** (.docx) - Processed with python-docx
- **Text** (.txt) - Plain text files
- **JSON** (.json) - Structured data files

📈 Log Files

The system creates timestamped log files in the format:

- `log_YYMMDD_HHMM.log` (e.g., `log_250723_1430.log`)
- New log file created for each session
- Configurable via `config.yaml`

🧪 Testing

```
# Run with test data
python main.py init

# Test individual components
python main.py ingest -f ./data/raw/sample.pdf
python main.py query "Test question"
python main.py stats
```

🚦 Troubleshooting

Common Issues

1. **Missing GROQ API Key**

   ```
   export GROQ_API_KEY="your_api_key_here"
   ```

2. **Dependencies not installed**

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

3. **No documents found**

```
python main.py ingest -d ./your_documents_directory
```

4. **Permission errors**
   - Ensure write permissions for `logs/` and `data/` directories

Debug Mode

```
# Enable verbose output
python main.py --verbose <command>

# Check logs
tail -f logs/log_*.log
```

📄 License

MIT License - see LICENSE file for details.

🤝 Contributing

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

📞 Support

- 📖 Documentation: Check the `docs/` directory
- 🐛 Issues: Report bugs via GitHub issues
- 💬 Questions: Use GitHub discussions