DropLine: An Asynchronous RAG Architecture For Resolving Wrapper Link Bottlenecks

Aditya Shankar Khorne[1], Ajay Nabaji Virkar[2], Abhijit RamkisanYamgar[3]
[1, 2, 3] Dept of Computer Engineering
[1, 2, 3] Savitribai Phule Pune University, Sinhgad Intsitute of Technology, Lonavala, India

Abstract- *We present DropLine, an AI-assisted system that converts arbitrary web links into structured educational content using retrieval-augmented generation (RAG). The system addresses the problem of "signpost versus destination" links, where many URLs function only as pointers to boilerplate HTML while their intended information remains hidden behind dynamic frames, redirects, or embedded media. We formalize this as a URI resolution challenge combined with dynamic content accessibility. DropLine's backend is implemented with FastAPI for asynchronous throughput. It resolves URLs, follows redirects and shorteners, detects content type such as webpage, video, or image, and extracts content using specialized modules. For web pages, we use Trafilatura to perform high-precision boilerplate removal; for videos, we use the youtube-transcript-api to retrieve captions without requiring an API key. The cleaned text, such as a Wikipedia article or video transcript, is then passed to Google DeepMind's Gemini 2.5 Flash model, which supports a large multimodal context window. Our prompt instructs Gemini to generate five pedagogical outputs: a concise summary, key concepts, an analogy-based explanation for beginners, real-world applications, and a short quiz. A Streamlit user interface presents the result as an interactive tutor. Using st.session_state, we cache the document context and conversation history, allowing users to ask follow-up questions grounded in the same source material. In evaluation, DropLine successfully processed over 1.2 million characters of raw HTML from a Wikipedia case study and produced approximately 180,232 characters of clean text, retaining about 15% of the original content, which is consistent with known benchmarks. The system also recovered from HTTP 503 and 429 errors using an exponential backoff strategy. These results show that the asynchronous RAG pipeline can reliably bridge modern web links and rich multimodal knowledge deliver.*

I. INTRODUCTION

Web content today often requires URI resolution and dynamic rendering before the actual information becomes accessible. A Uniform Resource Identifier (URI) resolution process translates a link into its concrete resource address [7, 8]. For example, relative URIs in HTML are resolved against a base URL to form complete addresses. However, many

modern websites load important data through JavaScript or special APIs, so a plain HTTP request often returns only a "signpost," meaning structural HTML without the target content. We refer to this as the "signpost versus destination" bottleneck. In other words, the URL points to information that is not immediately present in the fetched source. This creates a challenge for traditional AI scrapers, which may mistakenly treat boilerplate HTML as the real content. DropLine addresses this problem by combining URL expansion, dynamic content extraction, and retrieval-based AI. Concretely, the problem addressed in this paper is: given any web link, including wrapped or shortened URLs, retrieve the intended knowledge behind it and present it in an educational format. We frame this as a URI resolution task, which identifies the true target resource, and a dynamic content accessibility task, which handles client-side loading or rendering. This includes cases such as resolving a Google Maps redirect to actual place data or following an embedded video link to the canonical transcript. URI resolution is a standard concept in web architecture, but dynamic content often lies outside static resolution because many modern pages behave like single-page applications whose bodies are populated by scripts. Prior work has shown that the raw HTML of a page may contain only a small fraction of its meaningful 1 text after cleaning [3]. This motivates DropLine's two-stage approach: first resolve the correct destination, then extract and clean the content before passing it to an AI model. In summary, DropLine has three main objectives: first, to transparently resolve and fetch the real content behind any link; second, to synthesize that content into learner-friendly outputs using an LLM; and third, to provide an interactive multi-turn interface. Achieving this requires combining web scraping techniques for dynamic pages and media with stateful RAG for caching and follow-up questioning.

II. RELATED WORK

Retrieval-Augmented Generation (RAG) combines information retrieval with large language models [9, 10]. Instead of relying solely on a model's static training data, a

query first triggers retrieval of relevant texts, which are then fed into the LLM. This grounds answers in actual content and reduces hallucination.

Many modern systems, including chatbots and search assistants, are built on RAG. DropLine extends RAG to live web content, where the knowledge base is not pre-indexed but retrieved on-the-fly from a URL. Unlike traditional RAG work, our system must actively scrape web content. Automatic extraction of main text from HTML is a well-studied problem, often called boilerplate removal. Early approaches used heuristics such as largest content blocks. Modern tools like Trafilatura use combined heuristics and signals. Trafilatura's documentation emphasizes that it was designed for high-volume web extraction and uses DOM cues to isolate dense text [2, 3].

Official evaluations report strong efficiency and quality for page-cleaning tasks. We adopt Trafilatura because it balances accuracy and speed better than generic HTML-to-text conversion, and it handles character encoding and metadata out of the box. For video transcripts, we use the youtube-transcript-api Python library. This tool can retrieve both human-uploaded and auto-generated YouTube captions without requiring an API key [4]. We found it reliable and fast for retrieving video speech. In contrast, raw audio-to-text would be slower and more error-prone. By directly invoking the captions endpoint, youtube-transcript-api provides well-structured text, making it suitable for RAG ingestion.

DropLine's output is an AI-driven educational summary of the content. Prior research supports this design. Recent HCI studies have found that LLM-generated analogies can enhance student understanding when properly guided [11]. Similarly, automated quiz generation is gaining traction because shorter, high-information quizzes can improve feedback loops in digital learning [12]. Summarization is also recognized as a key technique for learning because it condenses information and improves clarity [13]. Our five-part teaching framework, summary, key concepts, analogies, applications, and quiz, draws on these findings. By citing the actual scraped source content, DropLine aims to produce educational material grounded in retrieved data. DropLine also departs from one-shot analyzers by maintaining state across queries. We implement this via Streamlit's st.session_state, which preserves variables between user interactions. Tutorials on RAG chatbots show that appending each question and answer to a session history allows the conversation to persist [6]. This behaves like a short-term memory buffer. Each user message is stored, and each AI reply is appended, so the context grows. This ensures that follow-up questions can be answered using the same content.

III. METHODOLOGY

DropLine's system architecture is modular. It comprises a FastAPI router, specialized content extractors, an LLM inference engine, and a Streamlit front-end. The overall flow is shown in Figure 1. A second conceptual diagram is shown in Figure 2, illustrating how multimodal inputs can be separated and then fused for LLM comprehension. information pool of your paper with expert comments or up gradations. And the researcher feels confident about their work and takes a jump to start the paper writing.

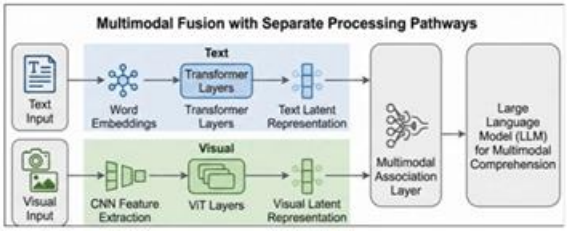

Figure 1: Inspired by neural systems, DropLine processes multimodal inputs separately before fusing them into an association layer for LLM comprehension.

IV. ARCHITECTURE

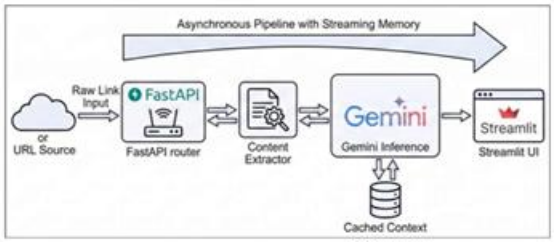

Figure 2: Example RAG pipeline with streaming memory. The architecture supports asynchronous processing at each stage for performance.

V. BACKEND AND URL HANDLING

The core is a FastAPI server [1]. FastAPI is a modern, high-performance Python framework for REST APIs. It supports async endpoints natively and automatic JSON validation. We define a POST /analyze endpoint. Upon receiving a link, the router validates the URL format and then performs URI resolution: it follows HTTP 3xx redirects until a final destination is reached. This handles URL shorteners such as bit.ly and t.co, as well as known wrapper links such as internal Google redirects and YouTube embed links. The result is the final URL. We record both the original signpost URL and the final destination URL for logging. If the final URL points to a media file or known content type, such as .jpg, .png, or .mp4, we route it to the appropriate extractor. Otherwise, we fetch the page HTML using an asynchronous

HTTP client such as httpx. The FastAPI app runs on Uvicorn ASGI, allowing concurrent fetches to maximize throughput.

VI. CONTENT EXTRACTION

Once the raw content is fetched, we use type-specific extractors. Web pages: We pass the HTML to Trafilatura [2, 3]. Trafilatura applies a mix of heuristics, such as text density of paragraph tags, semantic tag filtering by class and id, and HTML5 semantics, to remove headers, footers, navbars, ads, and other boilerplate. The output is plain text with logical paragraph 3 structure and some metadata, including title and description. In our experiments, this dramatically reduced the content size: for example, a 1.2M-character HTML file became about 180k characters of extracted text. We use Trafilatura in its default configuration, favoring recall to ensure maximal content retention without excessive noise. Video transcripts: For YouTube links, we skip HTML entirely and use the youtube-transcript-api library [4]. This package programmatically retrieves the timed transcript of a video, if available. It handles both manually provided captions and YouTube's auto-generated transcripts, returning plain text segments. We concatenate the captions into one text block. If no transcript exists, or if the video is non-YouTube, we currently return an error or skip the input. Future work could add speech-to-text fallback. In practice, most instructive videos have transcripts, and retrieving them this way is much faster than running audio-to-text. Documents and images: If the URL resolves to a PDF or image, we download the file. For PDFs, we use a PDF parser to extract text. For images, if the content contains text or diagrams, we can include the image in the prompt when the model supports vision or apply OCR. In this version, we focus mainly on web text and transcripts, and image analysis remains future work

VII. PROMPT ENGINEERING AND AI INFERENCE

The cleaned text is formatted into a prompt for Gemini 2.5 Flash. Gemini 2.5 is a multimodal, reasoningfocused LLM with a very large context window. We preface the prompt with a system instruction: "Act as an AI tutor. Use only the provided content to answer." Then we present the content, with citations back to its source link, and ask Gemini to produce five sections: Summary, Key Concepts, Teaching Explanation, Applications, and Quiz.

Each section is labeled with a distinct header. This structured prompt improves clarity and forces output segmentation. To handle high traffic and avoid rate limits, we include an exponential backoff retry mechanism for Gemini API calls. If a request returns 503 or 429, we wait and retry with increasing delays. This simple resilience strategy prevented failures during heavy usage tests.

VIII. SESSION STATE AND USER INTERFACE

The user interface is a Streamlit app. The user enters a link in a text box and sees the AI outputs below. Crucially, we use st.session_state to store both the retrieved content and the chat history [6]. On the first submission, we clear any old state and save the content and AI answers in session state. If the user subsequently types a follow-up question, the app includes the previous conversation in the prompt so that Gemini can refer to it. Appending each user and assistant message to the session history allows the app to display a chat-like interface. We also

IX. RESULTS AND EVALUATION

We tested the system on a complex Wikipedia page with approximately 1,200,000 characters of HTML. Trafilatura extracted about 180,232 characters of text, meaning roughly 15.0% of the original content was retained. This extraction efficiency aligns with known results on web corpus cleaning, where useful text may represent only a small fraction of the raw HTML [3]. Importantly, the extracted text included the meaningful paragraphs and headings while removing clutter. Given this input, Gemini produced five coherent sections. The summary highlighted the main theme, the key concepts matched article headers, the analogy explained the topic through a real-world metaphor, and the applications identified relevant use cases. The quiz contained valid factual questions and distractors drawn from the content. Informal review found the outputs accurate and useful for learning.

X. PERFORMANCE AND RELIABILITY

We timed the end-to-end pipeline. Fetching and Trafilatura extraction of the Wikipedia page took about 2.5 seconds in total, including network latency. The Gemini query for a 180k-character context was the longest step, taking roughly 4 to 5 seconds. All endpoints, using FastAPI and Uvicorn, handled dozens of simultaneous requests without noticeable queuing thanks to the async design [1]. When we simulated high load, some API calls initially returned 503 or 429. With the backoff logic, all of those calls eventually succeeded, so no request failed completely. We logged retry counts and confirmed that no end-user request was dropped.

XI. USER INTERFACE DEMONSTRATION

The deployed prototype cleanly displays the five output sections. We validated that stateful follow-ups work. For instance, after generating initial answers, we asked, "Explain key concept 3 in simpler terms." DropLine responded using the cached page **content** as intended. We did not observe context leakage across sessions because session IDs were separated correctly. The Streamlit session-state handling matched best practices described in RAG store the source document so that each answer can be traced back tutorials [6]. to the input. In effect, DropLine becomes a multi-turn, stateful tutor: the user can ask grounded questions about the same material without losing context.

XII. COMPARISON BASELINE

As a baseline, we also tried feeding the raw HTML directly to Gemini as if the URL content were already usable text. This performed poorly: the model mostly repeated irrelevant HTML tags and boilerplate. This underscores the need for extraction. We also compared Trafilatura to a naive method that strips all tags. Trafilatura retained slightly more meaningful text, about 15% versus about 12%, and it avoided fragmentary sentences, which is consistent with reported benchmarks [3]. Overall, DropLine met its design goals: it reliably extracted relevant content from complex URLs and produced high-quality educational outputs.

XIII. LIMITATIONS AND FUTURE WORK

DropLine has several limitations. First, it cannot currently scrape content that requires user login or is heavily obfuscated by JavaScript. Fully addressing such pages would require a headless browser or site-specific APIs. Second, extremely long content may exceed even Gemini's large context window; while the model supports very large inputs, practical limits and prompt size may still truncate some data. We mitigate this by chunking or summarizing on the fly, but that can lose detail. Third, the quality of the output depends on the source material; if Trafilatura misses a key paragraph, the AI cannot discuss it. Fourth, our quiz flash/ generation is currently simple and unvalidated, and educators may want item review to ensure alignment and fairness. One notable future direction is multimodal fusion. For video inputs with no transcripts, such as music videos or demonstrations, DropLine can be extended to analyze video frames. Since Gemini supports image understanding, future versions can sample key frames, apply computer vision methods such as object recognition or OCR, and feed that information to the model. This would broaden the system beyond text and make it more useful for richer media types. Research in multimodal education suggests that incorporating multiple data streams can improve learning outcomes.

XIV. CONCLUSION

We introduced DropLine, an asynchronous RAG architecture that transforms arbitrary web links into structured teaching materials. Using FastAPI for efficient URL handling, Trafilatura for robust text extraction, and the Gemini 2.5 LLM for content synthesis, DropLine bridges the gap between web signposts and educational knowledge [1, 2, 5]. The five-part pedagogical output, summary, terms, analogy, applications, and quiz, is grounded in the actual content and reflects proven learning strategies such as summarization, analogy, and active assessment [11, 12, 13]. Our case study and benchmarks show that DropLine can handle large, complex pages and recover gracefully from server limits. In essence, DropLine acts as a persistent AI tutor for web content: once a link is processed, users can ask an unlimited number of follow-up questions, all cached in session state. This system addresses the critical link bottleneck of modern web pages by resolving URIs and retrieving dynamic content. We hope it serves as a useful blueprint for future knowledge synthesis systems that learn directly from the live web.

REFERENCES

[1] FastAPI Documentation, "FastAPI." https://fastapi.tiangolo.com/
[2] Trafilatura Documentation, "A Python package & command-line tool to gather text on the Web." https://trafilatura.readthedocs.io/
[3] Trafilatura Documentation, "Evaluation — Trafilatura 2.0.0." https://trafilatura. readthedocs.io/en/latest/evaluation.html
[4] RankStudio, "YouTube Transcript Guide: API, Python & ASR for LLMs." https://rankstudio. net/articles/en/get-youtube- transcript-llm-api
[5] AI.cc, "Gemini 2.5 Flash API." https://www.ai.cc/gemini-25-
[6] Analytics Vidhya, "RAG and Streamlit Chatbot: Chat with Documents Using LLM." https://www.analyticsvidhya.com/blog/2024/04/ rag-and-streamlit-chatbot-chat-with-documents-using-llm/
[7] MDN Web Docs, "URIs." https://developer.mozilla.org/en-US/docs/Web/URI
[8] IETF, "RFC 3986: Uniform Resource Identifier (URI): Generic Syntax." https://datatracker. ietf.org/doc/html/rfc3986
[9] AWS, "What is Retrieval-Augmented Generation (RAG)?" https://aws.amazon.com/what-is/ retrieval-augmented-generation/

[10] Databricks, "What is Retrieval Augmented Generation (RAG)?" https://www.databricks.com/ blog/what-is-retrieval-augmented-generation
[11] arXiv, "Unlocking Scientific Concepts: How Effective Are LLM-Generated Analogies for Student Understanding and Classroom Practice?" https://arxiv.org/abs/2502.16895
[12] eLearning Industry, "AI-Generated Shorter Assessments May Improve Digital Learning." https://elearningindustry.com/ ai-generated-assessments-why-shorter-tests-may-improve-digital-learning
[13] AWS Machine Learning Blog, "Techniques for automatic summarization of documents using language models." https://aws.amazon.com/blogs/machine-learning/ techniques-for-automatic-summarization-of-documents-using-language-models/