WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

RAG Powered Website Chatbot

Jenifer R[#1], Kavishna S[#2], Karishni B[#3], Kaviya M[#4], Kavi Bharathi R[#5]

[#1] Assistant Professor, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.
[#2] Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.
[#3] Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.
[#4] Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.
[#5] Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.

ABSTRACT

Large language models answer fluently but can state claims that no source supports, and keyword search returns documents rather than answers. This paper presents RAVEN (Retrieval-Augmented Vectorized Engine for websites), a retrieval-augmented generation system that crawls a website, indexes its text by meaning, and answers questions using only the retrieved passages, with the source pages shown for every answer. The system combines a Playwright-based crawler, recursive character chunking, a locally executed 384-dimensional sentence-embedding model with a ChromaDB vector index, an open-weight language model (gpt-oss-120b) served through Groq, a GraphQL API, and a Streamlit interface. A normalised-question answer cache backed by Redis, with an automatic in-memory fallback, removes repeated retrieval and generation cost, and Docker Compose packages the services for one-command deployment. On a 20-page Wikipedia corpus on skin care, removing citation markup reduced the index from 1,769 to 898 chunks and eliminated bibliographic fragments from the top retrieved results. In preliminary qualitative tests, answers were grounded in the retrieved text and an out-of-scope question was declined; in single observations, a repeated question was answered from cache in under one millisecond, compared with 1.2 to 2.8 seconds for a newly generated answer. We discuss engineering lessons and limitations, including the absence of a quantitative accuracy benchmark, and outline future work.

Keywords: retrieval-augmented generation; large language models; semantic search; vector database; GraphQL; response caching; containerization

1. Introduction

Large language models (LLMs) produce fluent answers but can state claims that are not supported by any source, a failure commonly called hallucination [1]. Keyword search avoids invented text, but it returns documents rather than answers and matches words rather than meaning. Retrieval-augmented generation (RAG) combines the two: relevant passages are retrieved from a trusted collection and supplied to the model, which is instructed to answer from them [2], [3]. The approach lets a small deployment answer questions about a specific body of content, such as an organisation's website, without retraining a model.

Building a RAG assistant that people can actually run involves more than retrieval and generation. Content must be collected from live websites, cleaned and indexed; the service must expose a stable interface; repeated questions should not repeat the full retrieval and generation cost; and the system should start reliably on another machine. These engineering concerns are rarely reported in detail, yet they decide whether a prototype is usable.

This paper presents RAVEN (Retrieval-Augmented Vectorized Engine for websites), an end-to-end system built from open-source components and a free-tier hosted model. Its contributions are:

- An ingestion pipeline that crawls a website with a headless browser, removes navigation and citation markup, de-duplicates pages, and splits text into overlapping chunks.
- A retrieval and generation pipeline that answers only from retrieved passages, shows the source pages, and declines questions the corpus cannot answer.
- A GraphQL API and chat interface that report, for every answer, whether it came from cache and how long it took.
- A normalised-question answer cache (Redis with an in-memory fallback) and a Docker Compose deployment that starts the whole system with one command.
- A preliminary evaluation and a set of engineering lessons from building and porting the system.

The remainder of the paper is organised as follows. Section 2 reviews related work, Section 3 describes the design, Section 4 lists the implementation, Section 5 reports results, Section 6 discusses lessons and limitations, and Section 7 concludes.

2. Related Work

Retrieval-augmented generation. Lewis et al. introduced RAG, which conditions a sequence-to-sequence generator on passages retrieved from a dense index and showed gains on knowledge-intensive tasks [2]. A later survey organises RAG into naive, advanced and modular pipelines and reviews retrieval, generation and evaluation techniques [3]. Hallucination in generated text, its causes and its mitigation have been surveyed by Ji et al. [1]. RAVEN uses the naive pipeline (retrieve, then generate), which is simple to deploy and easy to inspect.

Dense retrieval and sentence embeddings. Dense passage retrieval replaces sparse term matching with learned vector representations and improves open-domain question answering [4]. Sentence-BERT produces sentence embeddings that can be compared with cosine similarity [5], and MiniLM compresses large transformers into small, fast models [6]. Approximate nearest-neighbour search over such vectors is commonly performed with hierarchical navigable small-world (HNSW) graphs [7], which RAVEN uses through ChromaDB.

3. System Design

3.1 Architecture overview

RAVEN processes a website in two phases. In the offline phase, a crawler collects pages, a preprocessing stage cleans and chunks the text, and an embedding stage stores one vector per chunk in a vector index. In the online phase, a question is embedded with the same model, the closest chunks are retrieved, and a language model writes an answer from them; the answer and its sources are returned through an API and shown in a chat interface. This separation keeps the costly one-time work of collecting and indexing content apart from the fast per-question work of retrieval and answer generation.

3.2 Data acquisition

The crawler uses Playwright [8], which drives a headless Chromium browser so that JavaScript-rendered pages are captured, and BeautifulSoup to parse the HTML. Starting from a seed URL, it performs a breadth-first traversal of same-domain links up to a configurable depth and page limit, with a short delay between requests. Links are normalised (fragments and trailing slashes removed, a leading "www" ignored when comparing domains) and non-HTML resources are skipped. For wiki-style sites the crawler also excludes non-article namespaces (talk, category, special and help pages) and edit or print URLs. During parsing, navigation, footers, scripts, forms and citation markup (footnote markers, reference lists and bibliographic sections) are removed, so that only readable page text and simple tables remain. Each page is stored as a record containing its URL, depth, title, text and tables.

3.3 Preprocessing and chunking

Preprocessing removes residual boilerplate lines (for example cookie notices and copyright lines) and discards pages whose normalised text is identical to that of an earlier page, detected with a SHA-256 hash. The text is then split with a recursive character splitter that prefers paragraph breaks, then line breaks, sentence ends and word boundaries before cutting inside a word. Chunks contain at most 500 characters (about 125 tokens, assuming roughly four characters per token) with a 50-character overlap, and chunks shorter than 30 characters are discarded. Each chunk keeps the URL and title of its page, which enables source attribution later.

3.4 Embedding and vector index

Each chunk is converted to a 384-dimensional vector by all-MiniLM-L6-v2, a six-layer sentence-embedding model in the MiniLM family [5], [6]. The model runs locally through the ONNX runtime bundled with ChromaDB [9], so embedding needs no API key and incurs no per-request cost. Vectors, texts and metadata are stored in a persistent ChromaDB collection that uses an HNSW index [7] with cosine distance, $d(q, c) = 1 - (q \cdot c) / (\|q\| \, \|c\|)$. Indexing uses upsert semantics keyed by chunk identifier, so rebuilding the index is idempotent.

3.5 Retrieval and grounded generation

At question time the query is embedded with the same model and the k closest chunks are retrieved ($k = 4$ by default). The prompt has a system message and a user message. The system message instructs the model to (i) use only the supplied context, (ii) say clearly when the context is insufficient instead of guessing, (iii) answer concisely in plain language, and (iv) not refer to "the context" in its reply. The user message lists the retrieved chunks, each labelled with its source title and truncated to 800 characters, followed by the question. The prompt is sent to gpt-oss-120b [10], an open-weight mixture-of-experts reasoning model with about 117 billion parameters in total and 5.1 billion active per token, served through Groq's hosted API [11] with temperature 0.3 and a limit of 600 output tokens. The answer is returned with the de-duplicated list of source pages. When the answer states that the information is not available, the interface hides the source list so that irrelevant citations do not appear beside a refusal.

3.6 API and user interface

The backend exposes a single GraphQL endpoint [12] implemented with FastAPI [13] and Strawberry [14]. Clients request exactly the fields they need. The main query, chat(question, topK), returns the answer, its sources, a flag indicating whether the answer was served from cache, and the server-side latency in milliseconds; auxiliary queries report the number of indexed chunks, cache statistics and service health. Inputs are validated (non-empty question, $1 \leq \text{topK} \leq 20$). The chat interface is built with Streamlit [15] and runs as a separate process that calls the API over HTTP, so the frontend can be replaced without changing the backend. It shows the conversation, a collapsible source list under each answer, a per-answer timing label, and a sidebar with the index size and the cache hit and miss counts.

3.7 Answer cache

Before running retrieval, the API looks the question up in a cache. The key is a SHA-256 hash of the normalised question (lower-cased, whitespace collapsed, trailing punctuation removed) together with the value of k, so that "What causes eczema?" and "what causes eczema" share an entry while a different k does not. Entries expire after one hour. The primary store is Redis [16], configured with a 128 MB memory cap and least-recently-used eviction. If Redis is unreachable at start-up or fails at run time, the cache falls back to an in-memory store capped at 500 entries, so the service never fails because of its cache. Only successfully generated answers are cached, and a cache hit skips both retrieval and the language-model call.

3.8 Containerized deployment

The system is packaged with Docker [17] and Docker Compose as three services: Redis, the backend and the chat interface. The backend and the interface share one image built from a slim Python base, so the runtime image contains only serving dependencies; scraping tools are excluded because crawling is an offline step. Health checks order start-up: the interface waits for a healthy backend, which waits for a healthy Redis. The vector index is stored in a named volume, the chunk file is mounted read-only from the host, and the embedding-model files are mounted so that the container never has to download them. A single command starts the system.

4. Implementation

RAVEN is built in Python from open-source components. Web pages are crawled with Playwright, which drives a headless Chromium browser, and parsed with BeautifulSoup. Text is split with LangChain's recursive character splitter. Each chunk is embedded by all-MiniLM-L6-v2, which runs locally through the ONNX runtime bundled with ChromaDB 0.5.15. The vectors are stored in a persistent ChromaDB collection with an HNSW index (chroma-hnswlib 0.7.6), and numerical operations use numpy 1.26.4. Answers are generated by gpt-oss-120b served through the Groq API. The backend is a FastAPI application with a Strawberry GraphQL endpoint, and the chat interface is built with Streamlit. Repeated questions are handled by a Redis answer cache with an in-memory fallback, and the whole system is packaged as three services with Docker and Docker Compose.

The main parameters of each component, including the crawl limits, chunk size and overlap, retrieval depth, generation settings and cache configuration, are given with the corresponding design description in Section 3. The source code is available at https://github.com/cj7-k/raven.

5. Experiments and Results

5.1 Setup

As a demonstration domain we crawled Wikipedia's "Skin care" article and the pages linked from it (depth 1, at most 20 pages). All runs used a single Windows laptop with 16 logical processors, and the language model was accessed over the internet. The results are preliminary and qualitative: no labelled question set was used, and the latency figures are single observations rather than averages.

5.2 Corpus and the effect of citation removal

On the seed page of the corpus, the crawler found 154 candidate internal links. Excluding non-article pages (such as talk and category pages) and utility URLs left 101 links, and 20 pages were crawled. The text was split into chunks of 500 characters with a 50-character overlap, and each chunk was embedded as a 384-dimensional vector.

To measure the effect of citation markup, the crawl was first run without removing it. This produced 1,769 chunks, many of which were reference-list text made up of journal names, DOIs and PubMed identifiers. After the markup was removed at the crawler, the same procedure produced 898 chunks, a reduction of 49%, which shows that nearly half of the text originally indexed was reference material rather than explanatory content. Because the linked pages visited can differ slightly between crawls, this comparison is indicative rather than exact.

5.3 Effect on retrieval

To examine how this cleaning affects retrieval, we compared the four closest chunks for the question "How does sunscreen protect skin?" before and after citation removal. Before removal, two of the four results were bibliographic fragments, that is, journal references containing DOI and PubMed strings. These ranked highly only because they share vocabulary with the query, and they contain no usable explanation. After removal, all four results were explanatory prose from the "Skin care" article. In order of increasing cosine distance, they stated that sunscreen shields skin from chronic UV exposure (0.26), described a protective barrier against UV-driven collagen damage (0.28), advised applying sunscreen before sun exposure and reapplying after sweating (0.33), and explained that the SPF number indicates how well the product protects (0.34). Since these are cosine distances, lower values indicate closer matches.

5.4 Grounded answering

We next examined how the system answers questions through the chat interface, using five questions that covered in-scope, partially covered and out-of-scope cases. For in-scope questions, such as "How does sunscreen protect skin?" and "What causes eczema?", the answers were consistent with the retrieved text and were displayed together with their source pages.

The vitamin C question illustrates the effect of limited corpus coverage. The corpus mentions vitamin C only as one ingredient of an oral formulation studied for melasma. Instead of inventing details, the system reported that it had no information about a vitamin C serum. The out-of-scope question was declined with a statement that the provided information does not include the answer. In both refusal cases no sources were displayed, so that irrelevant citations did not appear beside a refusal. These results are qualitative and are not a measure of answer accuracy.

5.5 Latency and caching

We recorded the server-side latency returned by the API for newly generated and repeated questions. A newly generated answer took 2.8 s in a first run with the in-memory cache, and 1.2 s in the containerized deployment with Redis. This time is dominated by the network round trip and the language-model call. Repeated questions were served from the cache and reported a latency of 0 ms, which means they completed in under one millisecond, since the latency field has millisecond resolution. Serving a repeated question from the cache therefore avoids both retrieval and the language-model call. These figures are single observations. The difference between 1.2 s and 2.8 s reflects the remote model service and the network, so no statistical conclusion should be drawn from them.

6. Discussion

6.1 Engineering lessons

Source noise strongly affects retrieval. Citation lists nearly doubled the index and displaced useful passages from the top results. Cleaning at the crawler, where the markup is still identifiable, was simpler and more reliable than filtering chunks afterwards.

Target availability can masquerade as bot protection. The first intended corpus, an institutional website, was unavailable because its hosting account had been suspended, while a bot-check page in front of it initially suggested an anti-crawling measure. The cause became clear only after the fetched HTML was saved and the site was opened manually. Logging the number of links found per page and keeping the first fetched page for inspection made such problems quick to diagnose.

Compiled dependencies limit portability. The pinned vector-index library (chroma-hnswlib 0.7.6) publishes prebuilt Windows packages only for CPython up to 3.11, and the pinned numpy 1.26.4 has none for Python 3.13, so installation on newer Python versions requires a C++ toolchain. Checking every package in the resolved runtime environment against the package index showed that Python 3.11 was the only Windows version for which all 129 packages were available prebuilt. Separately, ChromaDB 0.5.15 calls its telemetry library in a way that releases 6.x and later reject, which produced repeated warnings even with telemetry disabled; pinning the earlier 5.x release removed them. Containerization removes this class of problem for end users.

Simple caching is effective for repeated questions. Normalising questions lets trivially different spellings of the same question share an entry, and the fallback design means the cache can be added without making the service depend on Redis.

6.2 Limitations

- There is no quantitative accuracy evaluation: no labelled question set, no retrieval metrics (such as recall at k), no measure of answer faithfulness, and no baseline without retrieval.
- The corpus is small and covers a single topic from a single site; behaviour on larger or noisier sites is untested.
- Latency figures are single observations and depend on the network and a third-party model service; the system needs internet access to answer new questions.

- The cache matches only questions that are identical after normalisation; paraphrases are not recognised.
- Chunking is character-based and ignores document structure, and there is no re-ranking stage.
- Provenance is at page level rather than passage level, and detecting a refusal relies on matching phrases in the model's reply, which is brittle.
- The crawler does not parse robots.txt and has no authentication or rate limiting on the API.

6.3 Ethical and licensing considerations

Wikipedia text is available under the Creative Commons Attribution-ShareAlike 4.0 licence, and RAVEN displays links to the source articles with every answer. The demonstration corpus concerns skin care, and generated answers are not medical advice. The crawler waits 0.5 s between requests and identifies itself with a standard browser user-agent string; deployments against third-party sites should add robots.txt compliance and an identifiable user agent. The language-model key is supplied through an environment file that is excluded from version control and from the container image.

7. Conclusion and Future Work

This paper described RAVEN, a retrieval-augmented chatbot that answers questions about a website from its own content, shows the sources, declines out-of-scope questions, serves repeated questions from a cache, and runs as a three-service Docker Compose application. A preliminary study on a 20-page corpus showed that removing citation markup halved the index and removed bibliographic fragments from the top results, and that cached answers were returned in under a millisecond.

Future work has four directions. First, a quantitative evaluation with a labelled question set, retrieval and faithfulness metrics, a no-retrieval baseline and repeated latency measurements. Second, retrieval improvements such as structure-aware chunking, hybrid keyword and vector search and re-ranking. Third, a semantic cache that recognises paraphrases and passage-level citations. Fourth, an automated test suite and the ability to ingest a new website from the user interface.

Code and Data Availability

The source code is available at https://github.com/cj7-k/raven. The corpus consists of public Wikipedia pages and can be regenerated with the crawler and preprocessing scripts in the repository.

References

[1] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung, "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 55, no. 12, Art. 248, 2023.

[2] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. 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, 2020, pp. 9459–9474.

[3] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-augmented generation for large language models: A survey," arXiv:2312.10997, 2023.

[4] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W. Yih, "Dense passage retrieval for open-domain question answering," in *Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP)*, 2020, pp. 6769–6781.

[5] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *Proc. EMNLP-IJCNLP*, 2019, pp. 3982–3992.

[6] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 5776–5788.

[7] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 4, pp. 824–836, 2020.

[8] Microsoft, "Playwright," https://playwright.dev (accessed Oct. 2026).

[9] Chroma, "Chroma: The open-source embedding database," https://www.trychroma.com (accessed Oct. 2026).

[10] OpenAI, "gpt-oss-120b & gpt-oss-20b model card," arXiv:2508.10925, 2025.

[11] Groq, Inc., "GroqCloud documentation," https://console.groq.com/docs (accessed Oct. 2026).

[12] O. Hartig and J. Pérez, "Semantics and complexity of GraphQL," in *Proc. The Web Conference (WWW)*, 2018, pp. 1155–1164.

[13] S. Ramírez, "FastAPI," https://fastapi.tiangolo.com (accessed Oct. 2026).

[14] "Strawberry GraphQL," https://strawberry.rocks (accessed Oct. 2026).

[15] Streamlit Inc., “Streamlit,” https://streamlit.io (accessed Oct. 2026).

[16] Redis Ltd., “Redis,” https://redis.io (accessed Oct. 2026).

[17] D. Merkel, “Docker: Lightweight Linux containers for consistent development and deployment,” *Linux Journal*, vol. 2014, no. 239, Art. 2, 2014.