VIETNAM NATIONAL UNIVERSITY

UNIVERSITY OF ENGINEERING AND TECHNOLOGY

ENTERPRISE INTERNSHIP REPORT

Project Title:

AI-Driven Multi-Source Information Fusion

Student: Ngo Van Kiet – 22022643

Advisor: Dr. Nguyen Viet Cuong
Mr. Nguyen Duc Manh

Hanoi, 2025

Acknowledgements

I would like to extend my sincere gratitude to **Dr.Nguyen Viet Cuong**, and **Mr.Nguyen Duc Manh**, my supervisers, for their valuable guidance, continuous support, and encouragement throughout the entire duration of this internship.
This project has been an incredibly rewarding experience, enhancing both my technical skills and understanding of low code platforms and AI integration.
Working with my advisors has been an enriching experience. I have learned a great deal, especially in staying updated with the current and future landscape of the AI industry.
I have developed a comprehensive understanding of how systems should work in reality and how to effectively debug when errors occur.
After the time of internship, I may develop a hard skill on workflow Automation and a flexible mindset of problem solving, thanks to all the comments and advices during this course.

Contents

RielFezzTool

A Study Of Applying AI On Multisources Information Fusion

Student	Advisor	Co-Advisor
Ngo Van Kiet – 22022643	Dr. Nguyen Viet Cuong	Mr. Nguyen Duc Manh

My Project Section Github

Abstract

This project presents a multi-source, multilingual summarization and retrieval system that aggregates real time news articles from diverse online platforms in real time. The system enables users to query and access information through a simple, user-friendly interface while maintaining cost-efficiency and scalability. To achieve this, the solution leverages open-source and low-code technologies, including workflow automation with n8n, vector-based storage in PostgreSQL with pgvector, and a Streamlit-powered frontend for interaction. At its core, the system integrates local AI models deployed via Ollama to handle tasks such as preprocessing, summarization, and semantic search, thereby minimizing dependency on commercial APIs and reducing operational costs. By supporting multilingual data ingestion and real-time chatbot, the platform enhances accessibility for everyone despite their languages. The ultimate goal of this work is to provide a cost-effective, open-source pipeline that delivers rapid, reliable, and context-aware access to knowledge in an era of information overload.

Keywords: Crawling, Processing, RAG, Local AI Models, embedding, multilingual, open-source, price, latency, real-time

1 Introduction

In today's digital age, the way people receive information has changed significantly. Traditional media such as newspapers, books, and television are no longer the main sources of news. Instead, people now rely on online platforms that update continuously. This shift has created an overwhelming amount of information available on the Internet.

While this provides easy access to knowledge, it also brings challenges. Readers are often overloaded by repetitive articles, reliability, and time consuming reading articles. As a result, selecting the right information quickly has become an important issue.

To address this situation, it is important to clearly define the research problem, the challenges that come with it, and the objectives of this work.

1.1 Problem

At present, people can access a very large amount of information from many platforms and websites. However, searching and comparing this information manually is often complex and time-consuming, especially when some important sources are only available in other languages. There are already many news websites and tools that try to solve this problem. They collect and present information to users in different ways. However, these platforms still have some limitations, which may contain:

- *QnA:* Most platforms provide static content presentation without conversational interfaces, not supporting users a chatbot to ask specific questions or seeking clarification about complex topics. Even if they do, LLM may not return correct cites of its answer or may hallucinate while trying to generate response.
- *Language:* Due to no Chatbot to summarize and ask for information, the language from the platform may not available for who do not use that tongue.
- *Interface:* The User Web may display advertisements and unrelated banners, which interrupt user experience. As those sites have to benefit from affiliating, user may be require to pay for better usage.

Due to these limitations, there is a strong need for a system that can integrate automated real-time data collection, multilingual support, question answering, and summarization in one single platform. At the same time, this system must remain cost-efficient so that it can be widely accessible and practical for everyday use.

1.2 Challenges

While the problem is clearly defined, building such a system in practice is not straightforward. Several challenges arise during deployment and implementation.

1.2.1 Technology

Deploying a complete system is inherently difficult, as it requires identifying suitable open-source tools, managing AI models, and handling the associated costs.

Although many tools and platforms are available to assist in developing an end-to-end system, most still require a cost of deploying properties and a solid understanding of system integration.

Furthermore, keeping a domain running stably and a platform to store Data is a crucial statement when it comes to launch to public Internet.

1.2.2 Fetching

In term of crawling and processing data from news articles, several difficulties arise: overcoming anti-bot detection mechanisms, identifying key indexes from HTML parsers, and ensuring that the system can handle diverse website structures.

This challenge highlights the need for an AI-driven method to automatically detect and extract essential information (such as content, title, and author) rather than manually defining CSS selectors for each new website structure.

1.3 Proposed Solution

To address the challenges identified above, this study proposes the design and deployment of a low-code AI-powered system capable of automatically collecting, analyzing, and delivering information from multiple multilingual sources.

Instead of relying on manually defined CSS selectors or traditional parsers, the system leverages AI models to detect and extract essential components such as titles, authors, and content from diverse website structures.

By integrating workflow automation platforms like n8n, combined with open-source large language models (LLMs) deployed through Ollama, the system ensures both flexibility and cost-effectiveness.

A PostgreSQL database enhanced with pgvector is employed to store embeddings and enable efficient semantic retrieval, and real-time caching to optimize system performance.

Taking advantage of the easy deployed User Interface through Streamlit to visualize the working statement of the project immediately.

Through this architecture, the proposed solution minimizes the technical burden of building advanced chatbot systems, making it feasible for small teams, students, and non-expert developers to implement robust AI-driven data pipelines and conversational interfaces with minimal coding effort.

1.4 Objective

The primary objective of this research is to design and implement an AI-powered system that enables automated collection, processing, and retrieval of information from multiple multilingual sources. Specifically, the study aims to build a low-code platform that rovides users with a unified platform to freely interact, ask questions, and access the latest information on artificial intelligence.

Beyond delivering a functional solution, the project also serves as an opportunity for the author to strengthen essential skills, including automation workflow design, problem-solving and solution deployment, system debugging and maintenance, as well as the practical application of AI knowledge acquired throughout three years of academic study.

2 Related Work

The rapid growth of digital media has motivated a variety of studies and systems that address the challenge of multi-source information access and summarization. Early works focused on news aggregation platforms such as Google News and Feedly, which collect articles from diverse publishers and present them in a unified interface. While effective at providing breadth of coverage, these systems mainly present static content and do not support conversational or semantic querying.

Subsequent research has explored automatic text summarization and multilingual NLP. Models such as *BART* [Lewis et al., 2020], *PEGASUS* [Zhang et al., 2020], and *mT5* [Xue et al., 2021] have demonstrated strong performance on abstractive summarization across languages. In parallel, cross-lingual summarization studies [e.g., Cao et al., 2020]

have shown the potential to translate and summarize simultaneously, enabling broader accessibility. However, many of these approaches require large-scale cloud infrastructure and incur high computational cost, limiting their practicality for lightweight deployments.

Another significant line of work is retrieval-augmented generation (RAG). Introduced by Lewis et al. (2020), RAG combines dense retrieval with generative models to improve factuality and coverage in question answering. Later implementations, such as REALM [Guu et al., 2020] and open-source RAG pipelines, have extended this paradigm. While effective, most prior deployments assume cloud-hosted large language models (LLMs), which raises concerns of cost, latency, and data privacy.

In recent years, low-code/no-code AI integration has emerged as a practical solution to lower the barrier for building AI-driven systems. Platforms such as n8n, OutSystems, and Zapier allow non-expert developers to construct automation workflows. Studies on low-code AI [e.g., Mäkitalo et al., 2022] highlight their role in democratizing access to AI pipelines. Yet, integration of these platforms with local AI models remains under-explored.

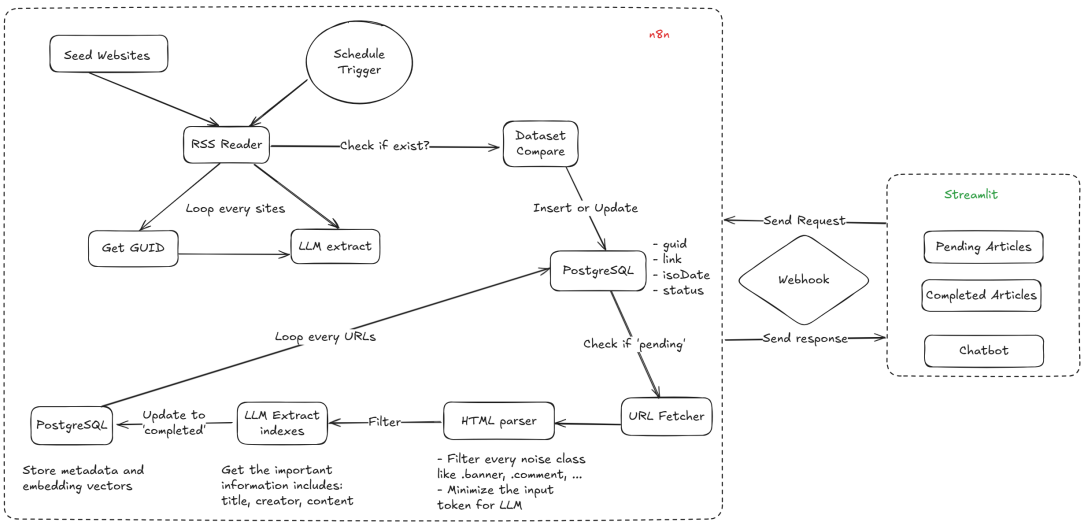
Research Gap

Although prior works have advanced multi-source aggregation, multilingual summarization, and RAG-based retrieval, few approaches simultaneously address real-time multi-source crawling, AI-powered extractor, conversational querying, and cost-efficient low-code deployment with local AI models. This motivates my study, which aims to design a system that unifies these features into a practical and affordable solution.

3 Proposed Method

System Architecture Overview

The system operates as an integrated pipeline with three main components: n8n for workflow automation, PostgreSQL database for storage, and Streamlit for interface. The architecture is designed to handle continuous data collection, intelligent content processing, and real-time user queries while maintaining cost-effectiveness through local AI model deployment. The core methodology emphasizes automation and reliability with fallbacks to prevent interruptions.



3.1 Processing Pipeline

RSS Feed Collection The system begins with a Schedule Trigger that activates at regular intervals throughout the day. It transmits a list of seed websites suchas VnExpress, InsideHPC,... to RSS Reader component to fetch every site to collect unique identifier.

GUID Generation For each collected article, the system generates a GUID using an LLM Extract component. The Dataset Compare component then checks in the Database too see if the article already exists in the PostgreSQL database by comparing GUIDs to decide whether to insert or update.

URL Fetching Articles marked as "pending" undergo detailed content extraction. The URL Fetcher component retrieves the full HTML. The HTML Parser then filters out noise elements while preserving the main article content.

Content Analysis The cleaned HTML content is processed by the LLM Extract component, which uses multiple AI models to identify and extract three key elements.

Storage Successfully processed articles are stored in PostgreSQL in two formats. The raw metadata and content are saved in relational tables for structured queries. The article status is updated to "completed" upon successful processing.

3.2 Interaction

Chatbot The system provides user access through a Streamlit-based chatbot interface that connects to the PostgreSQL database. Users can query the knowledge base using natural language.

Webhook A webhook system enables communication between the n8n workflow and the Streamlit interface.

Visualize Data The system supports view lists of pending articles, completed articles, and interact with the chatbot for specific information retrieval.

4 Implementation

The system operates through two main workflows: the Fetching GUID workflow, which stores unique GUID of every articles into PosgreSQL, and the Processor workflow, which get every upcoming URL from DB to parse HTML and extract information by LLM model. n8n manages these Crawler task and ensures smooth progress during every node. The system divides into 2 separate workflow to avoid monolithic procedure to reduce the latency between Crawling and Processing. Every URL is in pending status will have to wait in queue and will be processed respectively.

4.1 Technologies

In developing an optimized price-cost system, selecting the right tools and platforms is crucial to ensure smooth integration, reliable performance, and scalability. Many technologies are available to support different aspects of development, ranging from workflow automation to data storage, AI model hosting, and user interface design. This project carefully examined and selected a combination of tools that complement each other and meet the project's requirements for flexibility, ease of use, and open-source availability. The following sections describe the key technologies integrated into the system.

4.1.1 n8n

n8n is an open-source workflow automation tool that allows users to create complex workflows through a simple and intuitive interface without needing extensive programming skills. It provides a node-báe system where users can connect various services, APIs, and data processing steps to automate tasks efficiently. One of n8n's biggest advantages is its flexibility; it supports a wide range of integrations out of the box and also allows users to drag and drop every node task.

4.1.2 Ollama

Ollama is a platform that allows users to run LLMs locally on their own servers or machines. It provides a simple way to deploy and manage powerful AI models without relying on cloud services. With Ollama, users can perform various natural language processing tasks such as text generation, embedding, and question answering while maintaining full control over their data and system performance.

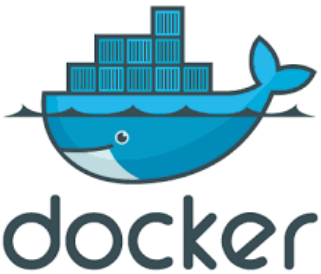
4.1.3 PostgreSQL

PostgreSQL is an open-source relational database management system widely recognized for its reliability, extensibility, and robust support for advanced data types. With its ability to handle structured and unstructured data, PostgreSQL has become a popular choice for building modern AI-driven applications. In this project, PostgreSQL is employed not only as a storage solution for raw and embedding vector database, enabling efficient semantic search and retrieval.

4.1.4 Streamlit

Streamlit is an open-source framework designed to simplify the development of interactive data applications. Within this project, Streamlit serves as the primary front-end platform, providing users with an intuitive and responsive interface to query the system, explore AI-related information, and interact with the underlying automation workflows in real time.

4.1.5 Docker



Docker is an open-source platform that enables the packaging and deployment of applications within lightweight, portable, and consistent containers, ensuring that the software can be executed reliably across diverse infrastructures. By abstracting the underlying system, Docker simplifies the processes of development, testing, and deployment, while also improving resource efficiency and system scalability.

4.2 Indexing GUID Module

This module workflow starts by reading data from RSS feeds and checking against the PostgreSQL database to see if the article's id already exists. If it's not exists, the system extracts a clean GUID from the URL and marks the record as "pending." into Postgre.

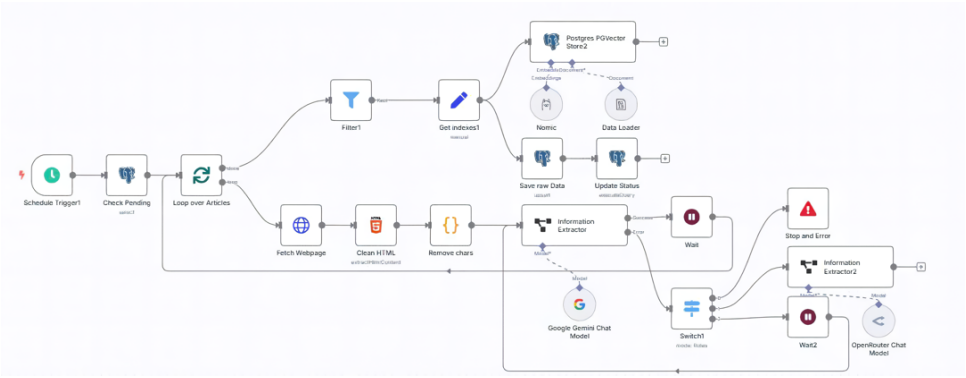
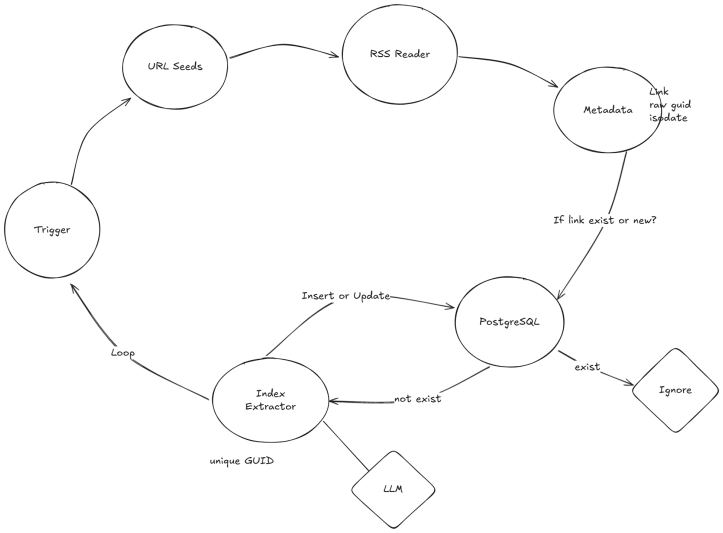


Figure 1: RSS Reader and GUID Indexer

To simplify, we will take a look at the pipeline below for better understanding



Schedule Trigger

The entire process is set into motion by a precise schedule trigger that activates the workflow four hours in a loop, creating a consistent and reliable rhythm for data collection without any need for manual intervention.

Seed URLs

Upon activation, the workflow first gets information from a code node, which contains a predefined array of RSS feed URLs from various prominent sources, which it then systematically feeds one by one into a looping mechanism designed to handle each source sequentially to maintain order and prevent overload.

RSS Feed Reader

Each individual feed URL is passed to an RSS reader node that performs the crucial task of connecting to the previous source, fetching the entire list of recent articles, and extracting all available metadata, including titles, descriptions, publication dates in ISO format, links, and each item's inherent GUID.

Filtering

Following this data retrieval, an immediate filtering operation is applied to scan through the newly articles, checking each item's ISO date against the current system date and allowing only those articles published on *today's format* of the workflow's execution to pass through, effectively acting a gatekeeper to ensure temporal relevance and filter out stale or old content. The articles that pass this date test are then queued up for individual processing, where each one is isolated and its core identifying data—namely the GUID, the link, and the ISO date—is carefully parsed to be carried out by the AI-powered node.

AI Integration

For each incoming article, the flow prepares the key details and then paths the article's unique ID to a Information Extracter node to clean it up. Often, these IDs are long, messy URLs, and inconsistent so the AI's job is to turn them into a neat and short standard format, like turning 'https://knowtechie.com/?p=12345' into a clean ID like 'knowtechie_12345'.

Beyond my evaluation of all available model, I choose gemini-2.0-flash-lite as the core model to carry out this task, the model works perfectly fit the aim of time consuming and properties limitation.

To avoid exceeding RPM of Gemini, the flow has to wait 30 seconds before continue processed the next article.

Insert or Update

To avoid duplication, every new article is compared with records already stored in the PostgreSQL. If the item is new, it is inserted into the database; if it is already exists but the isoDate is diffent, replaced with the old guid; otherwise, it is skipped. This ensures that all records follow an unique id.

After this step, the status corresponded with the id is marked as "pending." Later,the up-next module will handle preprocessing task. The process is designed to run automatically, with error handling and throttling mechanisms to ensure stable operation.

4.3 Extraction and Storage Module

After new articles guid are identified in the previous module, this workflow is triggered to extract and store the full article content. The process begins by fetching the raw webpage of each article link. The HTML is then cleaned to remove irrelevant sections. The entire content will be embedded to store in the PostgreSQL to further advanced usage of RAG Chatbot.

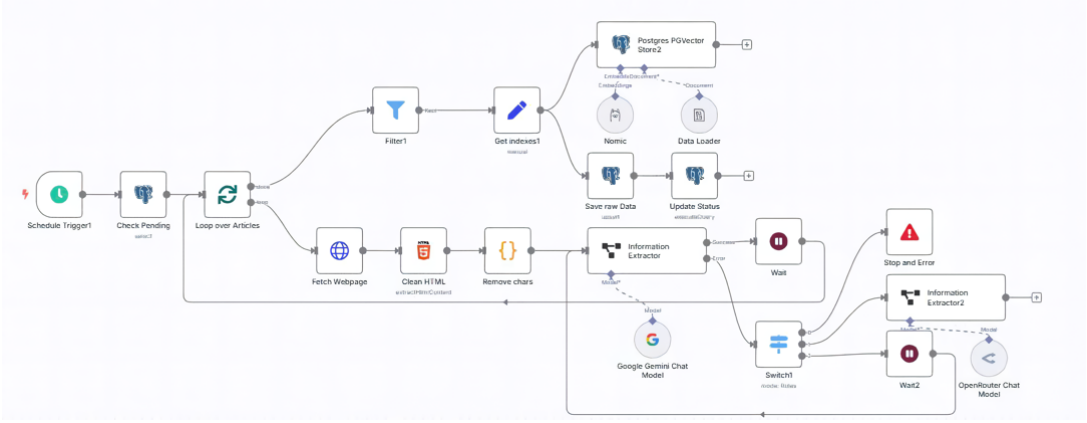


Figure 2: Extract Content and Store

Check Queue

This flow works like smart content processing factory that takes over after an article is marked as 'pending'. It starts with another scheduled trigger that wakes up the system to check the database for any articles that have the 'pending' status, meaning they are ready to be fully processed. Once it finds these pending articles, it loops through them one by one.

HTML Parsing

For each article, the first crucial step is to actually go out to the internet and fetch the full webpage content; it does this by sending an HTTP request to the article's link, using a fake "User-Agent" header to make it look like a regular web browser and avoid being blocked by the website. After grabbing the entire HTML of the page, it goes through a multi-stage cleaning process:

First, an HTML node tries to extract just the main content of the article by skipping common junk like headers, footers, sidebars, and advertisements using CSS selectors.

Then, a custom code node performs a deeper cleanup, using JavaScript to find and remove specific leftover patterns, navigation menus, social sharing buttons, copyright text, and any remaining URLs to leave behind only the purest possible text content.

Content Extractor

Those cleaned HTML is then sent to the powerful AI node, with gemini-2.5-flash engined, which acts as a precision information extractor. The AI is given very strict instructions to find and pull out three key pieces of information exactly as they are written on the page: the exact article title, the author's name, and the complete body content of the article, making sure to preserve every single paragraph, all original formatting, and without changing a single word, thus ensuring the content is captured perfectly.

HTTPS Fail Request

In a real concept, things do not always run smoothly. A request to an API might fail, the input data might be missing some fields, or a service could become temporarily unavailable. That is why error handling is an essential part of building workflows. Instead of letting the whole process stop, we can catch errors, handle them properly, and continue with the flow.

For the parsing task, if a web scraping task fails, the workflow can be configured to log the error, send a notification, and then move on to the next task without breaking the entire system. This makes the workflow more reliable and production-ready.

API Exception

This workflow is also designed to be highly reliable and has a smart backup plan in case the main AI service runs into problems. This is handled by a crucial component called a Switch node. This node acts like a intelligent router or a traffic controller, constantly checking the output from the primary Google Gemini step for any error messages.

Case 1 If the selected model return "could not be found" (a 404 error), the workflow immediately returns an error message and halts execution, allowing the developer to adjust the configuration before continuing.

Case 2 If the model fails to process the content or exceeds its quota (typically returning a 429 error), it automatically redirects the task, along with all the article's data, down a different pathway to a backup AI service called OpenRouter so that the system can continue without interruption.

Case 3 In the case of exceeding the request-per-minute (RPM) limit (403 error), the system is designed to wait for one minute before retrying the same node, thereby preventing unnecessary failures.

Together, error handling and switch nodes make the workflow smarter and more resilient. Errors can be isolated and handled gracefully, while conditional logic can dynamically adjust the process depending on the data. This combination is especially important in large-scale automation systems where different data sources and unpredictable failures are common.

Storage and Embedding

Once the title, author, and content are successfully extracted, they are packaged together with the article's GUID. This valuable data is then saved into two places:

Raw Data The raw data is stored in a structured format in the main PostgreSQL database for record-keeping.

Embedding the content is also converted into a numerical representation called an embedding using a local Ollama model named Nomic, which is then saved into a special PGVector database designed for similarity searches, allowing us to later find articles based on their meaning and not just keywords.

Finally, to mark this article's journey as complete, the workflow runs an SQL query to update its status in the database from 'pending' to 'completed'. And just like before, it includes a waiting period between processing articles to be polite to the websites and AI services, ensuring a smooth and sustainable automated operation.

4.4 Question-Answering Logic

The core of the system is implemented as a retrieval-augmented generation (RAG) workflow in `n8n`, which integrates multiple components to handle question answering. The overall flow can be described as follow:

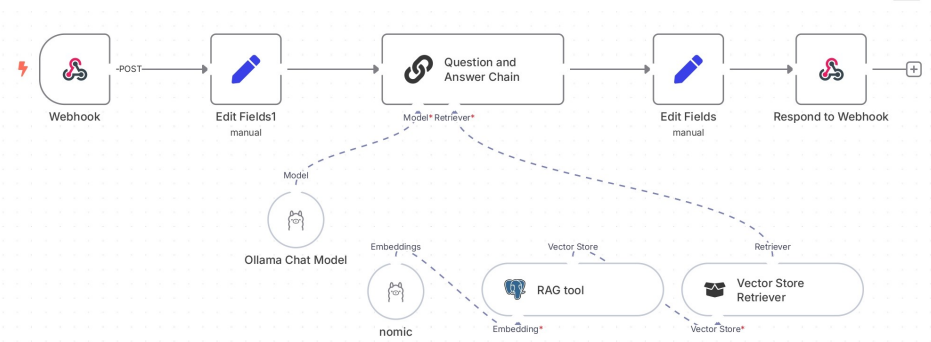


Figure 3: RAG Processing

Webhook Input: A user query from User Interface is sent via a webhook, which triggers the workflow execution.

Preprocessing: The query is extracted and normalized using a `Set` node to prepare it for downstream processing.

Retrieval-Augmented Chain: The query is passed to a `Retrieval QA Chain`, which combines three essential components:

Embedding model (`nomic-embed-text`) that transforms both the query and stored documents into vector representations.

Vector database (PostgreSQL with `pgvector`) that stores embeddings and performs similarity search to retrieve relevant context.

Chat model (`qwen2:7B` via Ollama) that generates an answer conditioned on the retrieved context.

Context-Aware Answering: The language model is instructed with a carefully designed system prompt, enforcing strict rules:

Use only the retrieved context to generate responses.

Provide partial answers if the context is incomplete.

Indicate explicitly when no relevant information is available.

Prioritize accuracy, specificity, and structured output.

Response Handling: The generated response is processed through a `Set` node and then returned to the user via the *Respond to Webhook* node.

In short

This modular design ensures that the system can answer questions reliably based on stored knowledge. By separating the embedding, retrieval, and generation steps, the workflow achieves both transparency and flexibility.

Overall, this workflow demonstrates the effectiveness of combining *vector search (retrieval)* with *generative reasoning (LLM)* to deliver accurate, context-aware answers in real time.

4.5 User Interface

The user interface of the system was built using Streamlit, which provides a simple yet powerful way to create interactive deployment in Python. The interface is divided into two main sections: Article Management Dashboard and Chatbot Interface

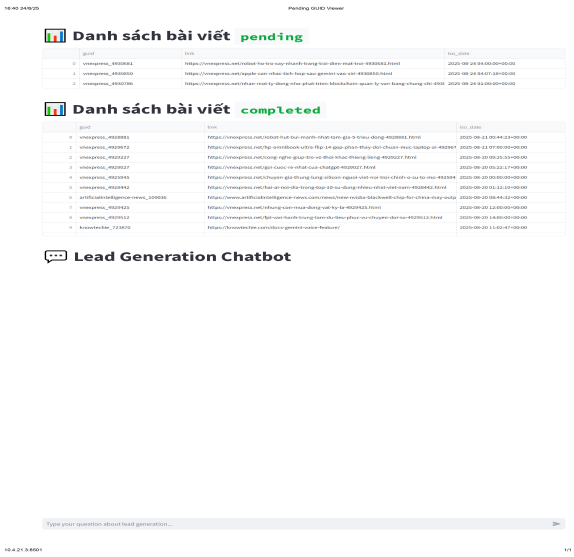


Figure 4: Dashboard

Data Visualize

The pending table is dedicated to monitoring the status of articles in the PostgreSQL database. It establishes a secure connection to the database using credentials stored in environment variables (like the username and password). It then runs two separate SQL queries: the first query fetches all articles that have a status of 'pending', showing their unique identifier (GUID), web link, and publication date, and displays them in a clean, interactive table on the page.

The completed table Right below it, a second query does the same thing but for all articles with a status of 'completed', providing a clear at-a-glance view of which articles are waiting to be processed and which have already been finished. This allows users to easily track the progress of the automated workflow.

User Interaction

The second section is an interactive AI chatbot designed for lead generation conversations. It features a familiar chat interface where the conversation history is persistently stored and displayed in the app, so the dialogue doesn't disappear when the user interacts with other parts of the page. When a user types a question into the chat input box, the application first saves and displays the user's message.

Crucially, it then takes that question and sends it as a JSON payload to a specified webhook URL that triggers n8n workflow. This n8n workflow presumably processes the query using the previously collected article data and an AI model. The dashboard then waits for a response from the n8n server. If the request is successful (status code 200), it parses the JSON response and displays the AI's reply in the chat history. Otherwise, the server returns an error code (like 404 or 500), or the response isn't in the expected JSON format, ensuring the user gets a clear error message instead of a crashed application.

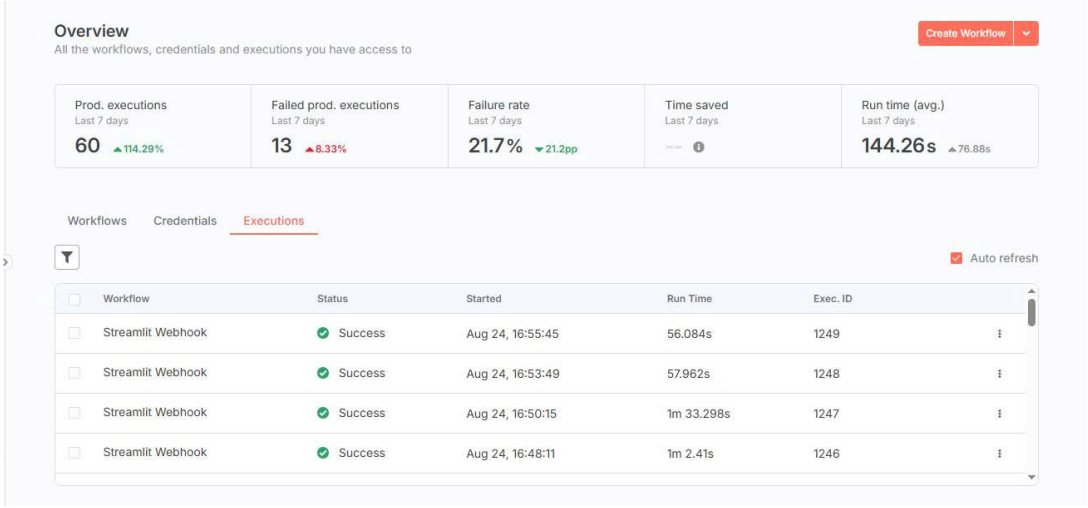
In essence, this dashboard acts as both a control panel for observing the backend data pipeline and a conversational interface for end-users to leverage the insights gathered by that pipeline.

5 Evaluation

This section evaluates the performance of the automated content processing system based on three key aspects: how reliable it is, how well the AI models work, and the effectiveness of the content grabbing method. To ensure that the system works smoothly and meets the project objectives, the following aspects should be evaluated

5.1 Workflow Performance

The overall system is highly reliable because it is built like a strong pipeline with backup plans. The use of a Error handling Logic to catch errors is a key strength. To validate the efficiency of the proposed workflow, performance metrics were collected from the n8n execution dashboard over the last 7 days. The results are as follows:



Production executions: 60 successful runs, representing a 114.29% increase compared to the previous period.

Failed executions: 13 failures, accounting for a 21.7% overall failure rate. However, this indicates a 21.2 percentage point improvement, suggesting that the recent error-handling mechanisms have significantly stabilized the system.

Average runtime: 144.26 seconds per workflow execution, which is 76.88 seconds faster than the previous measurement period. This shows that recent optimizations in HTML parsing, content extraction, and database operations have improved efficiency.

5.2 Model Capability

Since multiple LLMs can be integrated into the workflow, it is important to evaluate their suitability for the task. To assess the effectiveness of different language models in our information extraction pipeline, we conducted experiments using six candidate models across three news articles. Each model was evaluated on two key dimensions:

Precision: The percentage of correctly extracted fields compared to the ground truth.

Response Time: The average latency (in milliseconds) for the model to return results.

Link to my LLM Evaluation

Results

Table 1: Overall performance of LLMs on field extraction

Model	**Precision (%)**	**Avg. Response Time (ms)**
openai/gpt-oss-20b	∼75	12,387
Llama-3.3-70B	∼85	7,947
Gemini-2.0-flash-lite	∼87	5,914
Gemini-2.0-flash	∼88	5,790
Gemini-2.5-flash-preview	∼90	6,941

Observations

- *openai/gpt-oss-20b* achieves a precision of around 75%. However, its response time is the slowest (over 12 seconds on average), making it impractical for large-scale workflows despite the benefit of unlimited requests.
- *Llama-3.3-70B* improves accuracy to about 85% with significantly lower latency (∼8 seconds). It shows stable extraction across all fields but still lags behind Gemini models.
- *Gemini-2.0-flash-lite* and **Gemini-2.0-flash** both achieve strong precision (87–88%) with fast response times (5–6 seconds). They provide a reliable balance of accuracy and efficiency.
- *Gemini-2.5-flash-preview* reaches the highest accuracy (around 90%) with a moderate response time (∼7 seconds). While it is slightly slower than Gemini-2.0-flash, it consistently extracts more complete content and is therefore the best-performing option.

Conclusion

Based on these experiments, **Gemini-2.5-flash-preview** was selected as the final model for integration into the workflow. It provides the best trade-off between precision and response time, ensuring reliable extraction of *title*, *creator*, and *content* fields while maintaining acceptable speed.

The Request per Minute (RPM) might be a little low limited, still, the system has set timewait for the AI Agent to avoid exceeding request permit.

5.3 AI Crawling Method

This system uses a modern AI method instead of traditional CSS selectors for grabbing content, and each method has pros and cons.

*Traditional CSS Selector:*The old way is to write code that looks for specific HTML tags (like ¡div class=”article-content”¿) to find the content. The problem is that every website has a different structure. If a website changes its design, the code breaks and must be fixed by a developer. This method is rigid and requires a lot of maintenance.

AI-Powered: This system uses an AI model to read the cleaned-up HTML and understand what the title, author, and content are. The AI is given instructions like “find the title” and ”extract the origin article content” The huge advantage of this method is its flexibility. Since the AI understands the context, it can work on many different websites without needing someone to write new code for each one. It is much more adaptable to new websites and changes on existing sites. However, sometimes AI cannot give us the exact content that we require, especially when we trying to optimize the cost and properties, some LLM with less than 20B parameters can not work well, they fluctuate around 80% to 90% of the precision.

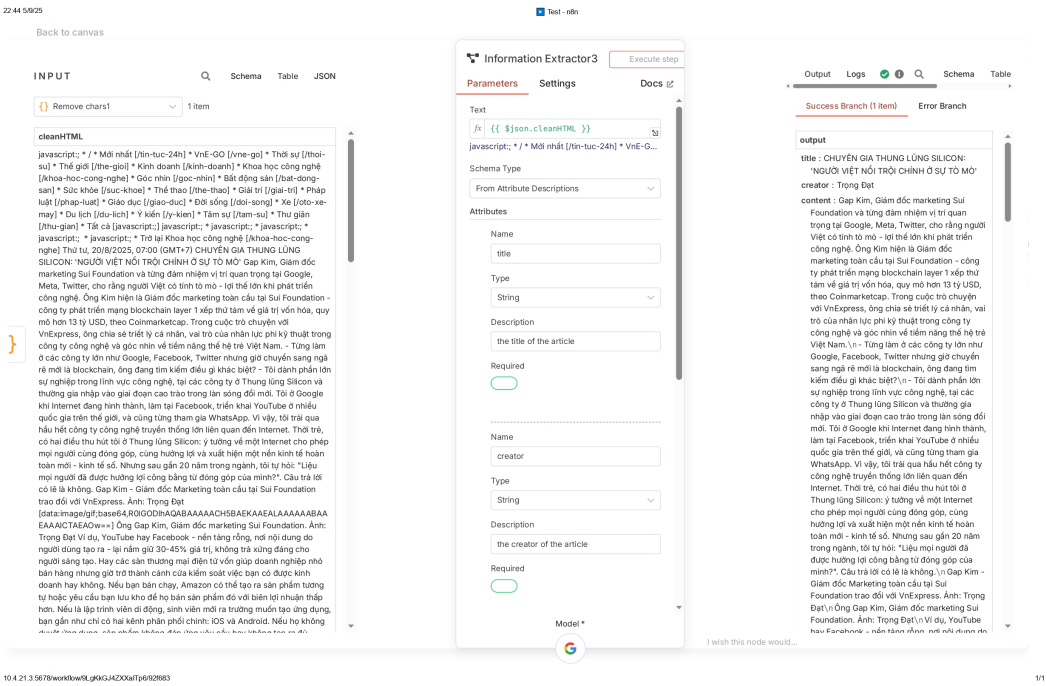
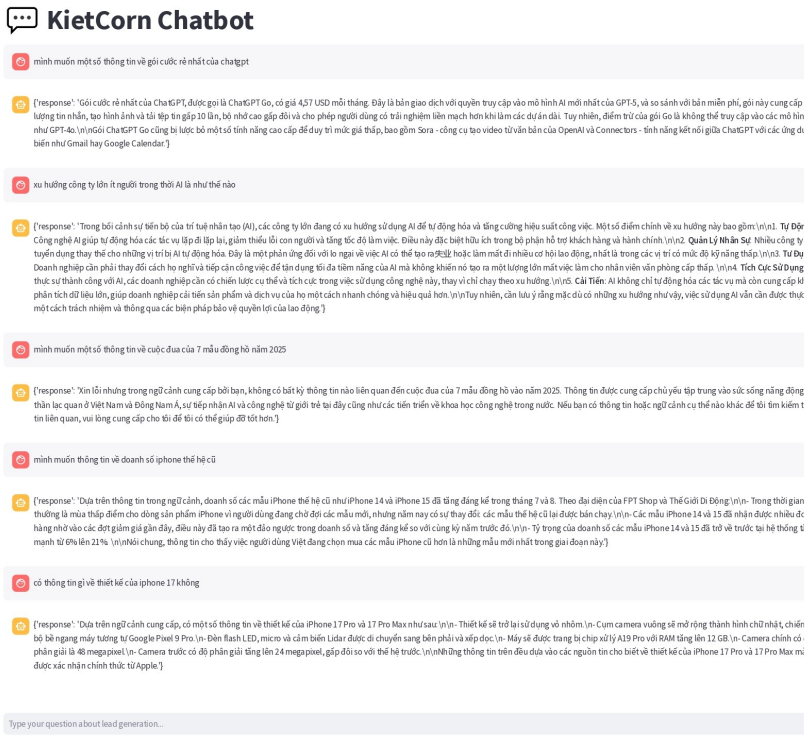


Figure 5: AI in Crawling

As you can see, the AI integration is more advanced and sustainable for a project that collects data from many different sources. However, it may resume risk that the output is not given as exactly as what it is prompted to do.

6 Result and Conclusion

The system was successfully able to perform question-answering based on the information stored in the database. The model was capable of generating coherent responses and managed to provide correct answers for approximately 85% of the input information. This demonstrates that the retrieval-augmented generation (RAG) pipeline worked as intended, with the model effectively leveraging stored knowledge to deliver relevant outputs.



Despite the limitations in resources and computational cost, the experiment was conducted using the `qwen2:7B` model. While this is not the largest available model, it showed notable efficiency in handling retrieval and generation tasks within the constraints of the system. The choice of a lightweight yet capable model also highlights the trade-off between accuracy and resource consumption, which is crucial for practical deployment in environments with limited infrastructure.

Besides, as you can see, when I ask a tricky question about the race of the seven watches in 2025, the LLM does not hallucinate the information and frankly response to the user that it does not contains that knowledge, which proves that the model is working relatively well.

Overall, the results suggest that even with a mid-sized model like `qwen2:7B`, the system can achieve a relatively high accuracy rate in answering domain-specific questions. This indicates the robustness of the RAG approach in enhancing response quality compared to using a standalone LLM. Nevertheless, there is still room for improvement in coverage and accuracy, especially for edge cases and ambiguous queries. Future work could explore the integration of larger models or fine-tuned domain-specific embeddings to further improve performance without significantly increasing resource cost.

7 Future Works

While the current system successfully extracts website content and enables a chatbot to provide relevant responses, it remains only a Minimum Viable Product (MVP) for a complete Summarization System. The current design lacks scalability for multi-user scenarios and does not fully reflect the flexibility of a robust research system.
During the development of this project, several promising directions for further research and improvement have been identified.
One key area is fine-tuning small language models. Instead of relying heavily on large language models (LLMs) for small tasks within the pipeline—such as summarizing, chunking, or metadata extraction—fine-tuned smaller models could perform these tasks more efficiently, enabling faster processing, lower costs, and greater scalability when dealing with a large number of upcoming data.
Another potential direction is the creation of autonomous agents. From the extracted data stored in the database, a dedicated news aggregation agent could be designed to collect and summarize up to hundreds of articles daily. For example, it could filter AI-related news and synthesize them into a single, coherent report centered around specific keywords.
Furthermore, based on user interaction history, a recommendation agent could be developed to learn user preferences and suggest relevant articles, potentially integrated into a personalized user interface.
Finally, an important direction for future work is to explore multimodal capabilities. While the current system focuses exclusively on text, expanding to multimodal inputs such as images, audio, or pdf could significantly enhance its usefulness. For instance, the system could analyze images or charts embedded in articles to provide richer summaries, allow users to interact with the chatbot through voice queries, or process multimedia documents like PDFs combining text and visuals. This expansion would bring the system closer to real-world applications where information rarely exists in a single modality.

8 References

References

[1] n8n Documentation. Available at: `https://docs.n8n.io`

[2] Ollama Documentation. Available at: `https://ollama.com/library`

[3] PostgreSQL Documentation. *Available at*: `https://www.postgresql.org/docs/`

[4] Streamlit Documentation. Available at: `https://docs.streamlit.io/`

[5] Docker Documentation. Available at: `https://docs.docker.com/`

[6] Low-Code Application Platforms. `https://www.outsystems.com/low-code/`

Evaluation Comments

Comments: .....

.....

.....

.....

.....

.....

.....

Score (Number): ..... Score (Word): .....

Hanoi, month _____ day_____ year _____

Advisor

(Signature and full name)