TẬP ĐOÀN CÔNG NGHIỆP - VIỄN THÔNG QUÂN ĐỘI

_____*_____

BÁO CÁO MINI PROJECT

Chương trình: Viettel Digital Talent 2025
Lĩnh vực: Software Engineering

ĐỀ TÀI: CÔNG NGHỆ THU THẬP DỮ LIỆU ỨNG DỤNG VÀO HỆ THỐNG THU THẬP DỮ LIỆU BÁO CHÍ

Người thực hiện:	Phạm Hữu Huy
Mentor:	Vũ Việt Hoàng
Đơn vị mentor:	Viettel AI

Hà Nội – 06/2025

Mục lục

I. GIỚI THIỆU CHUNG

1. Mô tả đề tài

Đề tài "Tìm hiểu công nghệ thu thập dữ liệu ứng dụng vào hệ thống thu thập dữ liệu báo chí" hướng đến nghiên cứu và xây dựng một hệ thống có khả năng tự động thu thập, xử lý và lưu trữ dữ liệu từ nhiều trang báo điện tử tiếng Việt. Có ba hướng tìm hiểu chính để ứng dụng vào hệ thống là công nghệ xử lý đa luồng, công nghệ lưu trữ, xử lý phân tán và tích hợp AI để phân tích cấu trúc HTML từ đó trích xuất thông tin.

2. Phạm vi nghiên cứu

Phạm vi nghiên cứu của đề tài bao gồm việc thiết kế và triển khai các module chính của hệ thống thu thập dữ liệu: từ quản lý danh sách url, thực thi việc tải nội dung, phân tích HTML, đến việc lưu trữ và tìm kiếm dữ liệu đã xử lý. Bên cạnh đó, đề tài cũng tập trung vào các yếu tố về hiệu năng, khả năng mở rộng, độ chính xác của trích xuất thông tin, cũng như việc tích hợp các công nghệ AI hiện đại như mô hình ngôn ngữ lớn (LLM) trong quá trình phân tích dữ liệu. Hệ thống sẽ tập trung thu thập dữ liệu từ 30 trang báo của Việt Nam, dữ liệu thu thập sẽ bao gồm các trường thông tin cơ bản của một bài báo: tiêu đề bài báo, nội dung chi tiết, thời gian xuất bản và thông tin tác giả.

3. Lý thuyết và công nghệ

Một mô hình thu thập dữ liệu không đơn giản chỉ là việc tải xuống nội dung HTML từ các trang web mà còn gồm việc phân tích cấu trúc trang, xử lý các thành phần quan trọng, xử lý loang các đường dẫn có trong trang, xử lý khi gặp lỗi kết nối, … Hệ thống này sẽ thường bao gồm các thành phần cơ bản như bộ điều phối truy url, bộ thu thập, phân tích nội dung, và nơi lưu trữ dữ liệu. Các thành phần này phối hợp với nhau đảm bảo luồng xử lý liên tục và hiệu quả.

Xử lý đa luồng (multi-threading) là kỹ thuật quan trọng giúp tăng tốc độ thu thập dữ liệu bằng cách cho phép nhiều tiến trình tải và xử lý trang web hoạt động song song. Bản chất của thu thập dữ liệu từ web là quá trình I/O intensive, và nó sẽ có thời gian đợi phản hồi từ server. Việc thiết kế thread pool hiệu quả cho phép tận dụng được thời gian chờ để thực hiện tác vụ khác, từ đó tăng được throughput tổng thể của hệ thống. Tuy nhiên việc triển khai đa luồng cũng cần chú ý đến quản lý, phân phối tài nguyên, cần chú ý đến tính đồng bộ của dữ liệu giữa các luồng tránh tình trạng race condition.

Công nghệ lưu trữ và xử lý phân tán là yếu tố quan trọng trong các hệ thống thu thập dữ liệu quy mô lớn. Thay vì thực hiện toàn bộ pipeline tuần tự trên một node, hệ thống phân tán sẽ chia nhỏ các công việc, phân bố cho các node, xử lý song song, độc lập trên các máy chủ khác nhau. Điều này giúp tăng khả năng mở rộng cũng như tăng tính sẵn sàng cho hệ thống. Với hệ thống thu thập dữ liệu thì mỗi thành phần chính như bộ điều phối, phân tích nội dung, lưu trữ đều có thể triển khai theo dạng các cụm với nhiều node khác nhau.

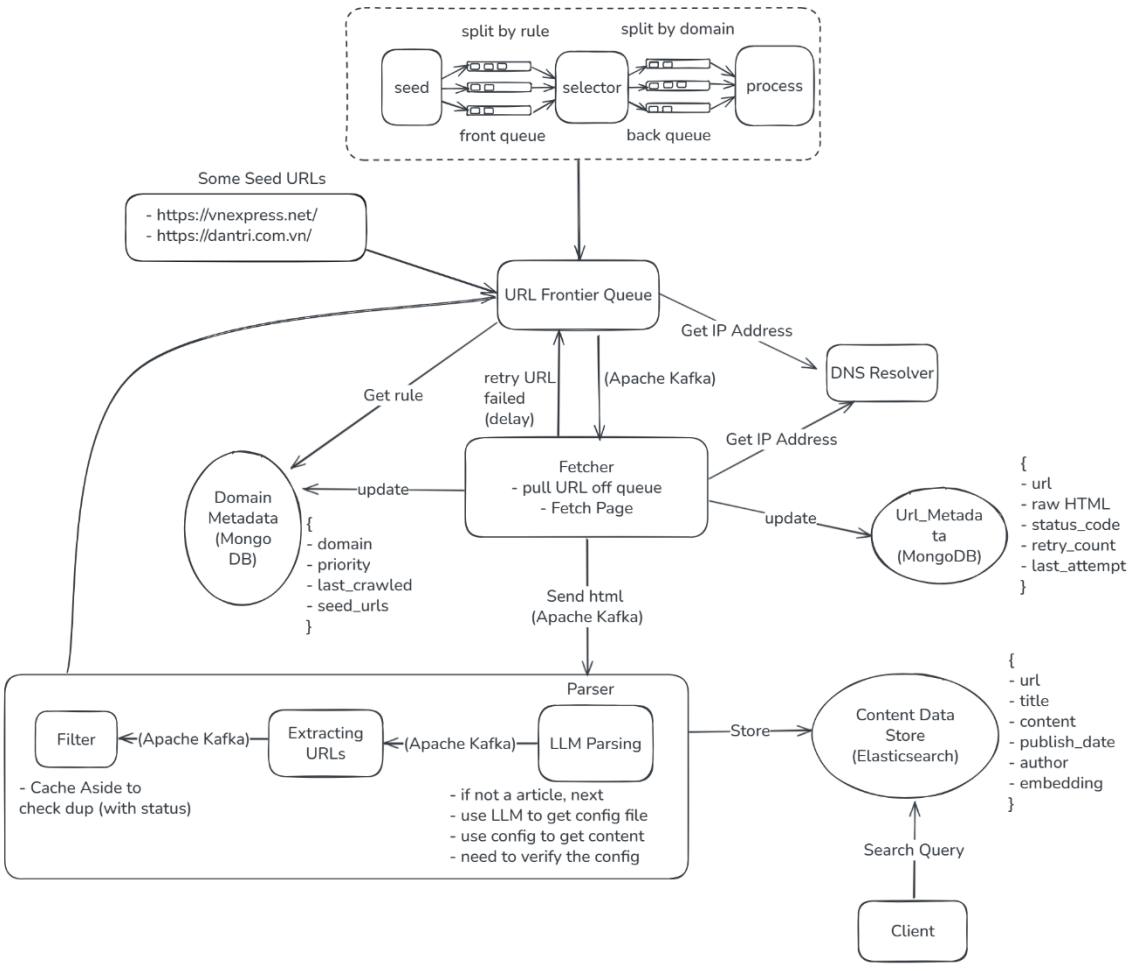
Một trong những thách thức lớn khi thu thập dữ liệu từ nhiều trang web khác nhau là sự không đồng nhất trong cấu trúc HTML. Việc tích hợp mô hình ngôn ngữ lớn (LLM) giúp tự động phát hiện các thành phần thông tin quan trọng như tiêu đề, nội dung, tác giả hoặc thời gian đăng bài, thay vì viết các parser thủ công cho từng trang hay áp dụng các rule-based cứng nhắc cho các trang báo. Điều này giúp tiết kiệm công sức phát triển và nâng cao khả năng thích nghi của hệ thống khi các trang thay đổi giao diện.

II. THIẾT KẾ VÀ TRIỂN KHAI HỆ THỐNG

Phần tiếp theo sẽ trình bày chi tiết về cách thiết kế và triển khai hệ thống thu thập dữ liệu báo chí, bao gồm kiến trúc tổng thể và vai trò của từng module chức năng. Đồng thời cũng chỉ ra những thách thức và giải pháp đưa ra khi triển khai thực tế.

1. Thiết kế hệ thống

Hệ thống thu thập dữ liệu (web crawler) trong lĩnh vực báo chí báo chí được xây dựng theo kiến trúc microservices, cho phép các thành phần hoạt động độc lập với nhau và có thể dễ dàng mở rộng riêng biệt theo nhu cầu của từng module. Kiến trúc này đảm bảo được tính linh hoạt trong phát triển và triển khai đồng thời cũng tăng khả năng chịu lỗi của toàn hệ thống.



Hình 1: Sơ đồ thiết kế hệ thống

Theo thiết kế trên luồng dữ liệu chính sẽ như sau, một url sẽ được điều phối thông qua Frontier, bộ điều phối này sẽ sắp xếp và gửi url đó đến Fetcher theo luật đã đặt sẵn. Fetcher sẽ thực hiện gửi HTTP request để lấy nội dung HTML của trang, sau đó phần nội dung HTML sẽ được gửi đến Parser để thực hiện trích xuất nội dung và các url khác có trong trang, sau cùng nội dung cần trích xuất sẽ được lưu và các url mới sẽ được thêm lại vào Frontier để thực hiện tiếp. Mỗi giai đoạn xử lý trên sẽ có chức năng riêng biệt nhưng được kết nối chặt chẽ thông qua các message hay API.

Event Streaming là quá trình thu thập, xử lý và truyền dữ liệu theo thời gian thực theo dạng luồng sự kiện có thể từ nhiều nguồn khác nhau. Dữ liệu này sẽ được lưu trữ lâu dài, bền vững, được xử lý và phản hồi theo thời gian thực, nó đảm bảo dữ liệu được truyền đi liên tục và đúng nơi đúng thời điểm.

Chính vì vậy Apache Kafka là công nghệ được sử dụng để điều phối các message giữa các module. Kafka là một công cụ hỗ trợ rất tốt cho các hệ thống phân tán, giúp các thành phần trong hệ thống giao tiếp và phối hợp với nhau một cách hiệu quả và bền vững. Với kiến trúc dựa trên mô hình publish-subcribe, kafka sẽ cho phép các producer gửi dữ liệu tới các topic, trong khi các consumer có thể đăng ký nhận và xử lý dữ liệu theo nhóm, điều này giúp phân công công việc đồng đều và tăng xử lý song song. Ngoài ra, Kafka có khả năng xử lý lượng dữ liệu lớn, độ trễ thấp, dữ liệu được lưu trữ bền vững, có khả năng retry, rất phù hợp đối với hệ thống này.

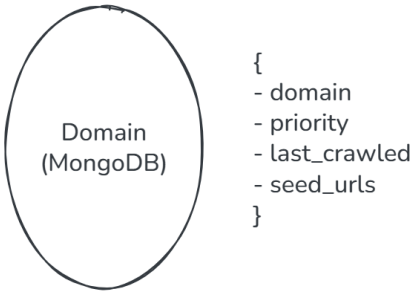
2. Frontier Module

Frontier đóng vai trò như một bộ điều phối của hệ thống, module này có nhiệm vụ quản lý danh sách url cần thu thập. Trước tiên ta cần phải biết Seed URLs là gì, đây chính là điểm bắt đầu của các domain khi một web crawler bắt đầu thực hiện. Ví dụ như ta có thể sử dụng chính tên domain cho Seed URLs, hoặc là trang chủ, các trang categories quan trọng, … Việc chọn được điểm bắt đầu một cách chính xác hay ít nhất là đủ tốt thì sẽ giúp hệ thống tìm nhanh được các thông tin mới, đặc biệt hữu ích trong lĩnh vực báo chí. Cụ thể chiến lược trong dự án này ta có thể thêm thủ công hoặc có thể tự động cho hệ thống nhận diện các trang categories để đưa làm Seed URLs. Cụ thể cách thức triển khai sẽ nhắc đến ở phần sau.

Sau khi có được các điểm bắt đầu thì ta cần xem xét đến việc ta sẽ thực hiện quản lý và điều phối. Có rất nhiều cách như dùng danh sách (List), dùng tập hợp (Set), dùng queue, tuy nhiên phải thiết kế sao để đảm bảo được ta sẽ thực hiện lấy thông tin theo mức độ ưu tiên của chúng ta muốn. Ví dụ như sẽ cần lấy thông tin của những domain có nội dung quan trọng hơn trước, hay muốn thực hiện lấy thông tin của domain có thời gian phản hồi nhanh, hoặc quan trọng hơn hết ta muốn phải đảm bảo được tính lịch sự (Politeness), tránh gửi quá nhiều request đến cùng một hosting server trong thời gian ngắn. Việc không đảm bảo politeness có thể khiến server nghĩ rằng chúng ta đang thực hiện tấn công DOS và sẽ chặn request từ phía ta. Như vậy có thể thấy việc thiết kế đơn

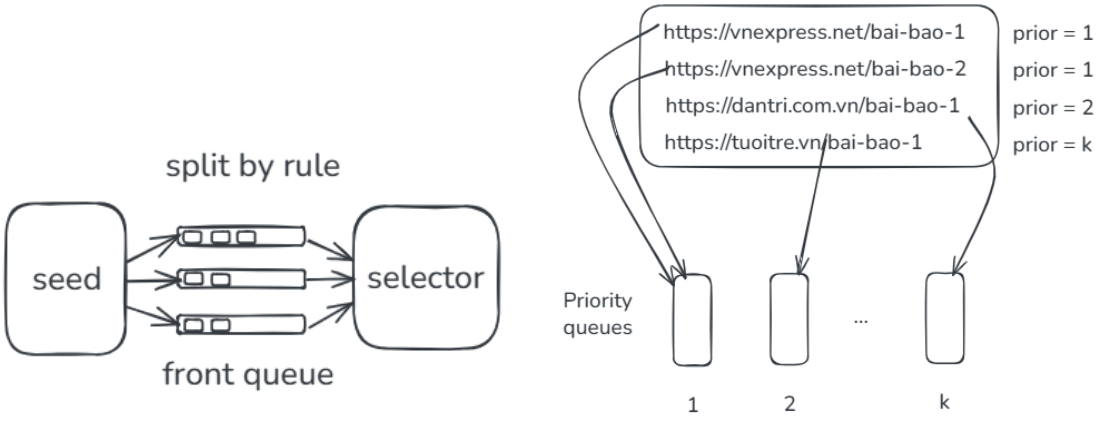
giản Frontier chỉ bằng danh sách, tập hợp, hay chỉ với queue đơn giản sẽ tồn tại rất nhiều hạn chế.

Ta sẽ có giải pháp riêng cho từng vấn đề. Trước tiên là độ ưu tiên, mỗi trang web sẽ có mức độ quan trọng của tin tức khác nhau ví dụ như trang báo vnexpress, laodong sẽ có mức độ thông tin chính xác và tốt hơn so với các trang báo kenh14, zingnews, … Ta sẽ cần lưu lại mức độ ưu tiên mà ta muốn cho từng domain. Ngoài ra ta sẽ xét thêm độ ưu tiên về lần crawl gần nhất, những url nào thuộc domain mới được crawl sẽ cần nhường chỗ cho các url thuộc domain khác thực hiện trước.



Hình 2: Lưu trữ thông tin về domain

Phần đầu tiên của Frontier sẽ là một bảng hash với key là độ ưu tiên đã tính toán, value sẽ là một priority queue. Frontier sẽ phân chia các url vào queue theo độ ưu tiên, và các url thuộc domain có lần crawl gần hơn sẽ ở sau. Cứ như vậy ta sẽ được một bảng hash có dạng như sau:



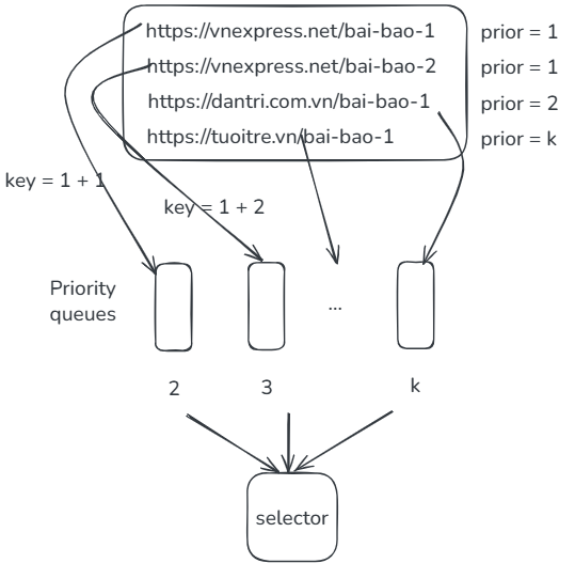
Hình 3: Giải pháp Frontier chia theo độ ưu tiên

Sau khi đã phân thành các queue như trên thì sẽ có Selector sẽ thực hiện lấy ra một loạt các url từ các queue có độ ưu tiên lớn hơn trước. Selector này sẽ thực hiện theo lịch, ví dụ như mỗi 2-3 giây lại lấy một lần, mỗi lần 10-20 url, cách cấu hình tùy thuộc vào cấu hình. Tuy nhiên sẽ có trường hợp nếu như có quá nhiều domain cùng độ ưu tiên hay số lượng url mới thuộc một vài domain có độ ưu tiên bằng nhau quá nhiều, ngược lại các url khác lại có số lượng ít hơn, như vậy cũng có thể dẫn đến việc chênh lệch quá lớn giữa số lượng thành viên giữa các queue, các url thuộc domain khác nhau nhưng

cùng độ ưu tiên sẽ phải chờ nhau rất lâu. Chính vì vậy, sẽ có một giải pháp tối ưu hơn, thay vì lưu trực tiếp các url có độ ưu tiên k vào queue k thì ta sẽ thực hiện lưu luân phiên, ví dụ ta có 3 url có độ ưu tiên k, ta sẽ thực hiện hash url và mod 3, khi đó key sẽ được tính là

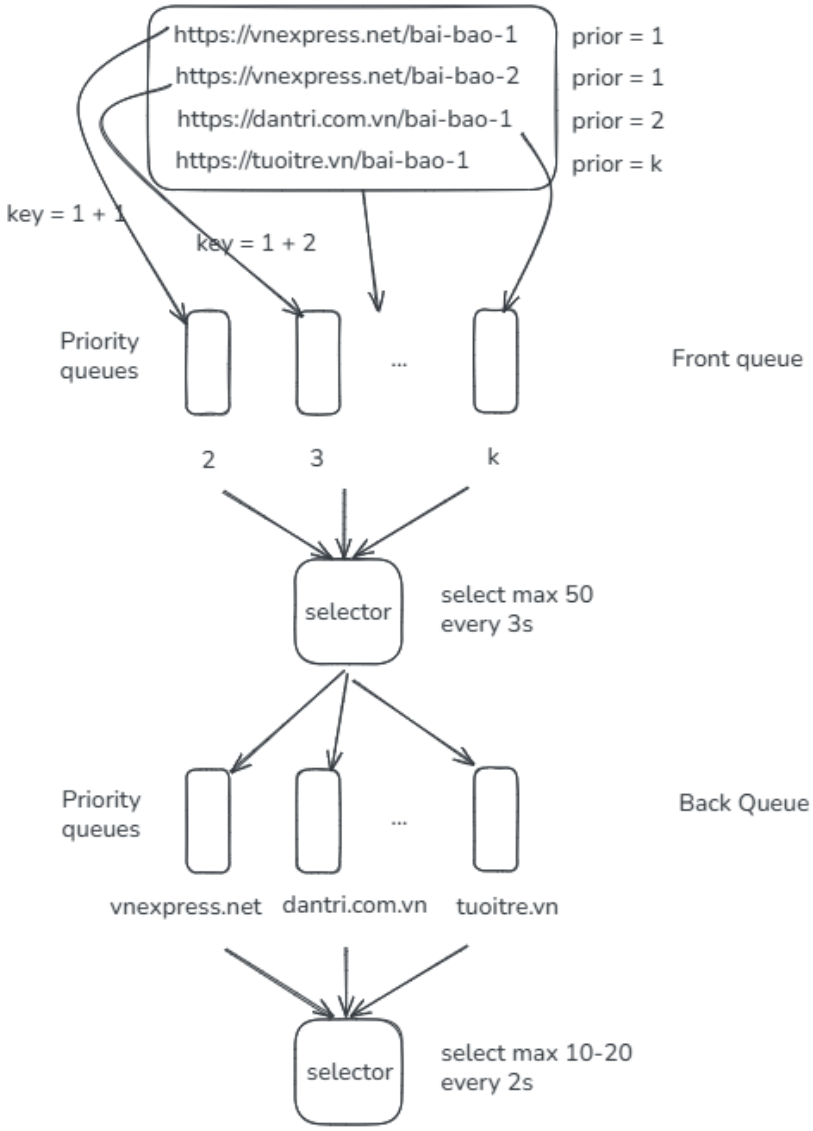
$$key = priority + (hashed\ url\ mod\ 3)$$

Khi đó các url sẽ được lưu vào queue có key được tính theo công thức trên, nếu như queue đầy sẽ thực hiện fallback xuống key thấp hơn, nếu quá 3 lần sẽ hủy bỏ. Và phần selector sẽ không luôn lấy các phần có key lớn nữa, ta sẽ sử dụng thuật toán Round Robin với trọng số của các url có độ ưu tiên cao, ví dụ như ta sẽ có 3 mức, mức cao nhất sẽ lấy 5 lần liên tiếp, mức trung bình lấy 3 lần liên tiếp, mức thấp lấy 1 lần. Như vậy sẽ đảm bảo bảo được các url sẽ chờ đợi nhau hợp lý hơn, tránh được tình trạng mất cân bằng giữa các queue dẫn đến tràn bộ nhớ cấp phát.



Hình 4: Tối ưu thuật toán cho Priority Queue

Tiếp đến là giải pháp cho tính lịch sự (Politeness), ở phần này các url thuộc cùng domain sau khi được lấy ra từ phần trên (ta gọi là front queue) sẽ được đưa vào một bảng hash với key chính là dựa vào domain. Và cũng sẽ có một Selector phụ trách việc lấy ra từ các queue (ta gọi là back queue) và gửi đến module sau. Phần Selector này sẽ thực hiện theo thuật toán Round Robin thông thường. Ngoài ra các url khi được thêm vào queue cũng đã được xem xét đến chỉ số crawl_delay trong robots.txt của domain đó, và thời gian mỗi lần Selector lấy cũng đủ để đảm bảo được tính Politeness.



Hình 5: Cách hoạt động của Frontier module

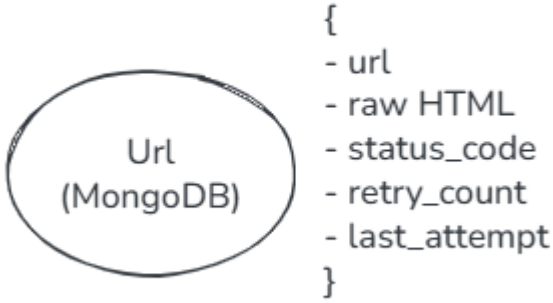
Tóm lại Frontier Module sẽ hoạt động như sau: Các url mới sẽ được kiểm tra xem đã có url này ở trong các queue chưa, có thuộc các domain cần crawl hay không (nếu là sub domain thì cũng vẫn bỏ qua), sau đó sẽ kiểm tra xem có được phép truy cập không, và lấy thời gian crawl_delay từ robots.txt. Nếu được phép thì lấy thông tin về độ ưu tiên trong database rồi đưa vào front queue, sau đó sẽ theo lịch trình mà đi qua back queue rồi được gửi thông qua Kafka với topic “fetching_tasks” đến Fetcher Module. Ngoài ra để đảm bảo được sẽ thu thập các bài báo mới nhất muộn nhất là 30 phút thì sẽ có một cronjob cố định mỗi 20 phút sẽ thực hiện thêm Seed URLs vào Frontier để crawl lại từ đầu.

3. Fetcher Module

Fetcher chịu trách nhiệm tải nội dung HTML từ các url đã được từ Frontier thông qua giao thức HTTP. Với thiết kế đa luồng thì Fetcher có thể xử lý nhiều url mỗi phút. Để tối ưu cho module này, ta sẽ cần xử lý phần gửi request đến server của trang báo.

DNS Resolver chính là vấn đề ta cần quan tâm ở phần này vì các yêu cầu DNS có thể mất thời gian phản hồi có thể dao động từ 10 – 200ms. Giải pháp ở đây chính là cache lại DNS để tránh phải gọi thường xuyên, và sẽ reset lại cache nếu như quá thời gian. Ngoài ra module này cũng sẽ có cơ chế timeout và retry để đảm bảo tính ổn định, các url không phản hồi sẽ được gửi lại về Frontier, nhưng sẽ cần có thời gian để thực hiện thêm vào lại. Sẽ có một cronjob riêng phụ trách việc xử lý retry url bên phía Frontier.

Sau khi có được nội dung trang web và chắc chắn nó là dạng text hoặc HTML sẽ thực hiện cập nhật thông tin cho url vừa thu thập được. Nếu có lỗi thì sẽ lưu lại mã lỗi và số lần retry (hình 6). Phần HTML được lưu có thể sử dụng cho nhiều mục đích khác nhau sau này ví dụ như kiểm tra trùng lặp HTML nếu trùng thì không cần tiếp tục nữa, hoặc là kiểm tra version của bài viết nếu có thay đổi, … Đồng thời sau một khoảng thời gian cố định, hoặc đạt đủ số lượng url (ví dụ như 10) đã tải HTML từ cùng một domain thì sẽ cập nhật lại thông tin của domain về lần gần nhất crawl thông qua API từ frontier. Các Url sau bước này dù có lỗi hay không thì cũng sẽ lưu lại thông tin "status code" và "retry_count" vào Redis Cache, thông tin này sẽ được lưu với key là url được hash. Sử dụng Redis Cache ở phần này do sẽ cần kiểm tra trùng lặp ở module sau, hơn nữa Redis cũng hỗ trợ rất tốt cho hệ thống phân tán. Phần HTML thô sẽ được gửi đến Parser qua kafka với topic "parsing_tasks", có dùng thuật toán nén zstd. Một điểm hay của Kafka là khi dùng tính năng nén dữ liệu thì chỉ cần cấu hình ở producer, các consumer sẽ tự nhận và giải nén các message, rất phù hợp cho việc gửi nội dung text dài.



Hình 6: Lưu trữ thông tin về url

4. Parser Module

Parser có thể coi là trái tim của hệ thống này, là nơi phân tích và trích xuất dữ liệu có cấu trúc từ nội dung HTML. Việc thu thập từ các domain khác nhau, mỗi domain sẽ có giao diện khác nhau chính là vấn đề khó nhất để trích xuất đúng nội dung. Chính vì vậy module này ứng dụng mô hình ngôn ngữ lớn (LLM) để nhận diện các trường thông tin cần thiết như tiêu đề, nội dung, tác giả và thời gian đăng bài. Nhờ AI, parser

có thể tự thích ứng với sự thay đổi cấu trúc của các trang báo mà không cần sửa code thủ công.

Với dự án này, phần Parser sẽ có 3 phần chính. Thứ nhất là trích xuất các trang categories quan trong của các domain, thứ hai là trích xuất nội dung bài báo, thứ ba là trích xuất thêm các url khác. Công nghệ sử dụng để trích xuất trong module này là Jsoup, một thư viện mã nguồn mở, được thiết kế để thu thập và phân tích (parse) các tài liệu HTML từ web hoặc các nguồn khác. Nó cung cấp một API mạnh mẽ và dễ sử dụng để trích xuất dữ liệu, được sử dụng để trích xuất dữ liệu từ trang web tĩnh, khá phù hợp với các trang báo điện tử.

Phần đầu tiên là trích xuất các trang categories, đây chính là các trang điều hướng, trang danh mục của các trang web, nó sẽ chứa các bài báo theo từng loại, kết hợp với trang chủ thì đây chính là con đường nhanh nhất để tìm ra các bài báo mới. Giải pháp cho phần này chính là lấy ra các thẻ có liên quan đến điều hướng, ví dụ như có tên thẻ, tên class, id là “nav”, “menu”, “header-nav”, “site-map”, … sau đó sẽ lấy toàn bộ url có trong các thành phần đó, đi qua một bộ lọc với regex bao gồm kiểm tra xem có phải url dẫn đến một file ảnh, video không, hay có phải url của một bài báo không, hay có phải là url dẫn đến các trang như đăng nhập đăng kí không. Khi qua bộ lọc đó ta sẽ có được các trang categories của domain, các trang này sẽ được cập nhật thông qua API của Frontier.

Tiếp đến là trích xuất nội dung, một bài toán được đặt ra ở đây là làm sao để có thể biết đâu là url của một bài báo vì chúng ta không thể thực hiện tất cả url một cách giống nhau được, sẽ tốn rất nhiều tài nguyên và trả về rất nhiều lỗi. Giải pháp đưa ra là sẽ có một bộ lọc Heuristic để kiểm tra url hiện tại có là một bài báo không. Bộ lọc sẽ kiểm tra qua regex xem đây có những đặc điểm của bài báo không. Ví dụ như phần đường dẫn có chứa nhiều ký tự, kết thúc bằng đuôi html, htm, tpo, …, có chứa timestamp, có chứa nhiều số, chứa nhiều chữ kết hợp với số, … Đây chính là đặc điểm của các trang báo. Sau khi đã qua bộ lọc này ta sẽ thực hiện trích xuất nội dung thông qua CSS Selector.

CSS Selector của mỗi domain sẽ được sinh mới khi lần đầu crawl bài báo thuộc domain đó. Phần Config được sinh ra sẽ được lưu lại vào database với các trường thông tin như hình 7. Đồng thời ta sẽ sử dụng chiến thuật Cache Aside ở đây để lưu lại config CSS Selector của domain.

Hình 7: Lưu trữ thông tin CSS Selector cho domain

Để sinh được CSS Selector không đơn giản là ta chỉ cần gửi toàn bộ HTML của bài báo đến LLM, ta sẽ cần tối ưu token đầu vào bằng cách cắt bỏ đi phần nội dung không cần thiết như các thẻ "script", "noscript", "head", "link" các thành phần liên quan đến "nav", "menu", "social", "advertisment", "banner", "comment", "share"… Ngoài ra việc chọn mô hình LLM nào cũng cần phải tính toán. Dưới đây sẽ là chi tiết về hiệu quả sau khi thực hiện tối ưu token đầu vào cũng như một so sánh chi tiết giữa các mô hình LLM.

Ta sẽ chuẩn bị một bài test nhỏ với 4 domain khác nhau, thực hiện so sánh giữa trước và sau tối ưu token, so sánh giữa các mô hình Gemini, Mistral, Llama, Gemma.

```
urls = [
    'https://vnexpress.net/khong-quan-ukraine-tuyen-bo-ban-ha-tiem-kich-su-35-nga-4895862.html',
    'https://dantri.com.vn/xa-hoi/bo-chinh-tri-sap-nhap-tinh-dong-thoi-voi-hop-nhat-xa-o-noi-du-dieu-kien-20250607170102455
    'https://thanhnien.vn/dang-nha-nuoc-luon-quan-tam-cong-dong-nguoi-viet-o-nuoc-ngoai-185250607171121924.htm',
    'https://tienphong.vn/nhan-xet-giai-de-de-ngu-van-ha-noi-de-tho-nhung-kho-dat-diem-9-post1749112.tpo'
]

raw_htmls = []
for url in urls:
  response = requests.get(url)
  soup = BeautifulSoup(response.content, 'html.parser')
  html = soup.prettify()
  raw_htmls.append(html)
```

Hình 8: Các bài báo thuộc các domain khác nhau cần test

```
models = {
    "mistralai/devstral-small:free": "mistralai/Devstral-Small-2505",
    "meta-llama/llama-3.3-70b-instruct:free": "meta-llama/Llama-3.3-70B-Instruct",
    "google/gemma-3-27b-it:free": "google/gemma-3-27b-it",
    "gemini-2.0-flash-lite-001": "gemini-2.0-flash-lite",
    "gemini-2.0-flash-001": "gemini-2.0-flash",
    "gemini-2.5-flash-preview-05-20": "gemini-2.5-flash-preview",
}

model_names = {
    "mistralai/devstral-small:free": "Devstral-Small",
    "meta-llama/llama-3.3-70b-instruct:free": "Llama-3.3-70B",
    "google/gemma-3-27b-it:free": "Gemma-3-27b",
    "gemini-2.0-flash-lite-001": "Gemini-2.0-flash-lite",
    "gemini-2.0-flash-001": "Gemini-2.0-flash",
    "gemini-2.5-flash-preview-05-20": "Gemini-2.5-flash-preview",
}
```

Hình 9: Các mô hình cần test

```
def build_prompt(html: str):
  return f"""You are a CSS selector expert. Analyze this cleaned HTML from a news website and return ONLY a JSON object wit

HTML: {html}

Return JSON with these exact keys:
{{
  "title": "best CSS selector for article title",
  "content": "best CSS selector for main article content",
  "author": "best CSS selector for author name",
  "date": "best CSS selector for publish date"
}}

Rules:
- Use null if element not found
- Prefer class/id selectors over complex hierarchies
- Choose selectors that would work for similar pages
- No explanations, only JSON"""
```

Hình 10: Prompt dùng để lấy kết quả từ LLM

STT	Model	Content 1	Content 2	Content 3	Content 4
1	mistralai/Devstral-Small-2505	74601	148634	216730	28946
2	meta-llama/Llama-3.3-70B-Instruct	74601	148634	216730	28946
3	google/gemma-3-27b-it	96190	187469	284612	28555
4	gemini-2.0-flash-lite	74601	148634	216730	28946
5	gemini-2.0-flash	74601	148634	216730	28946
6	gemini-2.5-flash-preview	74601	148634	216730	28946

Hình 11: Lượng token đầu vào ước tính với mỗi mô hình trước khi tối ưu

STT	Model	Content 1	Content 2	Content 3	Content 4
1	mistralai/Devstral-Small-2505	3633	7419	8446	12509
2	meta-llama/Llama-3.3-70B-Instruct	3633	7419	8446	12509
3	google/gemma-3-27b-it	3704	7226	8218	11101
4	gemini-2.0-flash-lite	3633	7419	8446	12509
5	gemini-2.0-flash	3633	7419	8446	12509
6	gemini-2.5-flash-preview	3633	7419	8446	12509

Hình 12: Lượng token đầu vào ước tính với mỗi mô hình sau khi tối ưu

STT	Model	Content 1	Content 2	Content 3	Content 4
1	mistralai/Devstral-Small-2505	95.13%	95.01%	96.1%	56.79%
2	meta-llama/Llama-3.3-70B-Instruct	95.13%	95.01%	96.1%	56.79%
3	google/gemma-3-27b-it	96.15%	96.15%	97.11%	61.12%
4	gemini-2.0-flash-lite	95.13%	95.01%	96.1%	56.79%
5	gemini-2.0-flash	95.13%	95.01%	96.1%	56.79%
6	gemini-2.5-flash-preview	95.13%	95.01%	96.1%	56.79%

Hình 13: Lượng token đã giảm được khi thực hiện tối ưu

Có thể thấy trước khi tối ưu, lượng token đầu vào rất lớn hầu như đều trên mức 20,000 token, thậm chí có những trang còn ở mức hơn 200,000 token. Điều này là vượt quá giới hạn đầu vào của rất nhiều mô hình hiện nay. Nhưng sau khi thực hiện các bước lọc thông tin thì lượng token đã giảm xuống tương đối nhiều, gần như là giảm trên 90%, đối với web site đặc biệt, có cấu trúc phức tạp hơn, thì có thể sẽ không giảm được nhiều nhưng vẫn giảm được hơn 50% cũng đã đủ để dùng được cho các mô hình hiện nay.

Sau khi đã lọc, ta sử dụng chính kết quả đó để thực hiện lấy ra CSS Selector, dưới đây là chi tiết bảng so sánh, giữa kết quả được lấy bằng tay và lấy bằng LLM

Field	Reality Result	Devstral-Small	Llama-3.3-70B	Gemma-3-27b	Gemini-2.0-flash-lite	Gemini-2.0-flash	Gemini-2.5-flash-preview
title	h1.title-detail	.title-detail	.title-detail	h1.title-detail	.title-detail	.title-detail	h1.title-detail
content	article.fck_detail	.fck_detail	.fck_detail	article.fck_detail p.Normal	.fck_detail	.fck_detail	article.fck_detail
author	p.Normal strong	None	p.Normal strong	p.Normal strong	.fck_detail p strong	p.Normal[style='text-align:right;'] > strong	article.fck_detail p.Normal strong
date	span.date	.date	.date	span.date	.header-content .date	.date	span.date
time_res		2039ms	1549ms	4258ms	862ms	758ms	6251ms

Hình 14: Bảng so sánh kết quả đối với trang web 1

Field	Reality Result	Devstral-Small	Llama-3.3-70B	Gemma-3-27b	Gemini-2.0-flash-lite	Gemini-2.0-flash	Gemini-2.5-flash-preview
title	h1.title-page detail	h1.title-page.detail	.title-page.detail	article.singular-container h1.title-page.detail	h1.title-page.detail	.title-page.detail	h1.title-page.detail
content	div.singular-content	div.singular-content	.singular-content	article.singular-container div.singular-content	div.singular-content	.singular-content	div.singular-content
author	.author-name b	div.author-name > a > b	.author-name b	div.author-meta a b	div.author-name a b	.author-name a b	.author-name b
date	time.author-time	time.author-time	.author-time	div.author-meta time.author-time	time.author-time	.author-time	time.author-time
time_res		3109ms	1844ms	4818ms	834ms	745ms	3103ms

Hình 15: Bảng so sánh kết quả đối với trang web 2

Field	Reality Result	Devstral-Small	Llama-3.3-70B	Gemma-3-27b	Gemini-2.0-flash-lite	Gemini-2.0-flash	Gemini-2.5-flash-preview
title	h1.detail-title > span[data-role='title']	.detail-title span[data-role='title']	.detail-title span[data-role='title']	h1.detail-title span[data-role='title']	.detail-title > span[data-role="title"]	.detail-title > span[data-role="title"]	h1.detail-title
content	div.detail-content	.detail-cmain div[data-role='content']	.detail-content	.detail-content[data-role='content']	.detail-content.afcbc-body[data-role="content"]	.detail-content[data-role="content"]	div.detail-content
author	div.author-info a.name	.detail-author a[name]	.author-info .name	.detail-author a[href*='/author/']	.detail-author .name	.detail-author .name	div.detail-author a.name
date	div.detail-time div[data-role='publishdate']	.detail-time div[data-role='publishdate']	[data-role='publishdate']	.detail-time div[data-role='publishdate']	.detail-time [data-role="publishdate"]	.detail-time [data-role="publishdate"]	div[data-role="publishdate"]
time_res		2374ms	2005ms	5344ms	1026ms	849ms	3101ms

Hình 16: Bảng so sánh kết quả đối với trang web 3

Field	Reality Result	Devstral-Small	Llama-3.3-70B	Gemma-3-27b	Gemini-2.0-flash-lite	Gemini-2.0-flash	Gemini-2.5-flash-preview
title	h1.article__title	.article__title.cms-title	.article__title	h1.article__title.cms-title	h1.article__title.cms-title	.article__title.cms-title	h1.article__title
content	div.article__body	.article__body.cms-body	.article__body	div.article__body.cms-body	div.article-content	.article__body.cms-body	div.article__body
author	.article__author .name	.cms-author	.article__author .name	span.cms-author	span.name.cms-author	.article__author .name.cms-author	span.cms-author
date	.article__meta time	time[datetime]	.article__meta .time time	time[datetime]	time[datetime]	.article__meta .time time	span.time time
time_res		3476ms	2360ms	5227ms	949ms	768ms	5279ms

Hình 17: Bảng so sánh kết quả đối với trang web 4

Model	Title	Content	Author	Date	Thời gian phản hồi
Mistral/Devstral-Small	Gần như chính xác	Gần như chính xác	Độ chính xác chưa cao	Gần như chính xác	2–3s
Llama-3.3-70B	Gần như chính xác	Gần như chính xác	Gần như chính xác	Gần như chính xác	1–2s
Gemma-3-27b	Chi tiết, gần như chính xác	Chi tiết, gần như chính xác	Chi tiết, gần như chính xác	Chi tiết, gần như chính xác	4–5s
Gemini-2.0-flash-lite	Gần như chính xác	Gần như chính xác	Gần như chính xác	Gần như chính xác	~1s
Gemini-2.0-flash	Gần như chính xác	Gần như chính xác	Gần như chính xác	Gần như chính xác	< 1s
Gemini-2.5-flash-preview	Chính xác gần như hoàn toàn	Chính xác gần như hoàn toàn	Chính xác gần như hoàn toàn	Chính xác gần như hoàn toàn	3–6s

Hình 18: Bảng tổng hợp kết quả các mô hình với 4 trang web

Từ kết quả trên ta thấy rằng Mô hình mistral có vẻ không phù hợp với bài toán Crawler này do độ chính xác chưa đạt kì vọng cũng như thời gian phản hồi chưa phải quá tốt. Mô hình Gemma 3 mặc dù độ chính xác rất cao, chi tiết, nhưng do đây mới chỉ là bài test nhỏ nên hiện tượng quá chi tiết này có thể sẽ dẫn đến lỗi nhiều trong tương lai, do cùng một domain có thể sẽ được thêm, bớt các CSS selector khác nhau. Và thời gian phản hồi của mô hình này cũng là một điểm trừ. Mô hình Gemini 2.5 Flash mặc dù kết quả rất tốt nhưng thời gian phản hồi cũng như giới hạn lượng token đầu vào và lượng request trong 1 phút, trong 1 ngày cũng sẽ là lý do ta không lựa chọn cho bài toán này. Về Mô hình Llama 3.3, Gemini 2.0 Flash, Gemini 2.0 Flash lite kết quả đều rất tốt, thời gian phản hồi đủ nhanh. Cả ba mô hình này đều có thể dùng được cho bài toán lần này. Với free tier, ta sẽ ưu tiên sử dụng Gemini 2.0 Flash hoặc Gemini Flash lite hơn.

Tuy nhiên phần kết quả của LLM trả về không phải lúc nào cũng chính xác, ta sẽ cần thực hiện ném lỗi nếu như bị thiếu thông tin cần thiết, Trong tương lai có thể phát triển thêm cơ chế tự động sinh lại config nếu như số lượng bài lỗi của một domain vượt quá một ngưỡng nhất định, hoặc người quản trị có thể quyết định có nên áp dụng thay đổi hay không dựa trên số lượng lỗi. Những bài đã được trích xuất nội dung thành công sẽ được trích xuất thêm các url có trong trang để loang thêm các bài viết khác.

Phần thứ ba chính là trích xuất các url, lọc các url không cần thiết để đưa lại vào Frontier. Phần trích xuất này sẽ lấy toàn bộ url có trong HTML thô, sẽ loại bỏ đi các url ở phần categories cũng như các url được nghi là bài báo với status code kiểm tra trong

Redis Cache là mã 200 hoặc số lần retry và thời gian retry chưa đủ điều kiện. Phần nội dung bài báo trích xuất được sẽ được gửi đế Data Store Module bằng Kafka với message cũng được nén bằng thuật toán zstd với topic "storing_tasks". Phần url mới sẽ được gửi đến Frontier qua Kafka với topic "new_urls".

5. Data Store Module

Data Store Module chịu trách nhiệm quản lý việc lưu trữ và truy xuất dữ liệu đã xử lý nhận từ Parser. Elasticsearch là một công cụ tìm kiếm phân tán mạnh mẽ, được thiết kế cho các truy vấn văn bản không cấu trúc với khả năng mở rộng theo chiều ngang, độ trễ thấp và khả năng đáp ứng nhanh chóng cho các hệ thống dữ liệu lớn. Các lý do chính để lựa chọn Elasticsearch bao gồm khả năng tìm kiếm full-text hiệu quả, hỗ trợ từ stemming, typo correction đến các truy vấn Boolean phức tạp. Tích hợp tốt với dữ liệu JSON, dễ dàng index và truy xuất các tài liệu không đồng nhất.

Không chỉ dừng lại ở việc tìm kiếm theo từ khóa, hệ thống còn hỗ trợ Semantic Search – một kỹ thuật cho phép truy vấn thông tin theo ý nghĩa, thay vì chỉ dựa vào sự trùng khớp từ vựng đơn thuần. Để hiện thực hóa chức năng này, toàn bộ nội dung văn bản được chuyển đổi thành vector thông qua kỹ thuật embedding. Các vector này sau đó được lưu trữ trực tiếp trong Elasticsearch và có thể được truy vấn thông qua cơ chế tìm kiếm gần đúng theo vector (approximate vector similarity search).

Hiện tại, hệ thống sử dụng mô hình "nomic-embed-text-v2" – một mô hình embedding đa ngôn ngữ, được triển khai thông qua Ollama. Đây là một mô hình dạng Mixture-of-Experts (MoE), có khả năng sinh vector hiệu quả cho khoảng 100 ngôn ngữ, được huấn luyện trên hơn 1.6 tỷ cặp văn bản. Với kiến trúc Matryoshka Embeddings, mô hình này cho phép giảm đáng kể dung lượng lưu trữ vector mà chỉ chịu tổn thất hiệu suất rất nhỏ. Đặc biệt, mô hình cho thấy hiệu quả tốt trong việc xử lý tiếng Việt – cả về độ chính xác ngữ nghĩa và khả năng phân biệt ngữ cảnh – điều này rất quan trọng trong các trường hợp tìm kiếm nội dung tin tức hoặc truy xuất văn bản chuyên ngành.

Quy trình embedding diễn ra như sau: nội dung bài báo và tiêu đề sẽ được xử lý để đưa vào mô hình embedding, mô hình này trả về một vector có 768 chiều. Vector này được gắn cùng metadata và sau đó được lưu trữ vào Elasticsearch. Mỗi khi có truy vấn dạng semantic, hệ thống sẽ thực hiện embedding truy vấn, rồi tìm kiếm các vector gần nhất trong không gian embedding đã lưu.

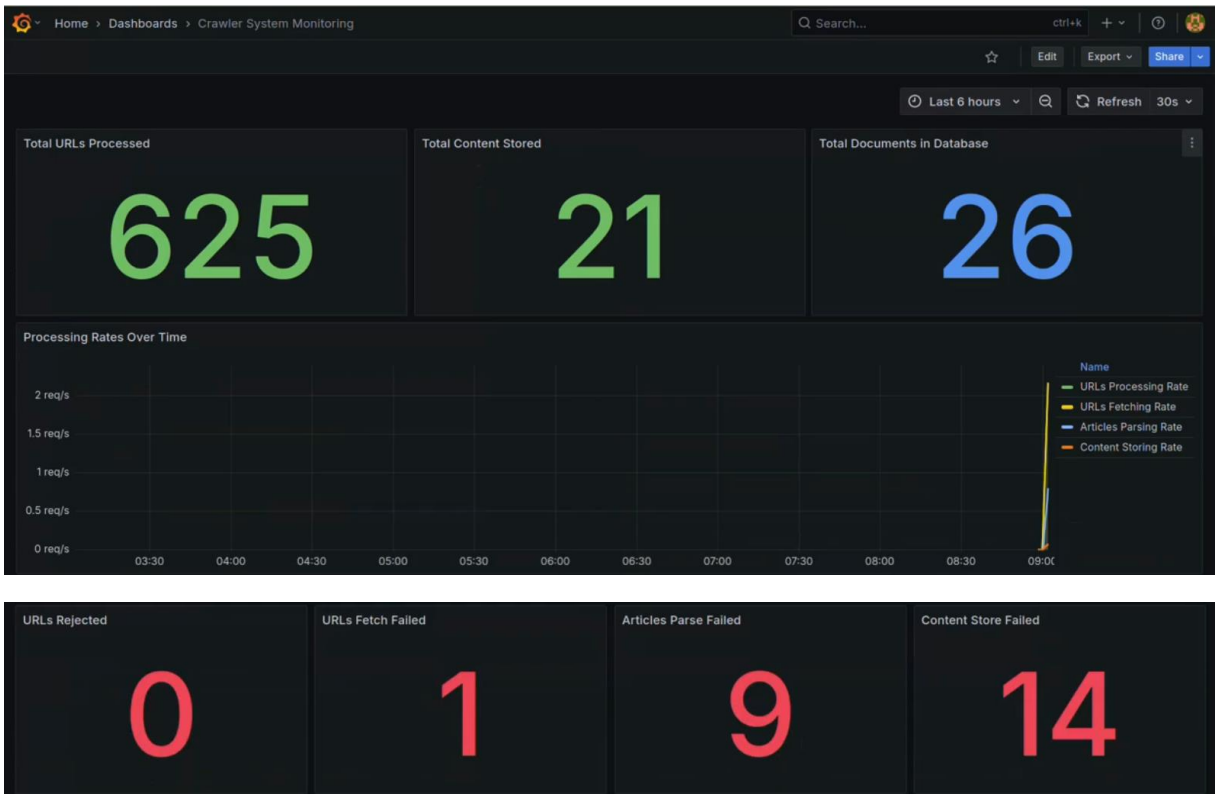
Hệ thống hiện cung cấp hai API tìm kiếm chính:

- Keyword Search API: hoạt động dựa trên các truy vấn truyền thống của Elasticsearch, sử dụng cấu trúc truy vấn "match", "term", hoặc "bool" để lọc chính xác các tài liệu chứa từ khóa hoặc cụm từ cụ thể.
- Semantic Search API: hoạt động bằng cách embedding câu truy vấn, sau đó tìm kiếm các vector tương đồng nhất theo cosine similarity trong không gian vector embedding.

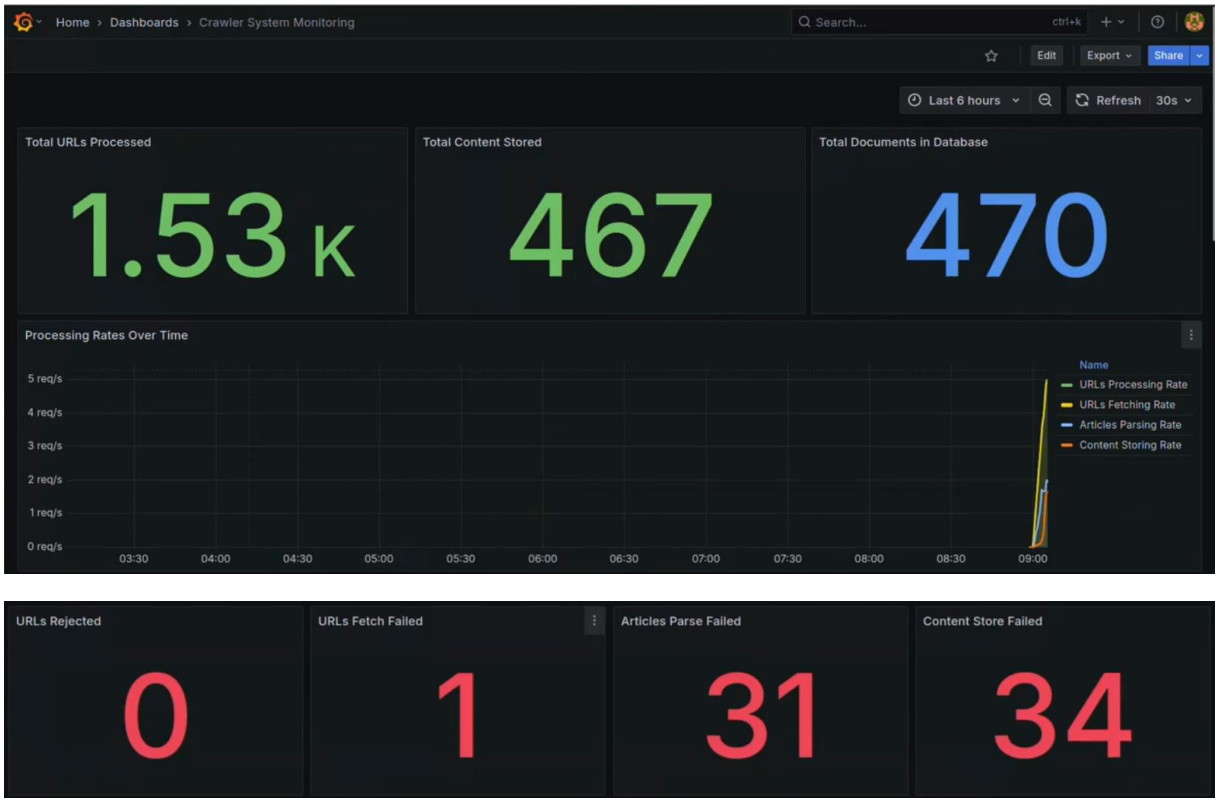
Nhờ sự kết hợp giữa hai phương pháp trên, ta có thể linh hoạt lựa chọn giữa việc tìm kiếm chính xác theo từ khóa và tìm kiếm theo ngữ nghĩa. Hệ thống đảm bảo hoạt động ổn định cho cả hai loại truy vấn, mang lại trải nghiệm tìm kiếm phong phú – từ việc theo dõi nội dung thời sự cho đến khai thác dữ liệu chuyên sâu phục vụ nghiên cứu và phân tích.

III. KẾT QUẢ THỰC NGHIỆM

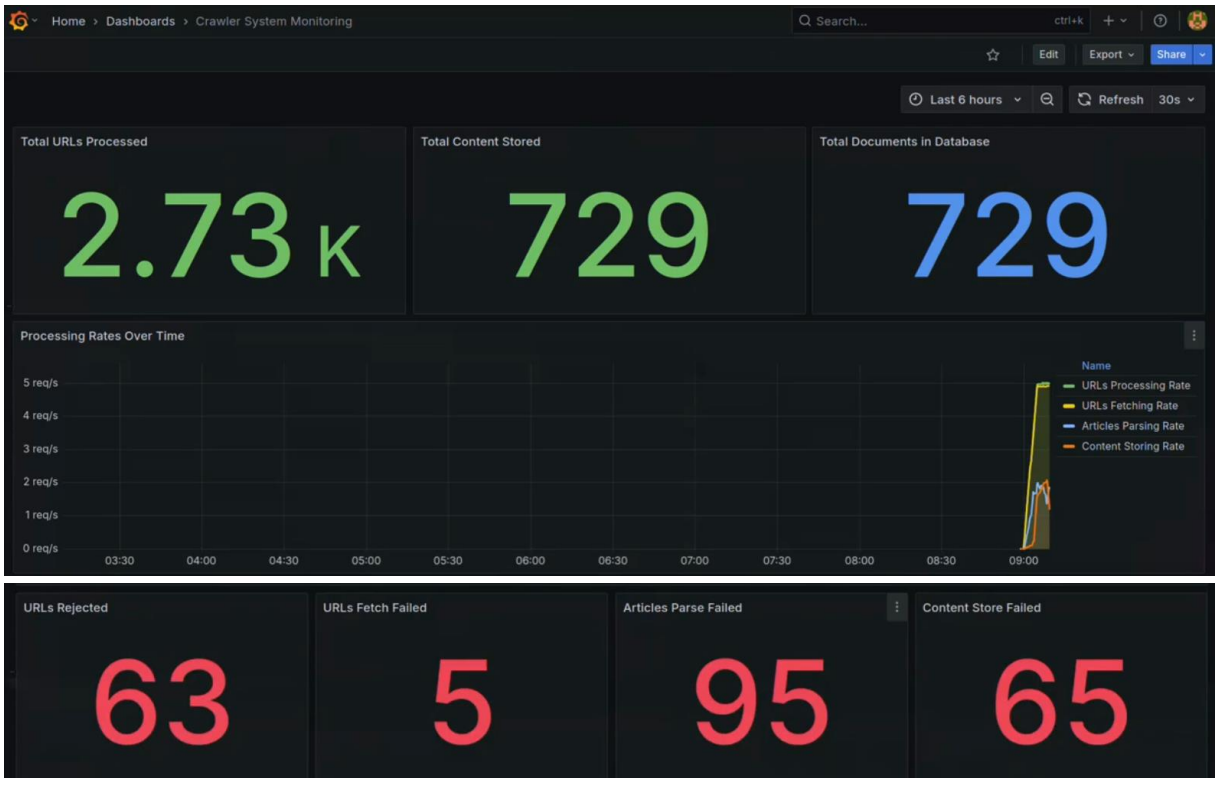
Việc đánh giá hiệu quả của hệ thống thu thập dữ liệu báo chí được thực hiện thông qua các chỉ số thống kê của hệ thống giám sát (monitoring). Hệ thống giám sát này sử dụng Prometheus để thu thập các metric là các chỉ số được expose từ các module, sau đó sẽ trực quan hóa bằng Grafana. Các chỉ số bao gồm tổng số url đã thực hiện trong lần chạy, số url đã thực hiện thành công, số url bị lỗi, số bài báo đã lưu, tổng số bài báo đã lưu từ trước đến hiện tại. Ngoài ra còn có các thống kê chi tiết về từng module, số lượng các url trên mỗi domain thực hiện thành công/lỗi trong một module. Dưới đây là một số hình ảnh thống kê của hệ thống khi chạy thử.



Hình 19: Thống kê kết quả sau 2 phút

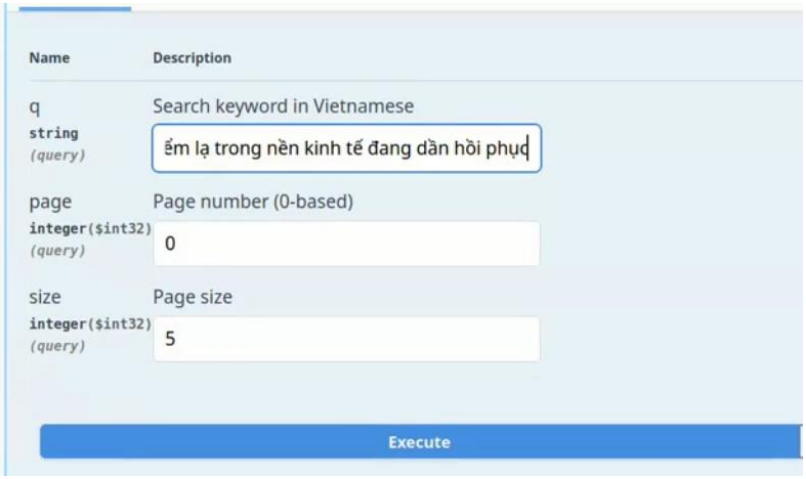


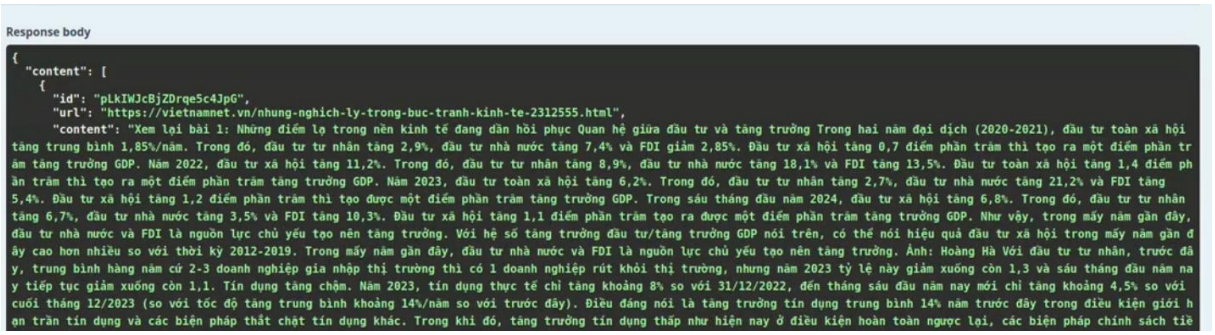
Hình 20: Thống kê kết quả sau 5 phút

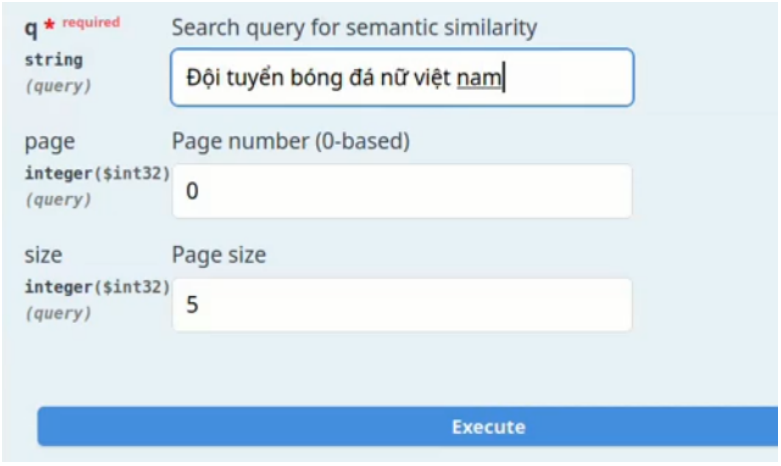


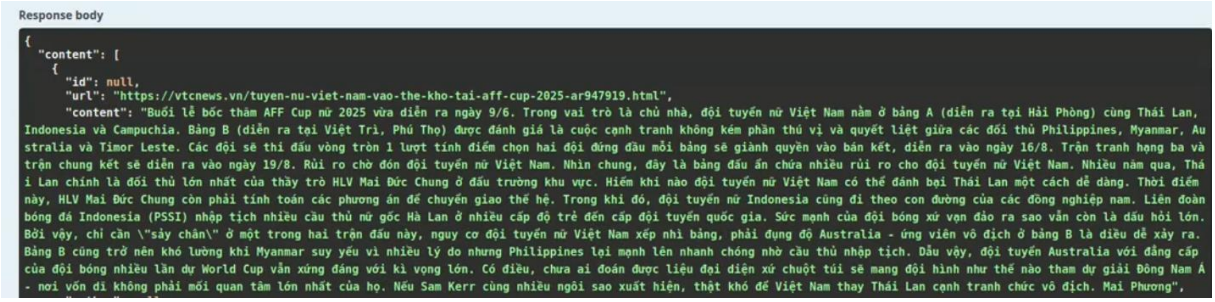
Hình 21: Thống kê kết quả sau 10 phút

Hình 22: Thống kê chi tiết về các module sau 10 phút









Hình 23: API search by key word/ semantic search

Về mặt hiệu suất, việc áp dụng công nghệ đa luồng đã giúp tăng tốc độ thu thập dữ liệu lên nhiều lần so với phương pháp tuần tự, tốc độ thu thập qua thống kê trên cho thấy khá ổn mặc dù mới chỉ ở khoảng 10 phút đầu. Độ chính xác trong việc trích xuất thông tin chưa quá ổn định tuy nhiên vẫn duy trì ở mức 70-80% ở các lần chạy khác nhau điều này cũng đã giải quyết được một phần bài toán thay thế phương pháp rule-based truyền thống. Hệ thống cũng đã ứng dụng được các rule để reject các url không được phép truy cập.

Về tính năng tìm kiếm, cả hai API gồm Keyword Search và Semantic Search hiện đang hoạt động ổn định. Phần Keyword Search sử dụng trực tiếp các truy vấn của Elasticsearch, cho phép truy vấn chính xác dựa trên từ khóa. Trong thử nghiệm với các tập dữ liệu đã crawl, phần Semantic Search cho kết quả truy vấn khá khớp về mặt ngữ nghĩa, đặc biệt hiệu quả trong các trường hợp câu hỏi không chứa chính xác từ khóa gốc của bài viết nhưng vẫn truy xuất được tài liệu liên quan.

IV. KẾT LUẬN

Qua quá trình nghiên cứu và triển khai, hệ thống đã chứng minh tính hiệu quả trong việc tự động thu thập và xử lý dữ liệu báo chí quy mô lớn. Việc áp dụng AI để thay thế cho rule-based truyền thống đã phần nào chứng mình được độ hiệu quả. Công nghệ đa luồng và lưu trữ phân tán đảm bảo hiệu năng và khả năng mở rộng.

Tuy nhiên, để đạt được độ tin cậy và ổn định cao hơn trong thực tế, vẫn còn nhiều yếu tố cần được tối ưu trong các giai đoạn tiếp theo. Trước hết là việc tăng cường độ chính xác của quá trình parser hỗ trợ bởi AI, đặc biệt trong việc xử lý các website có cấu trúc thay đổi liên tục. Bên cạnh đó, cần tiếp tục cải thiện thuật toán phân phối và đánh giá URL, chẳng hạn như xác định xem một URL có xứng đáng để trích xuất nội dung hay không, hoặc xây dựng chiến lược chọn Seed URLs hợp lý hơn nhằm tăng độ phủ dữ liệu và tránh trùng lặp.

V. TÀI LIỆU THAM KHẢO

1. Design a Web Crawler
2. Design a Web Crawler | Book Notes
3. yasserg/crawler4j: Open Source Web Crawler for Java