International Journal of Advanced Scientific Research in Engineering Trends and Technology

Volume 1 Issue 5, Nov 2026
www.ijengt.com

A Document-Grounded Question Answering Assistant for Government Information: The CivicGuide AI Implementation

Sheetal DN CR Pruthvik, D Samiksha, Teju RS, Deekshitha KP, and Arshil MD
Department of Artificial Intelligence and Machine Learning
Malnad College of Engineering, Hassan, India
dn@mcehassan.ac.in

***Abstract*—** Government-related answers are often contained in long notifications and reference documents rather than in one easy-to-use place. This paper reports the implementation of a document-based question answering assistant developed within the CivicGuide AI project. The standalone application accepts PDF and TXT files, extracts their text, and divides it into overlapping passages. The all-MiniLM-L6-v2 Sentence Transformers model converts those passages into vectors, which are stored in a persistent ChromaDB collection. For each question, the same encoder is used to represent the query; the configured retriever requests up to eight passages and applies a distance threshold of 1.3. The selected text is then supplied to Cohere Command R to prepare a response. The backend exposes its functions through FastAPI. Example queries about the UPSC Civil Services Examination and railway recruitment show that the query-to-answer path is operational. They do not, however, establish benchmark accuracy or response-time performance. The RAG application described here remains separate from the main CivicGuide AI website chatbot.

***Index Terms*—** CivicGuide AI, document question answering, government information, ChromaDB, Sentence Transformers, Cohere Command R, retrieval-augmented generation.

I. INTRODUCTION

A person checking an examination notification may want one detail: the age requirement, the documents to carry, or the order of the selection stages. The answer may be present in an official PDF, but finding the relevant paragraph can require opening several files and searching through pages of text. The same difficulty appears when users look for scheme conditions, scholarship rules, certificate procedures, and other public-service information.

A conversational interface can make this task less demanding, but a language model should not be treated as a replacement for the documents that define a government rule. A response generated only from a model's learned parameters may omit a condition or present information that is no longer current. Retrieval-augmented generation (RAG) offers a document-centred alternative: the application first locates passages from a chosen collection and then gives those passages to a language model as context for its response [1].

The CivicGuide AI project includes a separate assistant built around this idea. Its knowledge base is made from PDF and plain-text files. Sentence embeddings and a vector store are used to find passages related to a question, and Cohere Command R is used to draft the response. The work reported here concerns that RAG application only. It does not describe an integration with the primary CivicGuide AI website chatbot.

The contribution of this paper is an implementation account rather than a new retrieval algorithm. It records the file-processing settings, embedding and storage components, retrieval configuration, API structure, and the behaviour observed in sample examination queries. It also identifies what remains untested so that the current demonstration is not mistaken for a measured evaluation.

II. APPLICATION SCOPE AND DATA FLOW

The application has two operating paths. The first prepares the searchable collection from source documents. It is run when the collection is created or refreshed. The second path handles a user question: it represents the query, searches the stored vectors, builds a context from the retrieved passages, and requests a response from Cohere Command R.

These paths meet at the persistent ChromaDB collection named civicguide_documents. Storing the vectors allows later questions to reuse the prepared collection instead of repeating document extraction and embedding generation for every query. In the current version, the RAG service is exposed through the /ask, /search, and /health endpoints. The website chatbot uses its own /api/chat endpoint; that route does not call the RAG backend in the implementation examined.

This boundary matters when interpreting the system. The paper evaluates the standalone RAG application's demonstrated workflow. It does not claim that a visitor using the main website chatbot is already receiving answers from this document collection.

III. PREPARING THE DOCUMENT COLLECTION

A. File input and text extraction

The loader accepts PDF and TXT files. PDF text is read with PyPDF, while plain-text files are decoded as UTF-8 [7]. The extracted text becomes the input to the next stage. This means the quality of the stored collection depends partly on what the extraction step can read from each file. A scanned page, an unusual layout, or a document with poorly encoded text may need separate

ISSN : 3139 – 1303 ©IJENGT: All Rights are Reserved

handling; the current description does not establish that every possible PDF layout is processed equally well.

B. Passage segmentation

Rather than embedding an entire document as one unit, the application cuts the extracted text into shorter passages. The configured passage length is 1,000 characters. Adjacent passages share 200 characters. That overlap is intended to retain some of the wording around a split point, where a sentence or explanation might otherwise be separated between two stored items. These are the values used by the current implementation; they have not been shown to be optimal through a comparative experiment.

TABLE I. CURRENT DOCUMENT AND RETRIEVAL SETTINGS

Setting	Value in the implementation
Accepted files	PDF and TXT
PDF text extraction	PyPDF
TXT decoding	UTF-8
Passage length	1,000 characters
Shared text between passages	200 characters
Embedding model	all-MiniLM-L6-v2
Vector store	ChromaDB; civicguide_documents
Retrieved passages	Up to 8
Distance threshold	1.3 (configured)

The chunking step organises text for retrieval; it does not verify the authority or publication date of a source. The collection must therefore be populated with documents that the project team considers suitable for the intended use. When a department changes a notification or eligibility rule, the corresponding material should also be replaced or refreshed.

IV. EMBEDDINGS AND PASSAGE RETRIEVAL

After segmentation, the application uses the all-MiniLM-L6-v2 model through Sentence Transformers to encode each passage as a numerical vector [2], [5]. A vector provides a compact representation of the passage for similarity-based search. This lets the retriever find text that is related in meaning to a question, even when the question and the source passage do not use exactly the same words.

The vectors and their associated document information are written to the persistent ChromaDB collection civicguide_documents [4]. At query time, the same embedding model converts the user's question into a vector. ChromaDB is then asked to return the closest stored passages. The present configuration requests up to eight results and applies a distance threshold of 1.3.

The threshold and result count should be read as implementation settings, not as evidence of retrieval quality. The available project material does not specify a validated threshold-selection experiment or report a comparison across different values. A future evaluation should check whether the returned passages actually contain the facts needed to answer a representative set of questions.

V. RESPONSE CONSTRUCTION WITH COHERE

The retrieved passages are combined with the user's question and sent to the Cohere Command R model through the Cohere API [3]. The prompt directs the model to use the supplied material and avoid adding details that the retrieved context does not support. The generated text is returned to the RAG application and displayed to the user.

For a typical request, the backend performs the following operations:

1) accept the question from the RAG interface;
2) encode the question with all-MiniLM-L6-v2;
3) request relevant passages from ChromaDB using the configured retrieval settings;
4) combine the selected passages into the context supplied to Cohere Command R;
5) receive the model response and return it to the application interface.

The prompt instruction is a safeguard, not a proof of correctness. If the indexed documents are incomplete, outdated, or unrelated to the question, the model may still produce an inadequate response. For this reason, a response should be treated as document-assisted guidance and checked against the underlying official publication when a decision depends on it.

VI. BACKEND ORGANISATION

The service is written in Python and uses FastAPI to expose its operations [6]. The code is separated by responsibility so that file handling, vector operations, retrieval, and response generation are not all placed in one module. The implementation files listed in the project are:

1) src/api.py, which defines the FastAPI routes;
2) src/document_loader.py, which reads the supported document files;
3) src/chunking.py, which creates the overlapping text passages;
4) src/embeddings.py, which creates vectors from text;
5) src/vector_store.py, which manages ChromaDB data;
6) src/retriever.py, which selects passages for a question;
7) src/generation.py, which calls Cohere for response generation; and
8) src/rag_pipeline.py, which connects the stages into one request flow.

The /ask route supports question answering, /search exposes document retrieval, and /health provides a service-status check. These routes belong to the separate RAG service. No connection between this backend and the main website's /api/chat route is claimed in this paper.

VII. FUNCTIONAL WALKTHROUGHS AND OBSERVATIONS

A. UPSC Civil Services question

One trial asked the assistant about eligibility for the UPSC Civil Services Examination, including age limits and the examination structure relevant to IAS and IPS aspirants. The displayed response brought together details about nationality, educational qualifications, age conditions, and the stages of the examination. The example shows the system turning a question into a structured response using the document-retrieval workflow.

ISSN : 3139 – 1303
©IJENGT: All Rights are Reserved

B. Railway recruitment question

A second trial covered RRB NTPC and Group D. The question requested the selection stages, syllabus, and eligibility conditions. The displayed answer outlined the Computer Based Test, its duration and question structure, the subject-wise distribution, and later selection requirements. The example indicates that the same interface can be used for more than one examination topic.

C. What these trials establish

The observed responses were organised with headings, bullet points, and numbered material. The interface also showed relevance indicators. Those indicators are produced by the application and should not be interpreted as measured answer accuracy. The available trials establish that the application can accept questions, retrieve text, and display generated replies. They do not measure factual correctness across a large question set, retrieval recall, citation quality, or response time under load.

VIII. LIMITATIONS AND WORK AHEAD

The present implementation is an operational prototype, and its limitations should be considered when interpreting the examples. First, the quality of an answer is tied to the material indexed in ChromaDB. Missing documents or old notifications can lead to answers that are incomplete or no longer applicable. Second, text extraction and chunk boundaries can affect which information is available to the retriever. Third, a language model may misread or overstate evidence even when context is supplied. Finally, the standalone RAG application has not yet been connected to the main CivicGuide AI website chatbot.

The next development stage should begin with the knowledge base. Official documents can be catalogued with their issuing authority, publication date, and source location, making it easier to review and replace outdated entries. The team can then test different passage sizes, overlap values, retrieval counts, and distance cutoffs using questions with manually verified answers. Retrieval quality and answer quality should be measured separately, because finding a relevant passage does not guarantee that the final response represents it correctly.

Other planned improvements include adding support for more Indian languages, creating a routine for refreshing documents, and integrating the RAG service with the website after the API and interface have been checked. A larger evaluation should report the test questions, reference answers, evaluation method, and observed results. Until such testing is completed, numerical claims about accuracy, reliability, or latency should be avoided.

IX. CONCLUSION

This paper has documented the standalone document-questioning assistant developed for the CivicGuide AI project. The application turns PDF and TXT material into overlapping text passages, represents those passages with all-MiniLM-L6-v2 embeddings, stores them in ChromaDB, and uses retrieved context with Cohere Command R to prepare an answer. FastAPI provides the service endpoints used by the application.

The UPSC and railway-examination walkthroughs show that the complete question-to-response path works for the examples tested. They do not establish how accurately the assistant performs across government information in general. The RAG service also remains separate from the main website chatbot in the version described. Future work will focus on maintaining a reliable document collection, evaluating retrieval and answer quality with verified questions, and completing the planned website integration.

REFERENCES

[1] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” in Advances in Neural Information Processing Systems, vol. 33, pp. 9459–9474, 2020.

[2] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,” in Proc. 2019 Conf. Empirical Methods in Natural Language Processing and 9th Int. Joint Conf. Natural Language Processing (EMNLP-IJCNLP), pp. 3982–3992, 2019, doi: 10.18653/v1/D19-1410.

[3] Cohere, “Command R Model Documentation.” [Online]. Available: https://docs.cohere.com/docs/command-r. [Accessed: Nov. 19, 2026].

[4] Chroma, “Chroma Documentation.” [Online]. Available: https://docs.trychroma.com/. [Accessed: Nov. 19, 2026].

[5] Sentence Transformers, “Sentence Transformers Documentation.” [Online]. Available: https://www.sbert.net/. [Accessed: Nov. 19, 2026].

[6] FastAPI, “FastAPI Documentation.” [Online]. Available: https://fastapi.tiangolo.com/. [Accessed: Nov. 19, 2026].

[7] PyPDF, “PyPDF Documentation.” [Online]. Available: https://pypdf.readthedocs.io/. [Accessed: Nov. 19, 2026].

ISSN : 3139 – 1303
©IJENGT: All Rights are Reserved