Sarkari Sathi

RAG-based Government Information Chatbot

Project Documentation
Generated on: December 03, 2025 at 07:03 AM

Table of Contents

1. Project Overview

Sarkari Sathi is a Retrieval-Augmented Generation (RAG) chatbot designed to provide citizens with accurate, up-to-date information about government services. The system uses semantic vector search to retrieve relevant information from government documents and provides structured, source-grounded answers to user queries.

Key Objectives:

• Enable 24/7 access to government service information
• Support multilingual queries (English and Nepali)
• Provide accurate, source-attributed answers
• Handle diverse document formats (PDF, DOCX, tables)
• Operate cost-effectively without external API dependencies

2. Technologies and Tools

Technology/Tool	Version	Purpose
Python	3.13	Core programming language
FastAPI	0.118.2	Web framework for REST API backend
Uvicorn	0.37.0	ASGI server for FastAPI
Pinecone	7.3.0	Vector database for semantic search
Sentence Transformers	5.1.1	Local embedding model (all-MiniLM-L6-v2)
PyTorch	2.8.0	Deep learning framework for embeddings
pdfplumber	0.11.8	PDF text extraction
python-docx	1.2.0	DOCX file processing
python-dotenv	1.1.1	Environment variable management
Pydantic	2.12.0	Data validation and settings
Jinja2	3.1.6	HTML templating
NumPy	2.3.3	Numerical computations
Transformers	4.57.0	Hugging Face transformers library
Hugging Face Hub	0.35.3	Model downloading and management

Embedding Model:

- **Model:** sentence-transformers/all-MiniLM-L6-v2
- **Dimensions:** 384
- **Purpose:** Converts text (queries and documents) into dense vector representations
- **Advantages:** Fast, lightweight, free to use, good semantic understanding

3. API Keys and Configuration

Required API Keys:

Service	Environment Variable	Purpose	Where to Get
Pinecone	PINECONE_API_KEY	Access to vector database	https://app.pinecone.io
Pinecone	PINECONE_ENVIRONMENT	Pinecone environment/region	Pinecone dashboard
OpenAI	OPENAI_API_KEY	Optional: for paid version	https://platform.openai.com

Configuration File (config.py):

- PINECONE_API_KEY: Your Pinecone API key
- PINECONE_ENVIRONMENT: Your Pinecone environment
- PINECONE_INDEX_NAME: Name of the index (default: 'rag-chatbot')
- FREE_INDEX_NAME: Name for free version index (default: 'rag-chatbot-free')
- OPENAI_API_KEY: OpenAI API key (optional, for paid version)
- APP_HOST: Server host (default: '0.0.0.0')
- APP_PORT: Server port (default: 8000)
- EMBEDDING_MODEL: Embedding model name (default: 'text-embedding-ada-002')
- FREE_EMBEDDING_MODEL: Free model name (default: 'all-MiniLM-L6-v2')
- TOP_K_RESULTS: Number of documents to retrieve (default: 5)
- SIMILARITY_THRESHOLD: Minimum similarity score (default: 0.3)

4. Project Structure

File/Directory	Purpose
main.py	FastAPI application entry point
main_free.py	Free version (no OpenAI) entry point
rag_chatbot_free.py	RAG chatbot logic with rule-based generation
vector_store_free.py	Pinecone vector store operations
config.py	Configuration and environment variables
sample_data.py	Sample documents for testing
setup_knowledge_base_free.py	Initialize knowledge base with sample data
add_chandragiri_pdf.py	Add PDF documents to Pinecone
add_nepal_docx.py	Add DOCX files to Pinecone
templates/index.html	Web chat interface
requirements.txt	Python dependencies
.env	Environment variables (API keys)
.venv/	Python virtual environment

5. Core Components

5.1 Vector Store (vector_store_free.py)

Purpose: Manages all interactions with Pinecone vector database

Key Methods:
• **create_embedding(text):** Converts text to 384-dimensional vector using MiniLM-L6-v2
• **add_documents(documents):** Embeds and stores documents in Pinecone
• **search_similar(query, top_k):** Finds most similar documents using cosine similarity
• **get_index_stats():** Returns index statistics (vector count, dimensions)

Technical Details:
• Uses sentence-transformers for local embeddings (no API costs)
• Handles metadata size limits (40KB per vector)
• Supports chunking large documents automatically

5.2 RAG Chatbot (rag_chatbot_free.py)

Purpose: Orchestrates the RAG pipeline: retrieval + generation

Key Methods:
• **chat(query):** Main entry point - processes user queries
• **retrieve_relevant_documents(query):** Searches Pinecone for relevant chunks
• **generate_response(query, context):** Rule-based answer extraction
• **_handle_nepali_query():** Special handling for Nepali language queries

Response Extraction Types:
• Fee/Cost information
• Required documents (lists)
• Process/Steps
• Time/Duration
• Services available
• Location information
• Nepal-specific facts

5.3 FastAPI Backend (main_free.py)

Purpose: RESTful API server for the chatbot

Endpoints:
• **GET /:** Serves the web chat interface
• **POST /chat:** Processes chat queries, returns responses
• **POST /add-documents:** Adds new documents to knowledge base
• **GET /stats:** Returns knowledge base statistics
• **GET /health:** Health check endpoint

6. Implementation Details

6.1 Document Processing Pipeline

1. **Ingestion:** Documents are loaded from various sources (PDF, DOCX, text files)
2. **Text Extraction:** pdfplumber extracts text from PDFs, python-docx from DOCX files
3. **Cleaning:** Remove headers, footers, extra spaces, normalize formatting
4. **Chunking:** Split large documents into smaller chunks (max 3000 chars) to avoid metadata limits
5. **Embedding:** Each chunk is converted to 384-dimensional vector using MiniLM-L6-v2
6. **Indexing:** Vectors stored in Pinecone with metadata (title, source, category, content preview)
7. **Metadata Limiting:** Content in metadata limited to 2000 chars to stay under 40KB limit

6.2 Query Processing Flow

1. **Query Reception:** User query received via FastAPI /chat endpoint
2. **Preprocessing:** Query is preprocessed (Nepali keyword expansion, normalization)
3. **Embedding:** Query converted to vector using same embedding model
4. **Vector Search:** Cosine similarity search in Pinecone (top-k=5, threshold=0.3)
5. **Context Assembly:** Retrieved document chunks combined into context
6. **Query Classification:** System identifies query type (fee, documents, process, etc.)
7. **Answer Extraction:** Rule-based extractor pulls relevant information
8. **Response Formatting:** Answer formatted with bullets, citations, source attribution
9. **Delivery:** JSON response sent to frontend for display

7. Features and Capabilities

7.1 Multilingual Support

• **Nepali Language Detection:** Detects Devanagari script in queries
• **Keyword Mapping:** Maps Nepali question words to extraction methods
• **Query Preprocessing:** Adds English translations to Nepali queries for better embedding
• **Supported Nepali Terms:** ■■■■ (where), ■■■■■ (when), ■■■ (how much), ■■■■ (how), ■■ (what), ■■■■■■ (documents), ■■■■■ (fee), ■■■■ (service), etc.

7.2 Document Types Supported

• **PDF Files:** Text extraction using pdfplumber, table extraction support
• **DOCX Files:** Full text extraction using python-docx
• **Text Files:** Direct text processing
• **Tabular Data:** Tables converted to structured text format
• **Mixed Content:** Handles documents with both text and tables

7.3 Answer Extraction Capabilities

• **Fee Information:** Extracts cost/fee details (including '■■■■■■■■' - free)
• **Document Lists:** Extracts numbered lists of required documents
• **Process Steps:** Identifies and extracts procedure information
• **Time Information:** Extracts duration, deadlines, service times
• **Service Lists:** Identifies available services from context
• **Location Data:** Extracts geographical and location information
• **General Summaries:** Fallback to general information extraction

8. Data Flow

8.1 Document Ingestion Flow

Document → Text Extraction → Cleaning → Chunking → Embedding → Pinecone Storage

8.2 Query Processing Flow

User Query → Preprocessing → Embedding → Pinecone Search → Context Retrieval → Answer Extraction → Response Formatting → User

8.3 Current Knowledge Base

Document Type	Count	Source
Sample Documents	10	sample_data.py
Nepal Information	1	nepal.docx
Citizen Charter (Basic)	1	Manual entry
Citizen Charter (Complete PDF)	151 chunks	[redacted].pdf
Total Vectors	163	Pinecone index: rag-chatbot-free

9. Deployment Information

9.1 Setup Steps

1. Create virtual environment: **python -m venv .venv**
2. Activate virtual environment: **source .venv/bin/activate** (Mac/Linux)
3. Install dependencies: **pip install -r requirements.txt**
4. Create .env file with API keys (PINECONE_API_KEY, PINECONE_ENVIRONMENT)
5. Initialize knowledge base: **python setup_knowledge_base_free.py**
6. Start server: **python main_free.py**
7. Access web interface: **http://localhost:8000**

9.2 Adding Documents

• **PDF Files:** python add_chandragiri_pdf.py [path_to_pdf]
• **DOCX Files:** python add_nepal_docx.py
• **Via API:** POST /add-documents with JSON payload
• **Interactive:** python interactive_add_documents.py

9.3 System Requirements

• **Python:** 3.8 or higher
• **RAM:** Minimum 4GB (8GB recommended for embedding model)
• **Storage:** ~500MB for dependencies and models
• **Internet:** Required for Pinecone API and initial model download
• **OS:** macOS, Linux, or Windows

9.4 Cost Analysis

• **Embedding Model:** Free (local, no API calls)
• **Pinecone:** Free tier available (limited to 1 index, 100K vectors)
• **Response Generation:** Free (rule-based, no LLM API calls)
• **Total Operating Cost:** $0 (within free tier limits)

Note: For production scale, Pinecone paid plans start at $70/month

10. Technical Specifications

10.1 Vector Database Configuration

- **Index Name:** rag-chatbot-free
- **Dimensions:** 384 (MiniLM-L6-v2)
- **Metric:** Cosine Similarity
- **Cloud Provider:** AWS
- **Region:** us-east-1
- **Metadata Limit:** 40KB per vector
- **Current Vectors:** 163

10.2 API Endpoints

Method	Endpoint	Purpose	Request/Response
GET	/	Web UI	Returns HTML chat interface
POST	/chat	Chat query	{"message": "query"} → {"response": "...", "sources": [...]}
POST	/add-documents	Add docs	{"documents": [...]} → {"success": true}
GET	/stats	Statistics	Returns index stats
GET	/health	Health check	Returns system status

11. Known Issues and Limitations

11.1 Current Limitations

• **Multilingual Support:** Embedding model is English-centric, affecting Nepali query accuracy
• **Response Quality:** Rule-based extraction may miss nuanced queries
• **Context Window:** Limited to top-5 retrieved documents
• **No Conversation Memory:** Each query is independent, no context retention
• **Metadata Size:** Large documents must be chunked to fit 40KB metadata limit
• **No OCR Support:** Scanned PDFs with images not yet supported
• **Similarity Threshold:** Fixed at 0.3, may need tuning for different query types

11.2 Future Improvements

• Integrate multilingual embedding models (multilingual-MiniLM)
• Add OCR support for scanned documents
• Implement conversation memory and context retention
• Add hybrid search (keyword + semantic)
• Integrate local LLM for better response generation
• Add user authentication and query history
• Implement caching for frequently asked questions