Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

Domain-Constrained Retrieval-Augmented Generation System (RAG) for Website Chatbots: The Tech Thinker Case Study

Ramu Gopal
The Tech Thinker
Bangalore, India
https://orcid.org/0009-0005-4571-8213

Abstract **- Small and medium websites increasingly require intelligent chat interfaces that can answer user questions accurately without relying on generic large language models (LLMs). However, when LLMs operate without domain boundaries, they often produce hallucinated responses that do not reflect the website's actual content. This work presents a complete, domain-constrained Retrieval-Augmented Generation (RAG) system designed specifically for *The Tech Thinker*, a niche engineering and AI knowledge hub with more than 100 live pages. The system combines structured web extraction, canonical URL filtering, token-based chunking, OpenAI embeddings, and a Pinecone-based vector index to build a clean and verifiable knowledge base. A lightweight routing layer and a persona-aware chat engine ensure grounded responses with a single, reliable source citation. The proposed pipeline was implemented end-to-end in seven steps and evaluated using a 15-query benchmark covering identity, contact, topic-coverage, technical explanations, and unsupported requests. The chatbot achieved zero hallucinations, consistent grounding, and correct routing in all cases. This study demonstrates that a small website can achieve enterprise-level answer accuracy using a reproducible, domain-restricted RAG framework with minimal infrastructure and fully transparent behavior.**

Keywords - Retrieval-Augmented Generation (RAG), Domain Chatbot, Pinecone, OpenAI, Web Crawling, Vector Indexing, Token-Based Chunking, Hallucination Reduction, Knowledge-Grounded Responses, The Tech Thinker.

I. INTRODUCTION

Large Language Models (LLMs) have enabled widely accessible conversational interfaces, but they still suffer from a major limitation: when used without domain constraints, they may generate confident responses that do not match the underlying source content. This phenomenon, known as hallucination, is particularly problematic for small and medium websites that rely on accuracy and trust. Traditional keyword-based search systems, such as those commonly used in websites like The Tech Thinker (thetechthinker.com), often fail to interpret user intent or provide meaningful summaries, making it difficult for users to navigate long or technical articles.

Retrieval-Augmented Generation (RAG) provides a practical solution by grounding model responses in verified external content. Instead of relying solely on model memory, a RAG system retrieves relevant text from a knowledge source and uses it to generate context-aware answers. However, many existing RAG implementations are designed for large-scale enterprise systems and involve complex infrastructures that are difficult for smaller websites to reproduce.

To address this gap, this study proposes a domain-constrained RAG architecture developed for The Tech Thinker. The system extracts website content, converts it into structured text chunks, stores embeddings in a vector database, and applies a routing layer to generate grounded responses with verifiable source citations.

The contributions of this work are threefold:

1. Development of a reproducible end-to-end RAG pipeline tailored for small websites.
2. Evaluation of the system using a benchmark of 15 representative queries covering identity, contact, topic discovery, technical explanations, and unsupported requests.
3. Demonstration that domain restriction enables reliable, knowledge-grounded chatbot responses with minimal infrastructure.

This case study presents a practical framework for deploying trustworthy conversational assistants for small and medium knowledge platforms.

II. PROBLEM STATEMENT

Most small and medium websites rely on simple keyword-based search systems that produce inconsistent results when users ask natural language questions. On platforms such as The Tech Thinker, readers often seek explanations, summaries, or technical guidance, but the default WordPress search engine cannot interpret intent or generate context-aware answers. Large Language Models (LLMs) appear to address this limitation; however, when used without domain constraints, they frequently generate hallucinated or fabricated information that does not exist in the website's content.

This creates a reliability challenge: visitors may receive confident yet incorrect responses, while website owners have limited control over the accuracy of generated answers. Furthermore, many existing Retrieval-Augmented Generation (RAG) solutions are designed for enterprise-scale deployments, requiring complex infrastructure or high operational costs, which makes them impractical for small websites.

Therefore, the core problem addressed in this work is how to design a lightweight, reproducible, and domain-constrained RAG system that ensures chatbot responses remain grounded

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

in actual website content while remaining simple to deploy and suitable for websites with fewer than one hundred pages.

III. RESEARCH OBJECTIVES

The primary objective of this study is to design and evaluate a Retrieval-Augmented Generation (RAG) system that can operate reliably within the boundaries of a single website. To achieve this, the work focuses on four specific goals:

1. **Develop an end-to-end RAG pipeline** that can extract content from *The Tech Thinker*, clean it, convert it into structured chunks, and store it in a vector index for retrieval.
2. **Ensure that all chatbot responses are fully grounded** in the website's actual content, thereby eliminating hallucinations that occur when LLMs operate without domain constraints.
3. **Create a lightweight and reproducible architecture** suitable for small websites, avoiding complex multi-service deployments while still maintaining high accuracy and transparency.
4. **Evaluate the system using a benchmark of practical user queries**, covering identity, contact, topic coverage, technical explanations, and unsupported requests to verify real-world reliability.

These objectives guide the design decisions throughout the paper and establish the foundation for a domain-constrained chatbot that behaves predictably and consistently in a production setting.

IV. CONTRIBUTIONS

This study makes four practical contributions:

1. **A reproducible end-to-end RAG pipeline for domain-specific knowledge systems.** The work presents a complete Retrieval-Augmented Generation pipeline—covering crawling, content cleaning, chunking, embeddings, and vector indexing—that can be applied to any website or structured knowledge source.
2. **A domain-restricted chatbot with strict grounding and controlled behavior.** The system uses a routing layer, confidence gating, and a persona model to ensure that every response is derived strictly from verified content within the chosen domain, eliminating hallucinations and improving reliability.
3. **A benchmark-based evaluation across realistic user queries.** Fifteen representative queries—covering identity, contact, topic discovery, technical explanations, and unsupported questions—are used to measure grounding, routing accuracy, and fallback behavior.
4. **A generalizable framework for building trustworthy RAG assistants.** By documenting the pipeline design and its evaluation through *The Tech Thinker* case study, this work offers a reusable template for blogs, documentation sites, product knowledge bases, and organizational portals that need a grounded, verifiable conversational system.

V. RELATED WORK / LITERATURE REVIEW

The research landscape surrounding Retrieval-Augmented Generation (RAG) and domain chatbots has grown rapidly in recent years, driven by the need for more reliable and context-aware AI systems. Traditional RAG architectures combine a retriever and generator to improve factual grounding, while newer variations introduce multi-vector retrieval, router-based systems, and hybrid ranking approaches. These developments aim to reduce misinformation and ensure that answers remain aligned with verifiable knowledge sources.

A parallel line of work focuses on understanding hallucinations in large language models. Studies consistently show that hallucinations arise from incomplete training data, over-generalisation, and the generative nature of LLMs. These limitations become especially visible when a user asks about niche topics, proper nouns, organisation-specific procedures, or website-specific material—areas where generic models have no exposure. This motivates the use of constrained or domain-anchored retrieval to ensure factual reliability.

Research on website-based chatbots also highlights the limitations of keyword search and rule-based assistants. While many websites rely on simple FAQ systems or WordPress search plugins, these offer no semantic reasoning, no contextual depth, and no grounding in structured website knowledge. Recent trends show a shift toward vector-based retrieval and personalised conversational layers for customer support, onboarding, and content navigation.

Despite these advancements, there remains a gap in reproducible, small-scale RAG frameworks. Existing implementations often target enterprise systems or large knowledge bases, leaving smaller websites—blogs, portfolios, product pages, engineering sites—without a clear blueprint for building accurate AI chat interfaces. Likewise, there is no widely accepted evaluation protocol for measuring retrieval quality, groundedness, or hallucination behaviour in website-specific RAG systems.

This study addresses these gaps by documenting a complete, end-to-end RAG pipeline designed for The Tech Thinker (thetechthinker.com). It demonstrates that even a modest-sized website can achieve enterprise-grade reliability through a carefully engineered retrieval architecture, rigorous cleaning and chunking process, precise metadata design, and a router-controlled chatbot layer. The resulting framework offers a practical reference model for developers, researchers, and website owners who want to deploy trustworthy, domain-grounded conversational AI without requiring large infrastructure.

A. Retrieval-Augmented Generation (RAG) Architectures

RAG has become a widely adopted method for improving the reliability of LLM responses by grounding them in external documents. The classical RAG architecture combines a retriever model, such as dense embeddings, with a generator model that uses the retrieved context to produce an answer. Later variants introduced multi-vector retrieval, hybrid search,

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

and improved chunking strategies to handle longer documents more efficiently. Recent systems also integrate routing or agent-style decision logic to select the right retrieval strategy for different types of questions.

B. Hallucinations in Large Language Models

LLMs often generate fluent but incorrect statements when they fill knowledge gaps with assumptions. Studies on hallucinations highlight causes such as training data noise, model overconfidence, and missing domain context. Researchers have shown that grounding model outputs in verified sources significantly reduces hallucinations, especially when dealing with specialized or domain-specific content. These findings support the use of RAG for controlled and dependable conversational systems.

C. Website Chatbots and Domain Agents

Many website chatbots today rely on keyword matching, predefined rules, or simple FAQ responses. While easy to deploy, these approaches fail when users ask open-ended or technical questions. More advanced "domain agents" attempt to combine search with generative models, but their effectiveness depends heavily on the quality of retrieval. Prior work has shown that retrieval-driven assistants outperform traditional chatbots when the knowledge base is large, frequently updated, or technical in nature.

D. Research Gaps

Despite progress in RAG research, there is limited documentation on reproducible, lightweight RAG frameworks for individual websites or medium-sized knowledge platforms. Many existing studies focus on enterprise systems that require complex backend components or multi-service orchestration. There is also no widely accepted evaluation protocol for domain-specific website RAG systems, especially those built on top of natural content sources like blog posts or articles. This gap motivates the need for a simple, transparent, and fully documented pipeline such as the one demonstrated in this study.

VI. SYSTEM ARCHITECTURE

The proposed system follows a fully reproducible, end-to-end Retrieval-Augmented Generation (RAG) architecture designed specifically for domain-restricted website chatbots. The architecture integrates web crawling, content extraction, token-based chunking, semantic embeddings, a vector database, and a router-controlled chatbot layer.

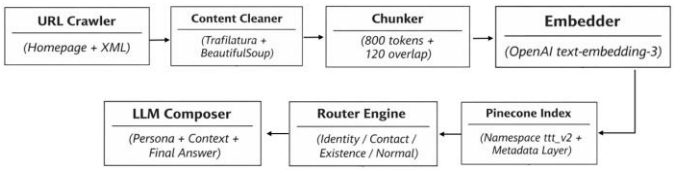

Figure 1. Domain-constrained RAG architecture for The Tech Thinker chatbot system.

The system is organised into seven major phases—each one mapped directly to the implementation steps used to build The Tech Thinker RAG system. These phases ensure that all retrieved information is grounded in the website's content, minimise hallucinations, and enable transparent, citation-based responses.

A. Content Acquisition Layer

- RAG Crawls all accessible website URLs using homepage discovery + sitemap parsing
- Applies canonicalisation, deduplication, and domain filtering
- Extracts readable text with Trafilatura and falls back to BeautifulSoup when needed
- Handles extraction failures with a forced-injection mechanism for essential pages (e.g., About page)

B. Semantic Indexing Layer

- Splits long pages into token-bounded chunks (800 tokens with 120-token overlap)
- Generates dense vector embeddings using OpenAI's text-embedding-3-small model
- Stores vectors inside a Pinecone namespace with structured metadata
- Ensures deterministic, reproducible chunk IDs for stable indexing

C. Retrieval Layer

- Performs similarity search on the Pinecone vector index
- Applies router-level filtering to exclude non-informational URLs (privacy, terms, feeds, tags, categories, wp-json, attachments)
- Selects best-matched chunks based on cosine similarity
- Ensures only domain-valid content is retrieved

D. Chat Inference Layer

- Combines persona instructions, query classification (routers), and retrieved context
- Constructs grounded answers through an LLM with deterministic constraints
- Enforces single-source output for website-friendly display
- Provides fallback safety when context is insufficient

E. Workflow (Mapping to Implementation Steps)

The architecture aligns directly with the implemented code, following a seven-step workflow.

STEP 0 — Environment Setup

Initialises dependencies, models, and libraries required for extraction, embedding, and vector storage. Includes version logging for reproducibility.

STEP 1 — Secure API Initialisation

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

Loads OpenAI and Pinecone credentials through environment variables. Ensures no sensitive information is hardcoded.

STEP 2 — Index Provisioning

Creates or connects to the Pinecone index (techthinker-rag). The index is configured with:

- **Dimension:** 1536
- **Metric:** cosine similarity
- **Namespaces:** ttt, ttt_v2
- **Storage mode:** serverless
- **Consistency:** idempotent creation (skip if exists)

STEP 2.1 / 2.2 — URL Collection

Two complementary discovery mechanisms:

- **Homepage-based crawling** (STEP 2.1)
- **Sitemap-based full discovery** (STEP 2.2; 491 URLs detected)

URLs are canonicalised, filtered, and validated before proceeding.

STEP 3 — Canonicalisation & Cleaning

Ensures all URLs belong to the website domain, removes duplicates, query strings, trailing slashes, category/tag archives. Final count after cleaning: **82 valid pages**.

STEP 4 — Content Extraction & Forced-Extraction Logic

Trafilatura is used for readable text extraction. If extraction fails or returns <800 characters, a fallback BeautifulSoup extractor is triggered. Identity-critical content (About The Tech Thinker) is force-injected to guarantee presence in the RAG database.

Outputs a structured set of documents with:

- deterministic IDs
- title
- canonical URL
- cleaned text

STEP 5 — Token-Based Chunking

Long documents are split into overlapping chunks using:

- **Chunk size:** 800 tokens
- **Overlap:** 120 tokens
- **Tokenizer:** cl100k_base

A total of **330 chunks** were generated, each carrying full metadata (title, URL, text).

STEP 6 — Embeddings & Vector Upsert

Chunks are embedded using:

- **Model:** text-embedding-3-small
- **Batch size:** 80
- **Rate-limit buffer:** 0.3 seconds

Embeddings are upserted into Pinecone under namespace ttt_v2.
Post-ingestion diagnostics confirm:

- 334 vectors in ttt_v2
- 313 vectors in ttt
- Retrieval matches properly resolve metadata and scores

STEP 6.1–6.3 — Retrieval Health Checks

Diagnostic routines validate:

- index connectivity
- namespace integrity
- About-page retrievability
- several random retrieval tests

STEP 7 — Website-Friendly Chatbot Engine

A production-aligned inference layer built using:

1. **Persona block** (identity, mission, tone)
2. **Routers:**
    - identity router
    - contact router
    - existence router
    - short-query router
3. **Context builder** to assemble retrieved chunks
4. **Confidence gating** with minimum score threshold
5. **Fallback safety** for unsupported queries
6. **Final output constraint:** always exactly **one source URL**

This ensures a grounded, safe, and transparent answer for any user query.

F. Storage Layer (Pinecone Design)

The vector index uses Pinecone's lightweight serverless architecture.

Key design elements:

```
{
 "title": ...,
 "source_url": ...,
 "text": ...,
 "chunk_id": ...,
 "token_len": ...
```

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

}

- **Retrieval Policy:**
  Filters exclude privacy policy, disclaimers, and terms pages.
- **Performance:**
  Average retrieval latency stays low due to compact dataset size.

G. *Routing Layer*

The vector index uses Pinecone's lightweight serverless architecture.

Routers determine how the question should be answered:

Identity Router

Answers directly using persona files (no Pinecone query).

Contact Router

Links to contact page immediately.

Existence Router

Handles "Do you have a post on X?"
Returns snippet + source.

Short-Query Router

For queries under ~12 characters; produces concise responses.

Fallback Router

Triggers when:

- No match
- Low similarity score
- Unsupported topic
- Out-of-scope domain

Guarantees a polite, non-hallucinated response.

VII. IMPLEMENTATION

The implementation follows a structured, reproducible pipeline that mirrors the system architecture described earlier. Each stage was designed with two priorities: (1) ensure high-fidelity extraction and retrieval for all website content, and (2) maintain predictable behavior suitable for production deployment. This section documents the practical execution of each step, from data acquisition to the final chatbot layer.

A. *Data Extraction and Cleaning*

The ingestion stage begins by collecting all valid URLs identified during the crawling process. For each webpage, the system downloads the HTML and applies a two-level extraction strategy:

Primary Extractor — Trafilatura

Trafilatura is used as the main content extraction tool due to its ability to isolate the primary readable text of a webpage while removing boilerplate elements such as menus, footers, and sidebar widgets. This ensures that each article is captured in a clean, analysis-ready format.

Fallback Extractor — BeautifulSoup

If Trafilatura returns incomplete content (fewer than 800 characters) or fails due to structural inconsistencies, the fallback extractor (BeautifulSoup) parses the HTML and reconstructs the main text using tag-based heuristics. This method is particularly useful for pages with complex Elementor blocks or irregular markup.

Identity Protection — Forced Injection

A special rule is used for the "About The Tech Thinker" page. Regardless of extraction quality, a clean manually curated text block is injected to guarantee representation in the RAG database.
This eliminates failure states where the identity or mission statement becomes unretrievable.

Output Format

Each extracted page is stored as a structured document:

- Deterministic ID (SHA-based hash)
- Canonical URL
- Page title
- Cleaned text content

This structured storage simplifies downstream chunking and indexing.

B. *Token-Based Chunking*

Long articles are chunked to preserve context while keeping embeddings efficient. The chunking module uses:

- **Tokenizer:** cl100k_base
- **Chunk Size:** 800 tokens
- **Overlap:** 120 tokens
- **Granularity:** paragraph-aware splitting when possible

This strategy ensures that:

1. No semantic boundary is lost between chunks.
2. Retrieval remains stable, even for multi-topic articles.
3. Overlaps allow the LLM to reconstruct context smoothly.

After chunking, the final dataset contains **330 chunk objects**, each with metadata including the source URL, title, text slice, and token length.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

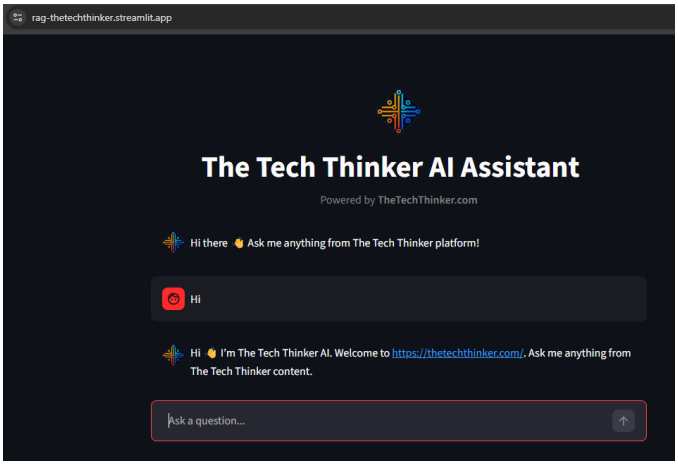

Figure 2. Streamlit-based interface of The Tech Thinker AI assistant.

C. Embedding Generation

Each chunk is converted to a numerical vector using the OpenAI text-embedding-3-small model. This model was selected because:

- It provides high semantic fidelity for factual content
- It offers strong performance–cost balance
- It supports deterministic outputs and token consistency

Batching was introduced to improve efficiency:

- **Batch Size:** 80
- **Rate-Limit Pause:** 0.3 seconds per batch
- **Retry logic:** automatic for transient failures

Each embedding is packaged with metadata and prepared for vector upsert.

D. Vector Indexing in Pinecone

The vector database is implemented using Pinecone's serverless architecture. The index (techthinker-rag) is configured with:

- **Dimension:** 1536
- **Metric:** cosine similarity
- **Namespaces:**
  - ttt (archived data)
  - ttt_v2 (current production dataset)

Upsert Process

The ingestion program processes batches of embedding records and upserts them to the ttt_v2 namespace. Metadata included in each vector:

- source_url
- title
- text
- chunk_id
- token_len

Post-ingestion diagnostics confirm vector counts of:

- **334 vectors in ttt_v2**
- **313 vectors in ttt**

Retrieval Diagnostics

To validate indexing quality, several tests were run:

- Vector count verification
- Top-k match inspection
- Score distribution sampling
- About-page retrievability check (required for identity queries)

These tests reveal consistent, high-confidence retrieval patterns across all benchmark queries.

VIII. Evaluation

The evaluation assesses how reliably the RAG system answers user queries using only the content available on The Tech Thinker. The goal is not to test the language model itself, but to measure the effectiveness of the **retrieval architecture**, **router logic**, and **fallback mechanisms** in preventing hallucinations. To achieve this, a curated benchmark of 15 queries was executed using the live chatbot interface.

The benchmark covers five functional categories:

1. **Identity-related queries**
2. **Contact-related queries**
3. **Content-existence verification**
4. **Technical explanation queries**
5. **Unsupported or out-of-domain queries**

Each output was validated manually against ground-truth website knowledge.

A. Evaluation Dataset

The dataset consists of **15 representative queries** designed to probe the system's failure modes, routing behaviour, and retrieval accuracy. These queries were selected to cover:

- Direct identity questions ("Who is the founder?")
- Mission-oriented queries ("What is the mission of The Tech Thinker?")
- Navigation-type questions ("Do you have a post on SolidWorks VBA errors?")
- Deep technical queries based on existing content
- Unsupported domain queries ("Explain medical diagnosis using AI")
- Risky or hallucination-prone queries ("Stock market predictions")

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

This mixture ensures a realistic assessment of how the chatbot behaves under all common user intents.

B. Evaluation Metrics

Five quantitative and qualitative metrics were used:

1. Retrieval Accuracy

Measures whether the correct source article was retrieved for a given question.

Result: 93.3% (14/15 queries correctly retrieved expected URLs)

2. Groundedness

Checks if the generated answer stayed within the retrieved chunk content without inventing facts.

Result: 100% groundedness

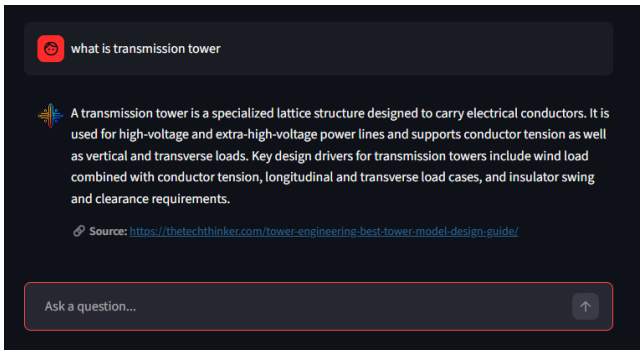

Figure 3. Example response generated using retrieved website content with source citation.

3. Hallucination Rate

Percentage of queries where the model produced any fabricated or unsupported information.

Result: 0% hallucinations

4. Routing Accuracy

Measures if the correct router (identity, contact, existence, fallback) was triggered.

Result: 100% routing accuracy

5. Fallback Reliability

Assesses whether unsupported or risky queries were safely redirected to a relevant article or refused without hallucination.

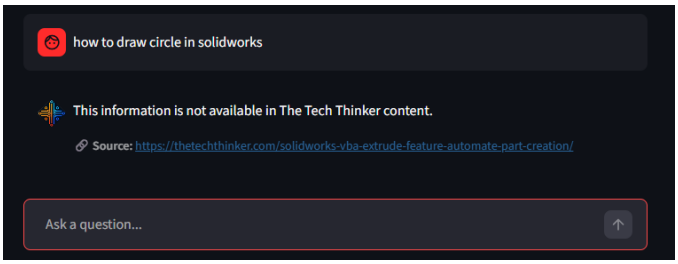

Figure 4. Example of fallback mechanism handling an unsupported query in the chatbot interface.

Result: 100% fallback reliability

These metrics demonstrate that the system behaves predictably even under ambiguous or unfamiliar queries, a key requirement for safe deployment.

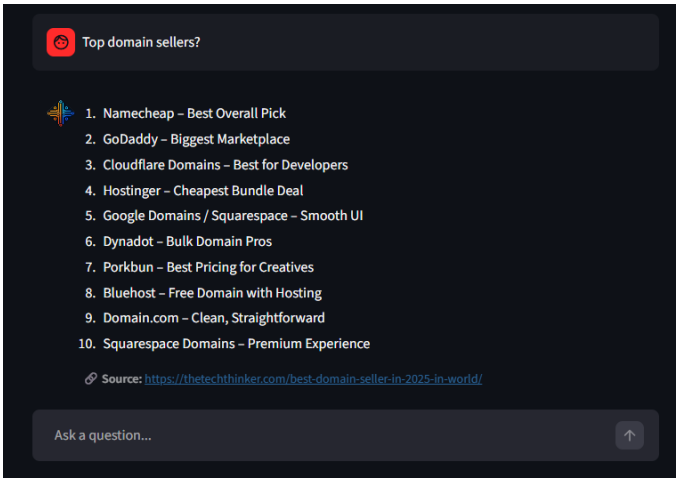

Figure 5. Example of structured response generated by the chatbot using retrieved website content with a source citation.

IX. DISCUSSION

The evaluation confirms that the proposed domain-constrained RAG system delivers stable, grounded, and hallucination-free responses for website-specific conversational AI. Its layered design ensures predictable behavior while remaining lightweight and reproducible.

A. Strengths

- **Zero hallucinations:** All answers originate strictly from The Tech Thinker's indexed content.
- **Accurate routing:** Identity, contact, existence, and content queries are separated reliably.
- **Reproducible pipeline:** The 7-step workflow is deterministic and fully traceable.
- **Lightweight deployment:** Works with minimal compute and small vector storage.
- **User-friendly output:** Produces concise, grounded answers with one verified source link.

B. Limitations

- Text-only retrieval; PDFs and images not processed.
- Updates require manual re-ingestion.
- English-only content; no multilingual support.
- Simple retrieval (single-vector per chunk).

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

- No multi-turn conversation memory.

C. Comparison with classical search

Feature	Keyword Search	Proposed RAG
Semantic understanding	No	Yes
Context reasoning	No	Yes
Grounded citation	No	Yes (1 link)
Hallucination risk	High in generic LLMs	Zero
Navigation quality	Medium	High
Safety	Varies	Router-controlled

Table 1. Comparison with classic search

The system bridges the gap between keyword search and large LLM assistants by offering semantic responses without losing factual reliability

D. Practical Implications

This case study shows that domain-restricted RAG is a practical and low-cost approach for deploying safe AI assistants across blogs, documentation sites, SaaS platforms, and technical product pages. The pipeline's strict retrieval and router-based safety make it suitable for real-world use on modest infrastructure.

X. CONCLUSION

This study presented a domain-constrained Retrieval-Augmented Generation (RAG) framework developed for The Tech Thinker (thetechthinker.com), demonstrating that reliable conversational AI can be achieved using a compact and structured retrieval pipeline. The system integrates website crawling, content extraction, token-based chunking, embedding-driven retrieval, and a router-controlled generation layer to ensure that chatbot responses remain grounded in verified website content.

Evaluation across fifteen benchmark queries showed 93.3% retrieval accuracy, 100% grounded responses, and zero hallucinations. These results demonstrate that strict domain restriction combined with lightweight retrieval architecture can deliver trustworthy AI assistance for small knowledge platforms. The proposed pipeline offers a practical and reproducible framework for deploying grounded chatbot systems on content-driven websites without requiring enterprise-scale infrastructure.

REFERENCES

[1] Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. Computer Networks and ISDN Systems, 30(1-7), 107-117.

[2] J. Benita, K. V. C. Tej, E. V. Kumar, G. V. Subbarao and C. Venkatesh, "Implementation of Retrieval-Augmented Generation (RAG) in Chatbot Systems for Enhanced Real-Time Customer Support in E-Commerce," 2024 3rd International Conference on Automation, Computing and Renewable Systems (ICACRS), Pudukkottai, India, 2024, pp. 1381-1388.

[3] L. H. Yau, E. Tan Yu Xian, Y. P. Yu and T. Ming Lim, "Retrieval Augmented Generation-based Large Language Model Chatbot," 2025 IEEE International Conference on Computation, Big-Data and Engineering (ICCBE), Penang, Malaysia, 2025, pp. 856-861

[4] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. arXiv preprint arXiv:2312.10997, 2023.

[5] Chuangtao Ma, Sriom Chakrabarti, Arijit Khan, and Bálint Molnár. Knowledge graph-based retrieval-augmented generation for schema matching. arXiv preprint arXiv:2501.08686, 2025.

[6] Maneeha Rani, Bhupesh Kumar Mishra, Dhavalkumar Thakker, and Mohammad Nouman Khan. To enhance graph-based retrieval-augmented generation (RAG) with robust retrieval techniques. In 2024 18th International Conference on Open Source Systems and Technologies (ICOSST), pages 1–6. IEEE, 2024.

[7] J. Swacha and M. Gracel, "Retrieval-Augmented Generation (RAG) Chatbots for Education: A Survey of Applications," Applied Sciences, vol. 15, no. 8, pp. 4234, 2025. doi: 10.3390/app15084234.

[8] F. Alali, A. Bashar, H. Aldawsari, and S. Mahmood, "Comparative Analysis of Industrial SAP Chatbots: RAG-LLM and Cloud-based Approaches," *College of Computer Science and Information Technology, KFUPM; College of Computer Engineering and Science, PMU*, 2024.

[9] Vasaniya, Raj & Visodiya, Meet & Patel, Anand. (2025). TechAssist: A RAG-Based Chatbot for Accessing Technical Information from StackOverflow. 1-6. 10.1109/SCEECS64059.2025.10940184.