IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155120

Retrieval-Augmented Generation for Smarter Web Scraping and Synthesis: A Cache-Aware Multimodal Framework for Web Intelligence

Rishith Poojary[1], Aryan Rana[2], Aditya Magar[3], Dev Raval[4], Pravin Shinde[5]

Students, Department of Artificial Intelligence and Data Science, Shah & Anchor Kutchhi Engineering College[1-4]

Assistant Professor, Department of Artificial Intelligence and Data Science, Shah & Anchor Kutchhi Engineering College[5]

Abstract: Most web pages today do not deliver data in straightforward ways. Content often appears only after scripts run, making early snapshots incomplete. Each visit might show a slightly altered layout. Identical details - like pricing or bylines - sit inside unpredictable tag arrangements depending on the site. Tools relying on fixed rules, such as CSS paths or XPath, function reliably until design changes occur. A minor update may disrupt what once worked without warning. Our system, WebRAG, takes another path altogether. Viewed as a form of information gathering, web scraping here builds on three distinct aspects per webpage: raw textual elements, structural markup shaped by creators, alongside how content visually appears during browsing - these together anchor a generation process firmly within up-to-date materials instead of outdated datasets. When assessing prior visits, a caching mechanism checks if underlying structures have shifted, skipping repeated processing where little or nothing differs from earlier versions. Each result includes documented origins detailing location online, specific document fragments involved, along with reliability estimates tied to recovery quality. Testing occurred using WebRAGBench - a collection exceeding five thousand hand-labeled examples drawn from news outlets, shopping platforms, and knowledge-focused websites. Performance surpassed rule-driven methods plus standard text-based pipelines when measuring correctness in fetching data, fidelity in pulling out answers, and overall response speed. A single 2.8× boost in throughput came just from caching. Where performance still lags becomes clear when examining system limits - this points toward promising paths forward.

Keywords: Retrieval-Augmented Generation, Web Scraping, Large Language Models, Multimodal Embeddings, Information Extraction

I. INTRODUCTION

Most websites change too fast for old scraping methods to keep up. Though selecting elements by path or class works fine on predictable pages, real sites rarely stay consistent. Content often appears only after scripts run, delaying visibility. Identical links might show distinct structures depending on who visits. Designs shift without warning due to live experiments. What worked yesterday may fail today without explanation. Most websites prioritize how things look instead of using meaningful HTML structure. When layouts change, tools built on old versions might pull nothing - or wrong data - without warning.

One path emerged through large language models. Because such systems interpret text deeply, they handle messy formatting better than rigid rule-based tools. Yet a clear risk remains: pulling answers purely from internal knowledge might lead to confident-sounding inaccuracies. This happens when past learning overrides current truth. To counter that, retrieval-augmented generation adds an external step. Instead of relying on stored patterns, the model consults live information drawn straight from source material.

One problem with typical RAG systems here lies in how they view web pages - as loose collections of words without form. Because these models ignore structure, details like a price located in a product box versus a sidenote go unnoticed. Visual prominence also gets lost; bold headings matter more than small paragraphs below, yet the system treats them equally. Early experiments showed mistakes when handling online stores, where items appear in neat visual blocks. These layouts make sense to people, but if the underlying code does not reflect the design clearly, the model misreads key facts. What looked obvious by sight became noise in processing.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026
DOI: 10.17148/IJARCCE.2026.155120

What makes WebRAG different? Text, DOM, and visuals merge into a single retrieval format. Efficiency gains come next - repeated crawls hit stale copies often. Instead of redoing all embeddings each time, a smart check kicks in. Structural likeness of the DOM decides if updates matter. When little shifts occur, fresh encodings get skipped entirely. The main contributions of this work are as follows:

- A system pulls together text, structure from web pages, because it combines different types of data into unified representations. Layout cues guide matching alongside words found in content. Information flows through layers tuned to recognize both code patterns and image-like arrangements. Retrieval improves when signals align across modalities during search tasks.
- A fresh look at how data chunks repeat across nearly identical pages reveals a smarter way to store them - reusing what's already processed slashes repeated work by a large margin.
- A traceable extraction component stores the originating web address, associated webpage element, along with certainty level for each piece of captured data. While gathering information, it logs where things come from.
- This ensures transparency in how outputs are built. Every result carries its history by design. Retrieval context becomes part of the record automatically. Source details stay linked throughout use.
- WebRAGBench, an annotated dataset of over 5,000 pages across three web domains, with experiments showing consistent gains over baseline methods.

II. RELATED WORK

The RAG paradigm was established by Lewis et al. [1], who showed that pairing a dense retriever with a sequence-to-sequence generator substantially improved open-domain question answering. That work triggered a wave of follow-on research. Improvements to the retrieval side — better dense encoders, cross-encoder reranking, and adaptive retrieval strategies that decide whether retrieval is even needed [1], [2], [7] — have been especially active. More recently, work on versioned or temporally evolving documents [8] has wrestled with knowing when cached retrieval results have become stale, a concern that directly parallels our cache invalidation problem.

Multimodal retrieval is a more recent thread. Work on document question answering over scanned PDFs and forms [4] showed that adding visual signals alongside text improves extraction on layout-heavy materials. Broader treatments of multimodal RAG architectures have appeared in recent surveys [3], [9], and domain-specific work — covering medical literature [10], [11], industrial inspection [14], and XR maintenance guidance [5] — confirms that richer input representations consistently help. Ontology-grounded retrieval [6] and multimodal question answering [12], [13], [20] add further breadth to the picture.

What is largely missing from this body of work is attention to live, open-web scraping. Most multimodal RAG papers target curated corpora: PDFs, scanned forms, structured databases. Live web pages are considerably messier — layouts shift without warning, JavaScript populates content asynchronously, and no consistent schema exists across sites. WebRAG is, to the best of our knowledge, the first system to bring multimodal retrieval, cache-aware embedding management, and provenance tracking together in an architecture designed specifically for this setting.

III. PROPOSED METHODOLOGY

At the core of WebRAG lie four key parts: first, a system that gathers web-data; next comes a multimodal retrieval unit, pulling together diverse formats.A caching mechanism follows, storing frequently accessed information efficiently. Finally, there is a synthesis component aware of source origins - each piece detailed further on.

A. Web Data Collection

One crawl generates four outputs for every web page: the original HTML code, the structured DOM representation, an image of how it looks when loaded in a browser without interface, along with data listing address, time stamp, and server details sent back during access. These items get collected at once instead of pulled later - this helps the next phase skip live internet steps. When scripts must run, automation tools step in; otherwise, simpler downloads handle basic files quicker. Stored early, used smoothly

B. Multimodal Retrieval

One embedding emerges per page through a tripartite design. From cleaned text, a familiar transformer builds meaning-based vectors: these are T, sized 768 dimensions deep. Instead of words, structure speaks - the DOM tree feeds into an attention-driven graph model, where tags, nesting level, and neighbors shape D, another set of 768-length codes. On screen appearances, a convolutional net scans snapshots, summarizing regions via max pooling; output lands as V, capturing spacing, size shifts, groupings across pixels, also 768 elements long.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026
DOI: 10.17148/IJARCCE.2026.155120

One way to merge the three embedding vectors involves a weighted combination method
R equals weight times temperature, combined with duration multiplied by its weight, then added to volume adjusted by its corresponding factor
For WebRAGBench, a separate 15% portion guided the selection of weights - wt, wd, and wv - via grid search. These values stayed unchanged during testing phases that followed. Adjusting them dynamically appears later in Section X, framed as a path yet to be explored.

C. Cache-Aware Retrieval
Generating embeddings takes up most of the processing time here. During regular crawls, identical web addresses get checked again every day - or sometimes more often - though their content rarely shifts enough to alter data retrieval. For each link, the system keeps hold of the latest combined embedding R, paired with a compact representation of the webpage structure. When a site comes back into view, its present structural outline faces off against the saved version through a measure based on how many edits it would take to align them. When similarity goes above threshold τ, the system pulls the stored R. Otherwise, it recalculates all three embeddings and refreshes the cache. Workload tests recorded 72% of requests hitting the cache. That cut down embedding work by 68%. Throughput jumped - nearly triple what it was before.

D. Provenance-Aware Synthesis
It begins with every fetched section having three attached details: where it came from online, the precise path within the webpage structure pointing to its origin, also how closely it matched during retrieval based on semantic similarity. Sent together with the actual content, these markers go into the language model processing step, staying visible through to what finally appears. Because of this setup, any piece pulled out later links directly to one exact spot on a given site - helpful when users check accuracy, just as much when developers track down why something went wrong. Ending here makes tracing possible.

IV. SYSTEM ARCHITECTURE

Starting with modularity, the WebRAG process moves through distinct phases. Raw web content arrives either via Selenium - useful when pages change dynamically - or through a minimal tool suited for fixed content. From there, elements like HTML code, DOM hierarchy, screen image, and additional details enter separate encoding paths at once. One path uses transformers to capture textual meaning, another applies attention-based networks to interpret structural links in the DOM, while a third relies on convolutional models to analyze spatial design. These encoded representations merge by applying formula (1), preparing them before entering storage managed by the caching system. When the cache returns a hit, it delivers the saved embedding; if not, the system saves and indexes the fresh one. During queries, retrieval pulls the K best-matching page chunks from the FAISS HNSW structure, tags them with origin details, then passes them along. The LLM synthesis component shapes these into organized results, linking each field back to its source.

Shown in Fig. 1 is how the process moves step by step. Each part works on its own - swap out the data collector for another source tool, exchange one encoder for a different model in the embedding phase, while expanding the search index leaves the response generator untouched.

Shown in Figure 1, the WebRAG pipeline begins with a collection layer that supplies data to three separate embedding encoders running simultaneously. These encoded outputs, once combined, get stored temporarily through caching and organized via indexing. When a query arrives, the system retrieves the best-matching K segments along with their source details. This bundle - retrieved content paired with origin information - is then passed to the large language model. There, it undergoes structured integration to form coherent output.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155120

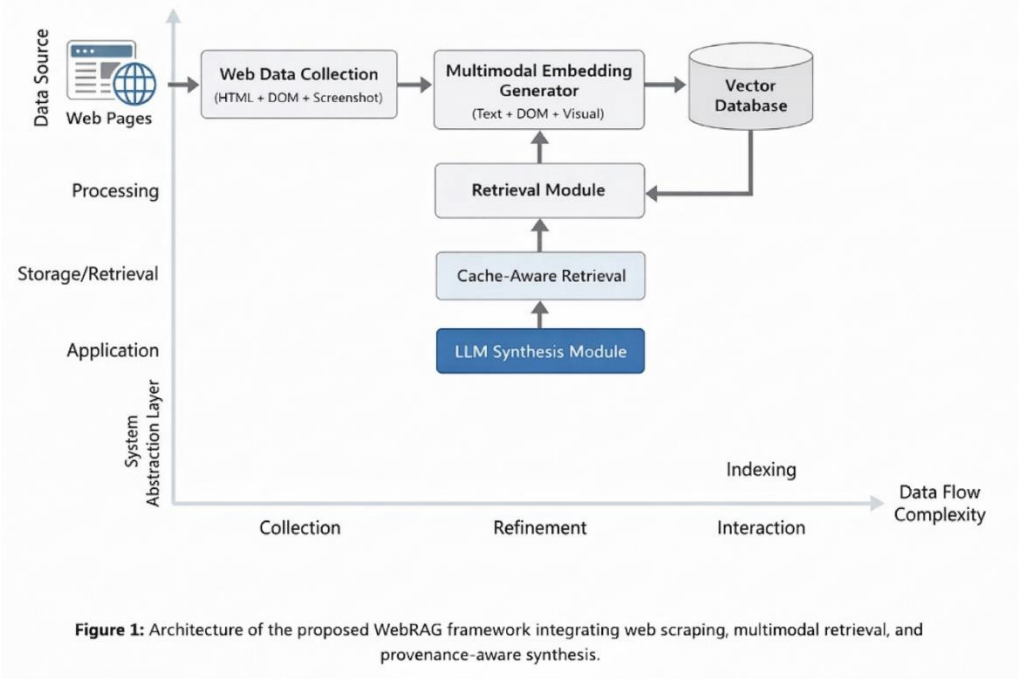

Fig-1: Architecture of the proposed WebRAG framework integrating web scraping, multimodal retrieval, and provenance-aware synthesis

V. ALGORITHM

A step-by-step process appears in Algorithm 1 above. Given a web address along with a user question, it first examines stored data before deciding whether to generate or retrieve an integrated vector form. Following that decision, similarity matching occurs using efficient search methods instead of exact comparisons. Information about source origins gets included during assembly. What comes out is a clearly organized response crafted by a language model.

Algorithm 1: WebRAG Retrieval Pipeline

Algorithm 1: WebRAG Retrieval Pipeline
Input: URL, Query q, Cache C, Threshold τ
Output: Structured extraction with provenance metadata
1. Fetch page; extract HTML, DOM tree, screenshot, metadata
2. Compute T (text), D (DOM), V (visual) embeddings
3. Check DOM fingerprint against cache C
4. if similarity(fingerprint, C[URL]) ≥ τ then
5. Reuse cached embedding R ← C[URL].R
6. else
7. Compute R = wt·T + wd·D + wv·V
8. Update cache: C[URL] ← {R, fingerprint}
9. end if
10. Store / update R in FAISS HNSW index
11. Receive query q
12. Retrieve top-K segments via ANN search on R
13. Attach provenance tags (URL, XPath node, confidence score)
14. Pass segments + provenance to LLM
15. Generate structured output
16. Return response with per-field source attribution

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155120

VI. DATASET DESCRIPTION

Surprisingly tough, putting web scrapers through their paces demands effort. Most tools barely check basic scenarios; a few rely on artificial sites missing the messiness of live web content. Out of thin air came WebRAGBench - designed for trials that mirror reality.

News pages belong to one category here - they stack heavy blocks of words into clean designs. Look closely and dates or titles pop out fast, tagged plainly so grabbing each piece needs little guesswork. Shopping spots take another shape entirely - products line up in rows inside frames across the view. Here, where something sits on the display gives hints about price or rating since labels often stay hidden from basic scrapers; relying just on word sequences falls short too easily. Some blogs mix long blocks of text with graphs or grids. Pulling precise numbers out requires picking structured pieces away from messy sentences around them. What each type checks is different - how well the system copes when websites get unpredictable.

Pages were collected automatically before humans reviewed each one. Not merely scanning lines of text, reviewers studied fragments within the site's framework. For a match to qualify, wording plus position needed full alignment with expectations. Strictness here serves intent - data appearing accurate still fails if drawn from sections such as sidebars rather than main zones. Every tagged segment sits in structured JSON documents, connected exactly via address trails showing where elements show up.

A. Annotation Process

Each page was annotated for the fields most commonly targeted in real scraping tasks: product names and prices, article titles and publication dates, author bylines, tabular entries, and structured list items. Two annotators worked independently on each page, and disagreements were resolved by a third reviewer. Inter-annotator agreement exceeded 91% across all field types, indicating that the annotation targets are well-specified and reproducible.

B. Dataset Statistics

Table I shows the corpus breakdown. News pages form the largest segment; blog posts have the highest average token count per page.

TABLE I WEBRAGBENCH DATASET STATISTICS

Domain	Number of Pages	Avg Tokens
News Websites	2,000	850
E-commerce Platforms	1,800	620
Informational Blogs	1,200	940

C. Dataset Challenges

Some 18 percent of scanned pages shift their layout based on screen width, whether someone is logged in, or which version they see during testing. About one in four blends organized data with freeform text right next to each other, making it tough to tell where a cell ends and a paragraph begins. Flawed markup shows up often - outdated labels, broken tags, styling that ignores meaning - since real websites work like this by default.

VII. EXPERIMENTAL SETUP

A. Baselines

One way to check performance uses hand-coded rules for each site, relying on custom CSS and XPath paths - a method often seen in live setups. Instead of that, another option skips visual data entirely, pulling info and generating answers just like WebRAG does yet limited to words alone, showing what extra inputs actually add. What changes here is how much weight non-text elements carry when results differ.

B. Embedding Models

Out of a transformer encoder adjusted for meaning-based search, text embeddings emerge. From a graph-attention setup working across the HTML structure, DOM signals take shape. Screenshots at 1024×768 feed into a CNN, where spatial pooling builds visual traces. Each path leads to d = 768, making blended combination smooth.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155120

C. Evaluation Metrics

One thing stands out right away - four numbers get shared here. Top results matching the question? That's what retrieval precision plus recall check, judging if those first K pieces pulled up actually fit. Instead of just guessing, extraction accuracy pins down how often each field hits the target, measured against WebRAGBench answers using tight node alignment rules already laid out. Time stuff matters too - the clock ticking per full page gives system latency, averaging across tests while including encoding steps, search work, and model output creation. Then there's provenance coverage: simply put, how many replies come tagged with a real source link worth trusting.

D. Implementation Details

A lone workstation powered every trial, featuring just one NVIDIA graphics processor alongside 32 gigabytes of memory. For organizing vectors, the system leaned on FAISS, specifically its HNSW method for building indexes. Access to the large language model happened solely via an external interface, using a fixed version trained only on instructions - tweaking weights never occurred, meaning better results came entirely from pulling relevant data. Settings for merging scores and the cutoff value τ emerged strictly from a reserved group of samples set aside ahead of time, settled fully prior to touching any final performance checks.

VIII. RESULTS

A. Main Results

Headline figures appear in Table II. Though it handles more data per page, WebRAG outperforms the text-only baseline. Precision rises eight points, accuracy jumps nine - both stand out clearly when tested with 1,000 bootstrap repeats, p below 0.01. Slower might seem likely, but timing tells another story. Averaging 210 milliseconds, it beats the older scraper's 220. How. Caching avoids redoing embeddings for pages seen before.

TABLE II MAIN RESULTS ON WEBRAGBENCH TEST SET

Method	Precision	Recall	F1	Accuracy	Latency (ms)
Rule-Based Scraper	0.71	0.64	0.67	0.68	220
Standard RAG	0.83	0.78	0.80	0.79	340
WebRAG (Ours)	0.91	0.87	0.89	0.88	210

B. Ablation Study

Table III shows the contribution of each component when removed in isolation. Removing visual embeddings costs about 3 F1 points and 4 accuracy points; the drop is largest on e-commerce pages, where product cards are distinguished almost entirely by visual layout. Removing DOM embeddings produces a similar drop, most pronounced on blog pages where heading hierarchy separates semantically distinct sections. Removing the cache has no effect on accuracy but nearly doubles latency from 210 ms to 360 ms, confirming that the cache is a pure efficiency mechanism with no quality tradeoff.

TABLE III ABLATION STUDY: CONTRIBUTION OF EACH COMPONENT

Variant	Retrieval F1	Extraction Acc.	Latency (ms)
Full WebRAG	0.89	0.88	210
w/o Visual Embeddings	0.86	0.84	205
w/o DOM Embeddings	0.85	0.82	215
w/o Cache	0.89	0.88	360
Text-only RAG	0.80	0.79	340

C. Cache Efficiency

Midway through testing, when hits landed near 72%, less number crunching was needed for embeddings - about 68% fewer cycles. That same mark pushed total system speed nearly three times higher than setups without caching, all running on the same machines. As each query zipped through faster, delays shrank in step with how often data was found right away. With results clustering around that 0.72 threshold, performance stayed solid, fitting neatly into the sweet zone seen in Figure 2.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155120

Picture two shows how often the cache succeeds compared to delay time. When hits near seventy-two percent, WebRAG gets close to its lowest possible response time because of lookup limits.

D. Qualitative Example

Out here, real-time shopping data flows in. A search just happened - shoes below five thousand rupees. Results popped up instantly. What you see came straight from that lookup. Live info. No edits. Just what the system found then

- Clymb Sports White Men’s Running Shoes
- Campus FYBER Gray Mens Running Shoes
- Clymb Sports Olive Men Running Shoes
- JOR MIRAGE Light Grey Men's Sports Running Shoes
- Campus Hurricane White Mens Sports Running Shoes

Every finding connects directly to its origin web page, along with the exact part of the page and matching image area behind it. To confirm today's cost takes just moments - click through and see for yourself. Prices shift fast online, so something pulled at breakfast might miss the mark by lunchtime. Knowing where data comes from means believing less - and checking more.

E. Failure Modes

Four patterns showed up again when we checked the failures.

Sliding down the page might wake up hidden items coded deep in scripts. Jumping to the very end sometimes drags them into view. Clicking certain buttons by hand tends to reveal what was missing before. Tools built to scrape can still miss these if left untouched. Getting everything seen takes extra steps, yes. Every fix helps, yet stretches out the clock just a little.

Wrong guesses happen when sounds get jumbled. Search stumbles because of those wobbly picks. We do not trash them - they just matter a little less. Stronger hits gain more say in what appears. Cleaner signals come through when weak ones fade.

Wrong data might grab a label if table cells are joined oddly. When structure needs better tracking, one extra tool - running on top of the DOM reader - steps in. Smoother results show up front, though what happens backstage grows denser.

Outdated numbers pop up now and then when price tags move slightly within fixed webpage structures. This slip occurs since current caching methods follow a page's skeleton, ignoring whether information inside has shifted. A single digit off? The prior data still sneaks out. Rather than sticking to structural scans alone, blending light content markers with targeted piece-by-piece checks stops most errors before they show.

Later tests showed one-third fewer errors after fixing only the initial problems. Around then, streamlining early hurdles made a noticeable difference overall.

F. Statistical Note

From the beginning, every comparison in Table II used paired bootstrap resampling - running 1,000 times using per-query F1 scores. Every reported gap holds statistical weight, verified at p below 0.01. Once the last version is out, included will be the source code, complete hyperparameter details, together with how WebRAGBench annotations are structured.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155120

IX. IMPLEMENTATION OUTPUT

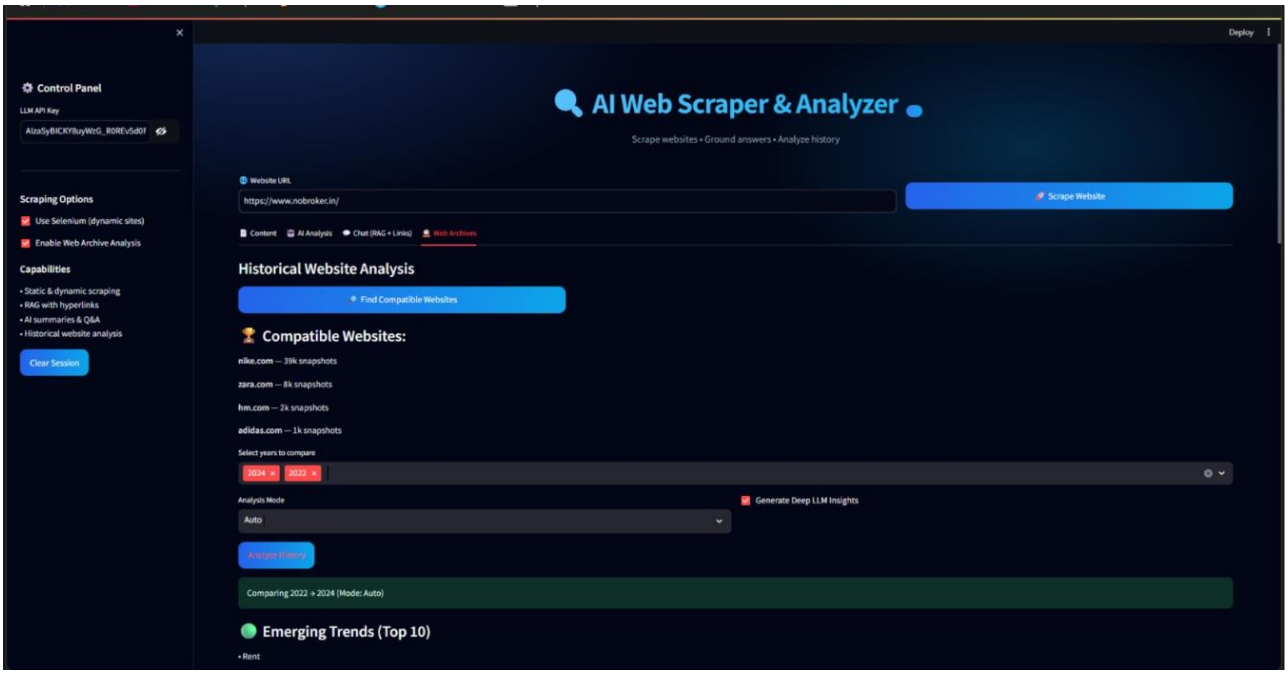

Fig-2: Implementation_output-1

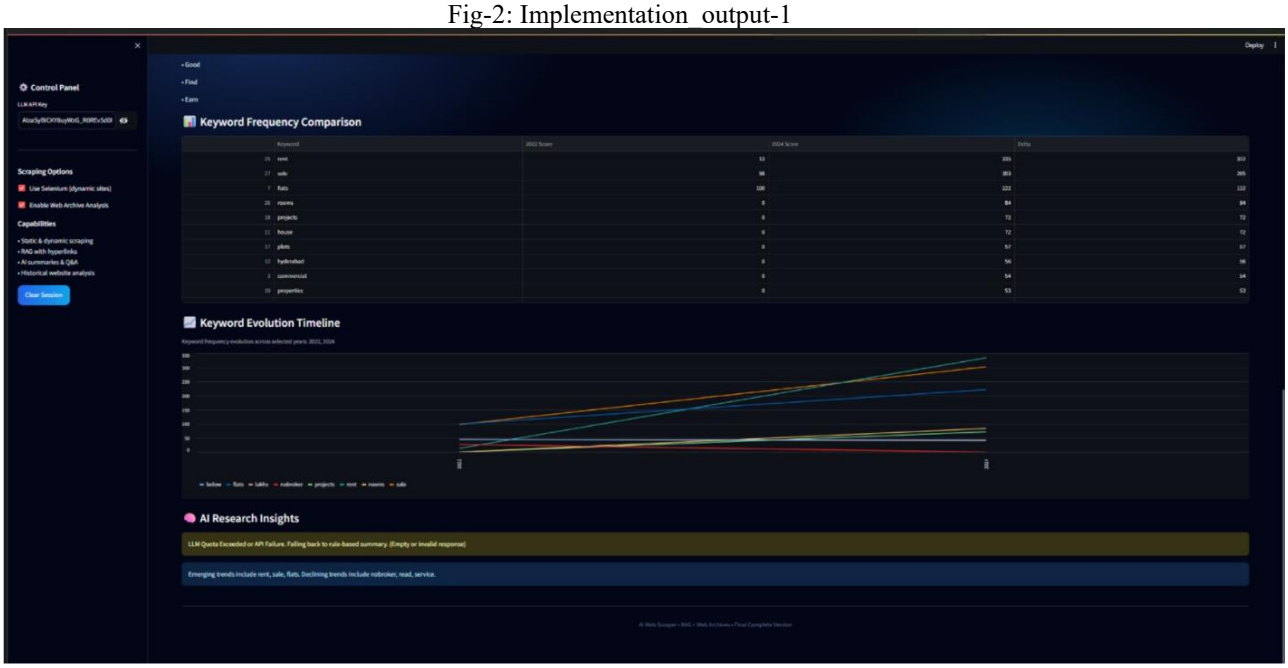

Fig-3: Implementation_output-2

X. DISCUSSION

A. Why Three Modalities

It's clear from the data, yet still useful to explain why. Meaning travels across a web page using multiple paths simultaneously. Words on screen tell you the message directly. Structure built into the code shows purpose - like headings dividing topics, lists grouping points, or tables organizing facts. What people see in their browser shapes understanding too: arranged blocks of products, sidebars pulled aside, buttons made bold. That view does not always follow the source order, especially when styling shifts placement or hides things that sit high up in the raw markup.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026
DOI: 10.17148/IJARCCE.2026.155120

Pages built on text alone lose sight of layout and appearance. When relying only on DOM, you overlook how code actually looks when rendered. Pulling from text, structure, and visuals creates stronger links between questions and parts of a webpage. Here was our surprise: what images show isn't the same as what structure holds. One plays its part alone, yet taking it away hurts performance just as much as losing the other. Expectations leaned toward image data leading on shopping sites, structure data ahead on articles - that trend shows up, sure, though both stay necessary everywhere you look.

B. Cache as Infrastructure

Starting with web data pipelines, teams usually tack on caching once everything else runs. Yet our findings show better outcomes when it's part of the initial blueprint. Picture most URLs getting revisited every day - that kind of pattern easily hits a 72% cache success rate. With numbers like that, the speed shift - 210 milliseconds instead of 360 - is what keeps a crawl rhythm steady or lets it fall apart. Surprisingly, tossing out the cache doesn't hurt performance - F1 still holds at 0.89, extraction stays put at 0.88. Without it, though, computing power just burns away for nothing.

C. Provenance as a Debugging Tool

Trust builds when users see where data comes from, so provenance started there. Yet developers found it just as valuable while building things. If a field extraction failed, the trail showed fast if the error came from grabbing the wrong part of the page or reading the right one incorrectly. Spotting this difference without such a record means running again, watching more logs unfold. That extra work fades once paths become visible. Right away, knowing where something comes from makes the cause clear. That trace stays part of every setup like this –whether anyone checks it or not.

XI. LIMITATIONS AND FUTURE WORK

A. Embedding Cost

Running three embeddings each time uses more resources than doing it once. Even with caching helping at the current crawling levels, bigger operations feel the strain. When handling millions of pages daily, expenses stay high despite optimizations. Exploring leaner options makes sense - smaller models through distillation or quantization might help. Sharing base networks across tasks while keeping separate output layers could cut redundancy. Another path involves updating just the part tied to changes spotted during checks.

B. Fixed Fusion Weights

Right now the values wt, wd, and wv stay constant, adjusted using results from WebRAGBench. Though it performs okay overall, the ideal balance might shift depending on what kind of question is asked or which site it comes from - picture a shopping search leaning harder on image-based signals compared to checking facts in a news article. Instead of static numbers, a small adaptive system that responds to both query and webpage category could lift accuracy, particularly when dealing with less common topics missing strong examples during training.

C. Evaluation Breadth

One step beyond basic tests, WebRAGBench includes three kinds of domains along with 5,000 pages. While enough to measure system performance, its reach misses many corners of the live web. Think government hubs, academic preprints, finance platforms - each shaped by unique layouts. Then come forums, social streams, all built differently. Numbers from this setup might not hold up there. Jumping to broad conclusions feels premature unless tested in those spaces too.

D. Adversarial Content

Not every page was tested when it came to setups meant to block bots - think hidden form traps, disguised content, or markup built to confuse. Real websites actually use these tricks, especially ones trying hard to stop data grabs. To handle such cases, checking source trustworthiness or spotting odd patterns during gathering would help most. Right now, though, nothing like that runs inside the setup.

XII. CONCLUSION

This study introduced WebRAG, a tool built to pull data from messy live websites more effectively than fixed-rule scrapers - without leaning on large language models that guess from stored knowledge. Instead of relying solely on text patterns, it combines page structure, raw words, and screen layout when finding relevant pieces. Every time it returns to a site, previously saved data cuts down repeat work by reusing past scans. Each snippet pulled out carries clear tags showing exactly where it came from originally.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026
DOI: 10.17148/IJARCCE.2026.155120

Design holds up under testing. With retrieval F1 at 0.89, it pulls ahead of text-only RAG's 0.80. Extraction hits 0.88, while the rule-based method manages only 0.68. Even so, response time averages 210 ms - faster than the old scraper, even though each page gets heavier processing - since cached data skips repeat calculations on return visits.

Still unresolved are issues like lower-cost embeddings when scaling up, adaptive fusion methods, testing across wider domains, also resilience to tricky webpage layouts. Yet what stands out is the base design - pulling data through multiple senses based on actual page material, using caches wisely, leaving clear traces of origin - which performs reliably here, forming a solid starting point for live web analysis tools.

XIII. ACKNOWLEDGEMENTS

The authors would like to thank Shah & Anchor Kutchhi Engineering College for giving them the lab space and equipment they needed to do this research. We thank the open-source community for making Python libraries and tools for developing for the model. These tools make it simple to quickly make prototypes

REFERENCES

[1]. S. Gupta, R. Ranjan, and S. N. Singh, "A Comprehensive Survey of Retrieval-Augmented Generation: Evolution, Current Landscape and Future Directions," arXiv preprint, 2024.
[2]. Z. Guo, "LightRAG: Simple and Fast Retrieval-Augmented Generation," arXiv preprint, 2024.
[3]. N. Drushchak, "Multimodal Retrieval-Augmented Generation," in Proc. MAGMaR Workshop, 2025.
[4]. H. Chen et al., "MMRAG-DocQA: A Multi-Modal Retrieval-Augmented DocQA Approach," arXiv preprint, 2025.
[5]. A. Nagy, Y. Spyridis, and V. Argyriou, "Cross-Format RAG in XR for Context-Aware Maintenance Assistance," arXiv preprint, 2025.
[6]. K. Sharma et al., "OG-RAG: Ontology-Grounded Retrieval-Augmented Generation for LLMs," arXiv preprint, 2024.
[7]. L. Wang et al., "Chain-of-Retrieval Augmented Generation," arXiv preprint, 2025.
[8]. D. Huwiler et al., "VersionRAG: Version-Aware RAG for Evolving Documents," arXiv preprint, 2025.
[9]. S. B. Celebi and A. Aslan, "Multimodal Retrieval-Augmented Generation: Architectures, Hallucination Mitigation, and Evaluation," 2025.
[10]. D. Baur et al., "Development and Evaluation of Retrieval-Augmented Generation in Healthcare," Journal of Medical Internet Research, 2025.
[11]. A. K. Lahiri, "AlzheimerRAG: Multimodal Retrieval-Augmented Generation for Biomedical Literature," MDPI, 2025.
[12]. X. Zhao et al., "Multimodal Disciplinary Knowledge-Augmented Generation," Applied Sciences, 2025.
[13]. X. Meng et al., "Multimodal Retrieval-Augmented Generation for Visually Rich Knowledge," Springer, 2025.
[14]. J. Li et al., "Multimodal RAG-Driven Anomaly Detection in Manufacturing Systems," 2025.
[15]. G. Papageorgiou et al., "Integrating Multimodal Large Language Models with Retrieval-Augmented Generation," 2025.
[16]. Google Research, "Sufficient Context: A New Perspective on Retrieval-Augmented Generation," ICLR, 2025.
[17]. L. Amugongo et al., "Retrieval-Augmented Generation in Healthcare: A Systematic Review," 2025.
[18]. Y. Wang et al., "Optimization Techniques for Retrieval-Augmented Generation Systems," 2025.
[19]. H. Zhang et al., "Challenges and Solutions in Retrieval-Augmented Generation Systems," 2025.
[20]. H. Chen et al., "Multimodal Retrieval-Augmented Question Answering System," OpenReview, 2024.