VIETTEL DIGITAL TALENT 2025

Software Engineer

BÁO CÁO MINI PROJECT

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: ViettelAI

NỘI DUNG CHÍNH

1. Giới thiệu
2. Thiết kế và triển khai hệ thống
3. Kết quả thực nghiệm
4. Kết luận

GIỚI THIỆU

VỀ ĐỀ TÀI: HỆ THỐNG THU THẬP DỮ LIỆU BÁO CHÍ

- Tìm hiểu công nghệ đa luồng
- Tìm hiểu công nghệ lưu trữ và xử lý phân tán
- Tích hợp AI để phân tích cấu trúc HTML từ đó trích xuất thông tin

GIỚI THIỆU

MỤC TIÊU & PHẠM VI NGHIÊN CỨU:

- Cần xây dựng hệ thống mở rộng được, hiệu suất cao
- Thiết kế và triển khai các module chính của hệ thống
- Tự động thu thập và xử lý dữ liệu từ các báo điện tử tiếng Việt

Thiết kế và triển khai hệ thống

Một hệ thống crawler cơ bản sẽ bao gồm 4 phần chính với các nhiệm vụ khác nhau tạo thành 1 pipeline

Chứa các url cần thực hiện lấy nội dung. Lên lịch, điều phối để xử lý các url

Thực hiện gửi yêu cầu, tải xuống nội dung HTML của trang web

Trích xuất nội dung theo một rule đã đặt sẵn

Lưu trữ nội dung đã trích xuất

Thiết kế và triển khai hệ thống

Một hệ thống crawler cơ bản sẽ bao gồm 4 phần chính với các nhiệm vụ khác nhau tạo thành 1 pipeline

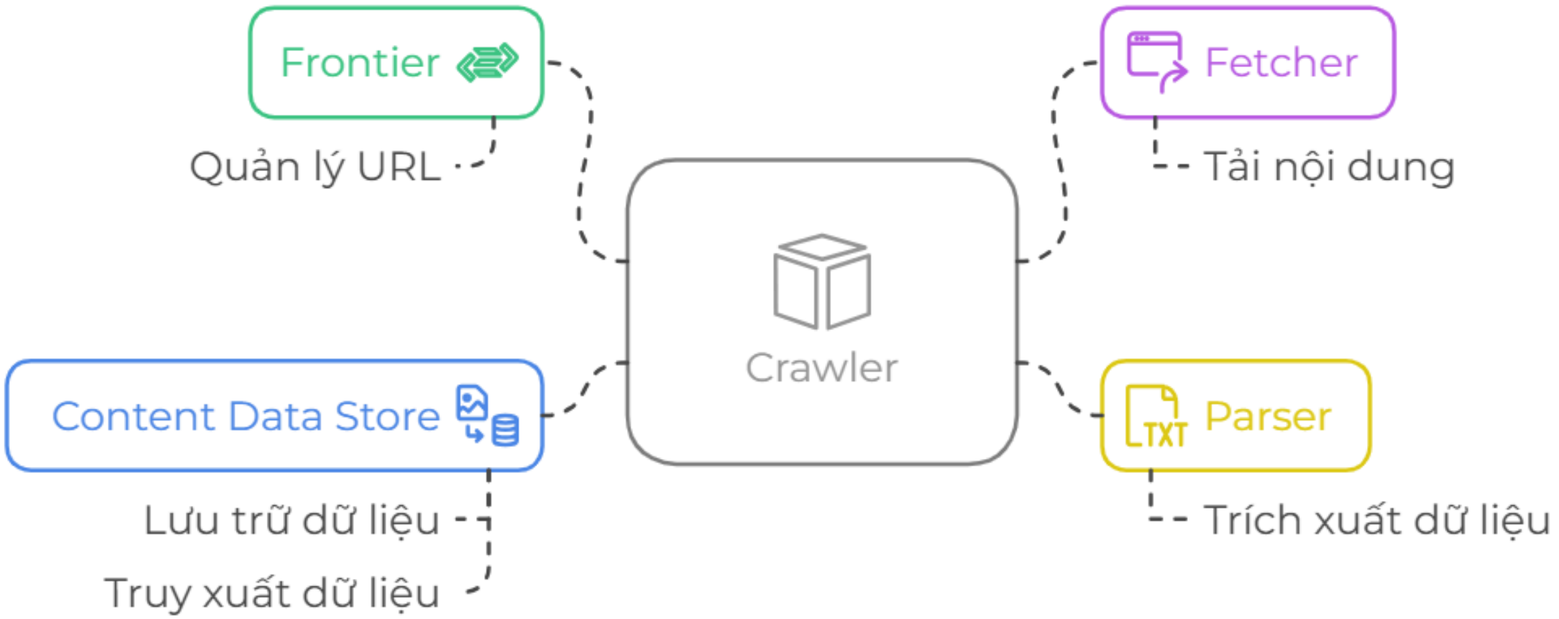
Với yêu cầu bài toán thì sẽ cần mở rộng thiết kế này, cần đảm bảo:

- Cơ chế điều phối thực hiện một cách hợp lý
- Cơ chế xử lý nếu gặp lỗi
- Tốc độ xử lý, lưu trữ, truy xuất thông tin
- Hệ thống có thể mở rộng dễ dàng

Thiết kế và triển khai hệ thống

=> Giải pháp: Xây dựng một hệ thống microservices. Mỗi module có thể mở rộng, khởi động độc lập

Các module chính của hệ thống:



Thiết kế và triển khai hệ thống

=> Giải pháp: Xây dựng một hệ thống microservices. Mỗi module có thể mở rộng, khởi động độc lập

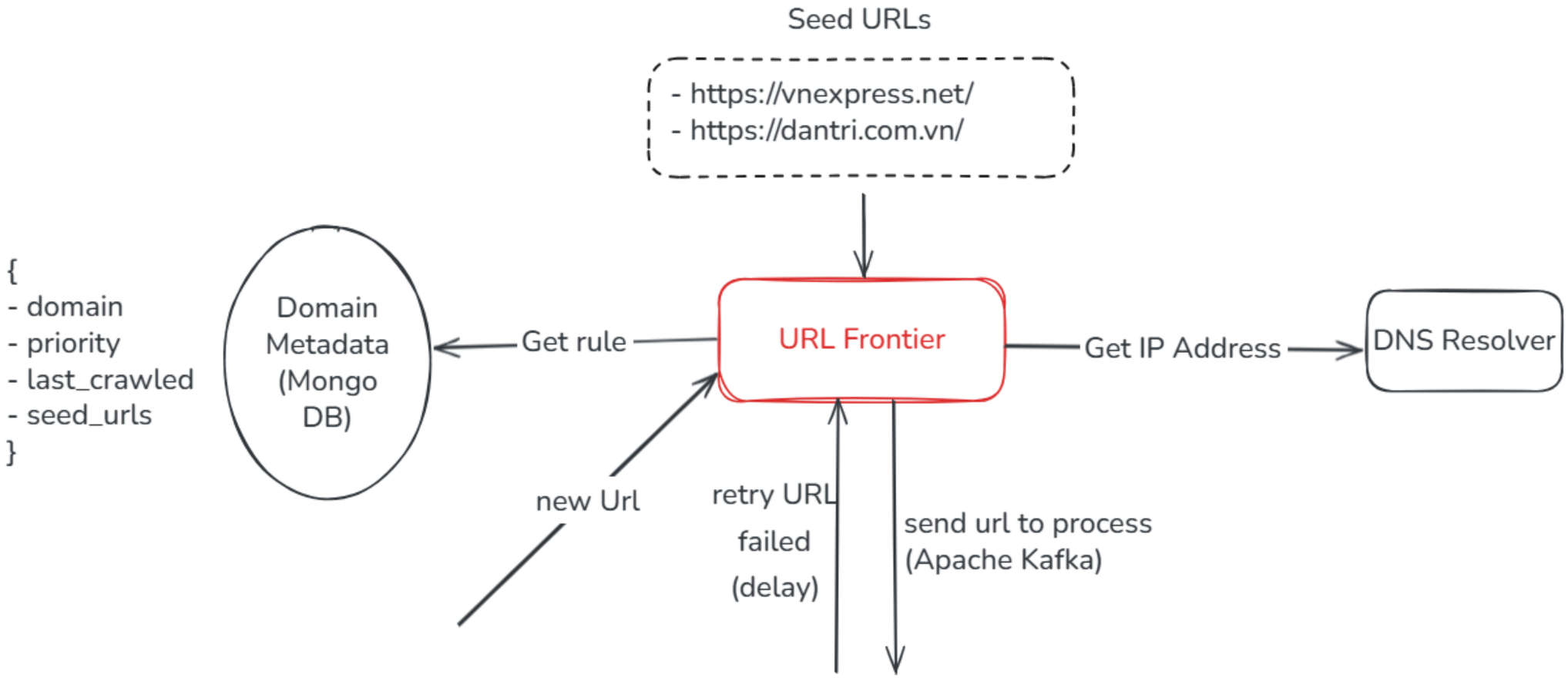
Các công nghệ sử dụng:

- **Java Spring boot**
- **Kafka:** Truyền message giữa các module. Hỗ trợ xử lý real-time, phân tán, bền vững
- **Redis:** Caching trạng thái, kiểm tra trùng lặp
- **Jsoup:** Thư viện parse nội dung cơ bản
- **LLM:** Sinh Css Selector cho các trang khác nhau
- **MongoDB:** Linh hoạt schema, tối ưu read/write
- **Elasticsearch:** Hỗ trợ search tốc độ cao
- **Prometheus, Grafana:** Monitoring

Thiết kế và triển khai hệ thống

Frontier Module

Quản lý, điều phối danh sách url, nhận url mới, url lỗi cần thực hiện lại



Thiết kế và triển khai hệ thống

Frontier Module

Frontier truyền thống:

- Chỉ là một danh sách, tập hợp đơn giản
- Có nhiều điểm hạn chế (priority, politeness)

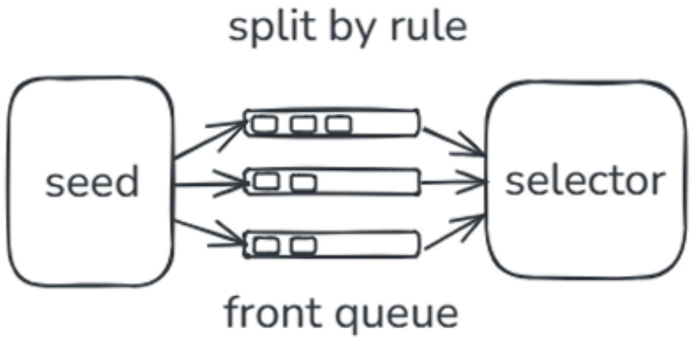
Vấn đề cần giải quyết:

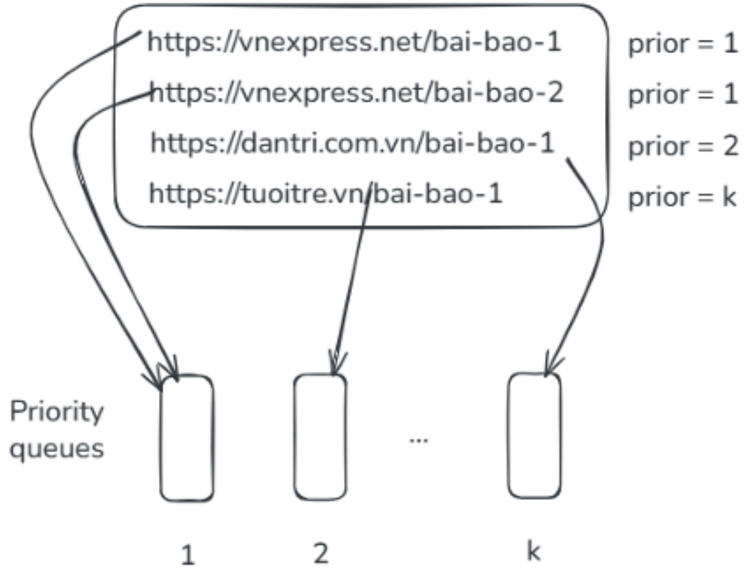
- Không thực hiện crawl hoàn toàn theo DFS hay BFS, cần phải xử lý theo độ ưu tiên giữa các domain
- Cần có rate limit giữa các lần thực hiện trên cùng 1 domain (đảm bảo politeness)

Frontier Module

Giải pháp cho vấn đề độ ưu tiên:

- Tạo 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
- Selector sẽ thực hiện lấy ra một loạt các url từ các queue có độ ưu tiên lớn hơn trước



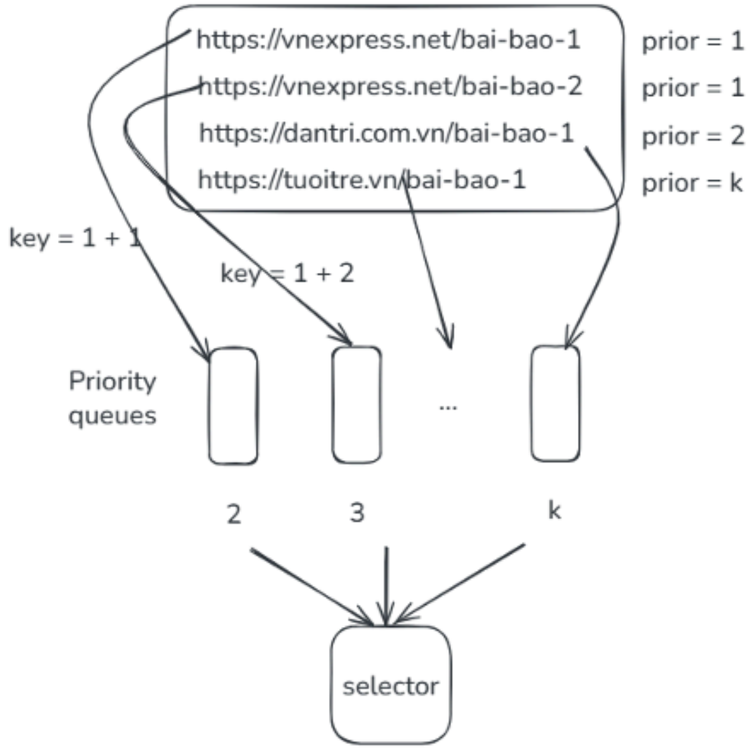


Frontier Module

Giải pháp cho vấn đề độ ưu tiên:

- Sử dụng cách tính key khác, thêm cơ chế fallback nếu đã đầy queue
- Áp dụng Round Robin có trọng số cho Selector, để tránh các url sẽ chờ nhau quá lâu

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

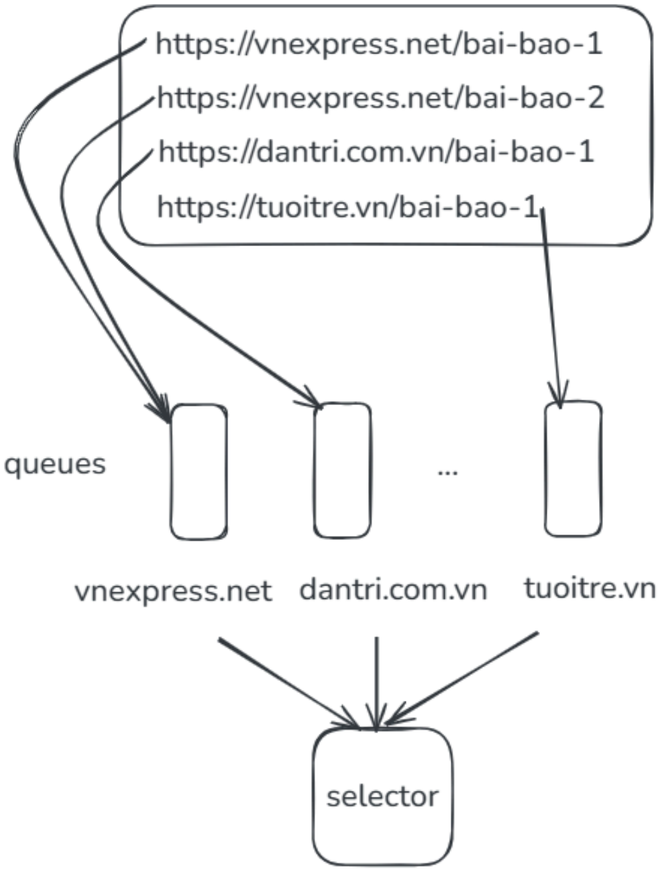


Thiết kế và triển khai hệ thống

Frontier Module

Giải pháp cho vấn đề Politeness:

- Tạo một bảng hash với key là domain, value sẽ là một priority queue
- Selector sẽ thực hiện lấy ra một loạt các url từ các queue theo thuật toán Round Robin, đảm bảo có rate limit để tránh bị chặn bởi các trang web



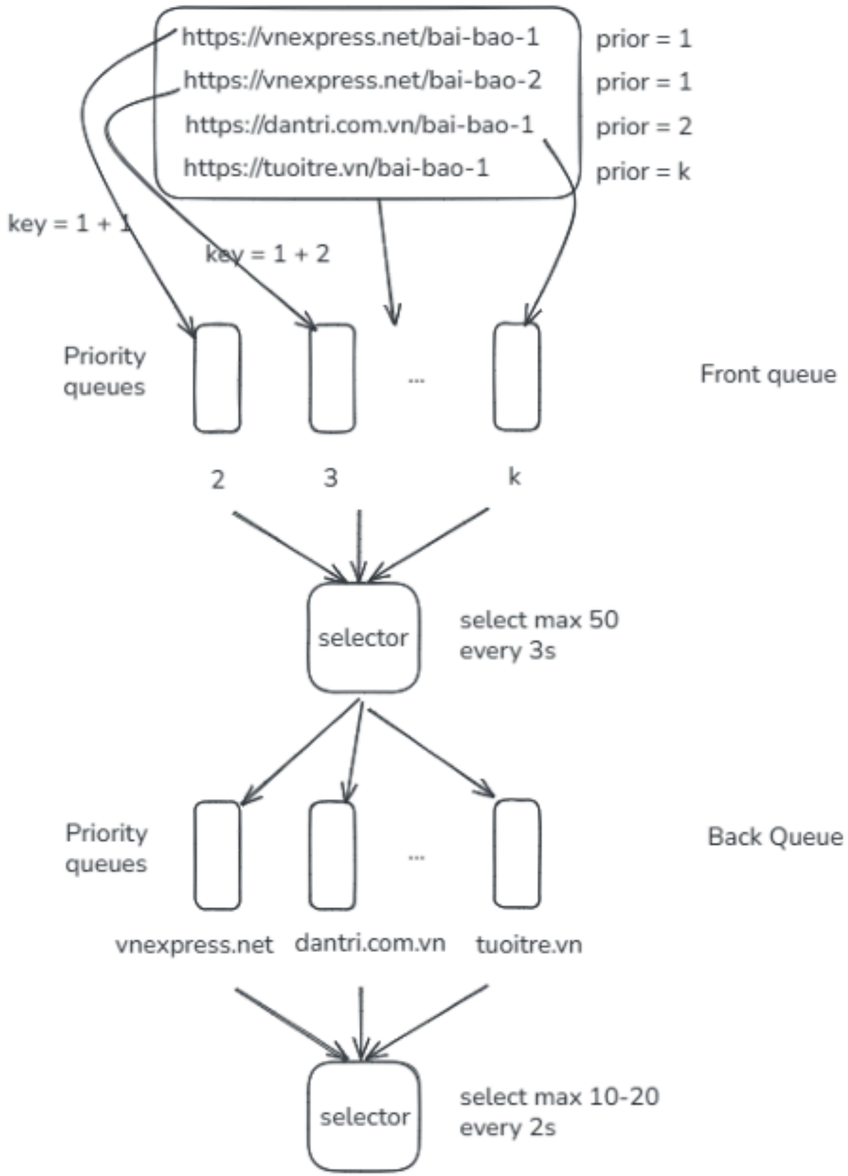
Thiết kế và triển khai hệ thống

Frontier Module

Kết hợp lại ta có một Frontier hoàn chỉnh như sơ đồ bên. Các bước kiểm tra về độ ưu tiên, quyền truy cập, crawl delay sẽ thực hiện truy vấn qua database và lấy nội dung từ robots.txt của các trang web.

Ngoài ra sẽ có thêm các cơ chế:

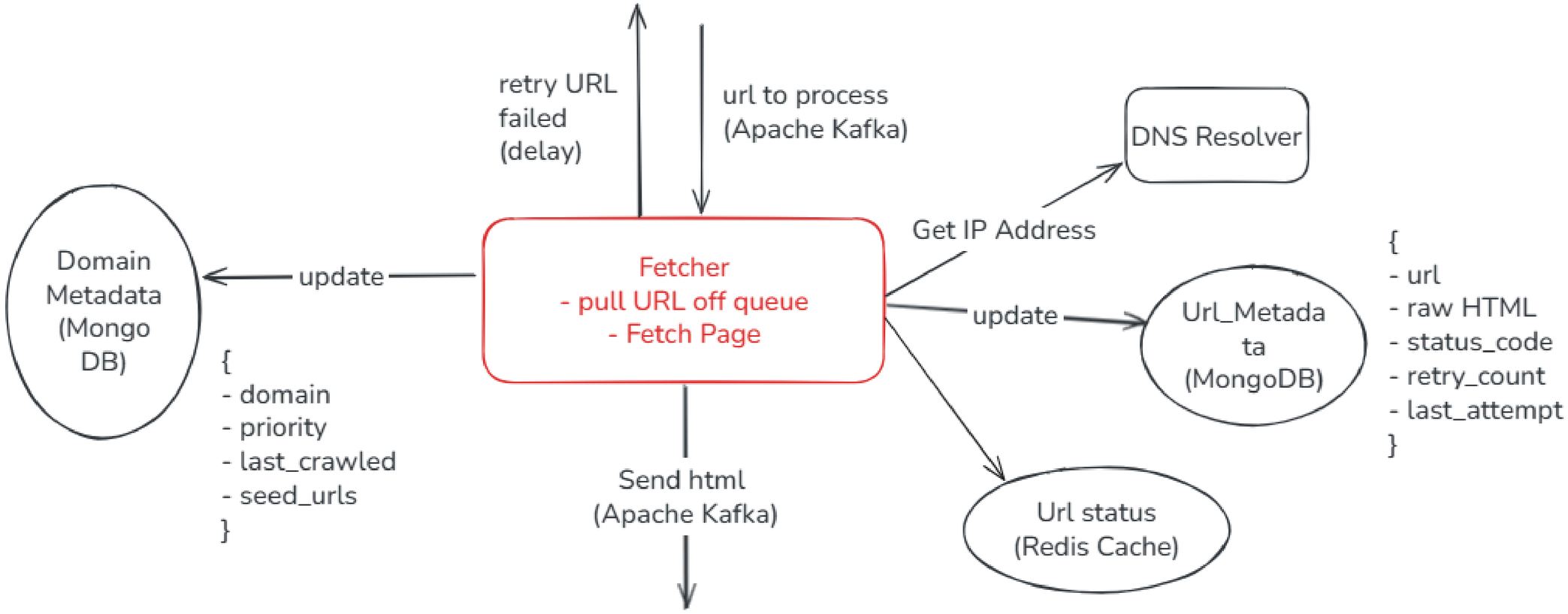
- Một Hash map để kiểm tra xem url mới thêm vào đã có trong bất kì queue nào hay chưa
- Kiểm tra domain của url mới (kiểm tra cả về subdomain)



Thiết kế và triển khai hệ thống

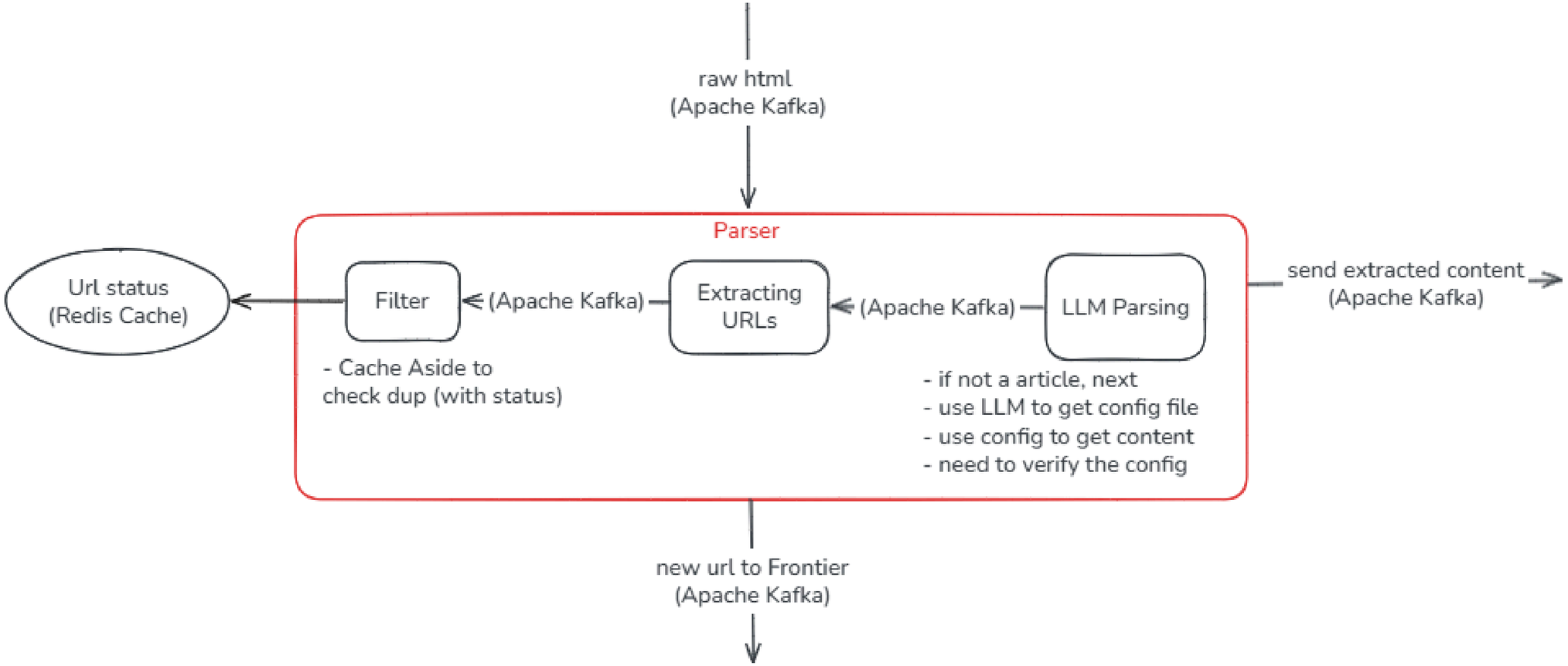
Fetcher Module

Tải nội dung HTML từ các url đã được từ Frontier thông qua Kafka thông qua giao thức HTTP. Để tối ưu ta có thể xử lý cache DNS Resolver để giảm độ trễ khi thực hiện request



Parser Module

Phân tích và trích xuất dữ liệu từ nội dung HTML, tìm thêm các url mới để thực hiện tiếp



Parser Module

Parser có thể trích xuất được các trang categories thông qua việc lấy các thẻ “nav”, “menu”, “header”, v.v. từ HTML. Sau đó lọc các link có trong đó bằng bộ lọc Heuristic.

Sau khi có được một tập các trang categories sẽ cập nhật các kết quả có được vào Database thông qua API để phục vụ các lần crawl sau này

Để giúp cho Parser không cần parse nội dung cho tất cả các url thì ta cần lọc ra đâu là url của một bài báo, tiếp tục có một bộ lọc Heuristic ở đây. Qua quá trình lọc sẽ giúp tiết kiệm được phần lớn thời gian.

Parser Module

Điểm chính của module này là trích xuất thông tin. Thay vì cần cấu hình thủ công cho từng domain thì AI sẽ làm thay việc đó. Các thông tin cấu hình về Css Selector sẽ được lưu lại vào database cũng như cache để tiết kiệm thời gian xử lý cho module.

Dưới đây là bài đánh giá cho phương pháp tối ưu token đầu vào khi gọi LLM cũng như so sánh độ chính xác, khả năng của các mô hình LLM khác nhau

Dữ liệu test sẽ gồm:

- 4 url bài báo từ các domain khác nhau
- 6 mô hình LLM khác nhau

Parser Module

Phần trăm lượng token giảm được qua bộ lọc

	Url 1	Url 2	Url 3	Url 4
Mistral	95%	95%	96%	56%
Llama 3.3 70B	95%	95%	96%	56%
Gemma 3 27B	96%	96%	97%	61%
Gemini 2.0 Flash	95%	95%	96%	56%
Gemini 2.0 Flash lite	95%	95%	96%	56%
Gemini 2.5 Flash	95%	95%	96%	56%

Parser Module

So sánh tổng quan kết quả đạt được của các mô hình

	Độ chính xác của các trường thông tin	Thời gian phản hồi
Mistral	Đa số đều gần như chính xác	2-3s
Llama 3.3 70B	Đa số đều gần như chính xác, mức độ chi tiết vừa phải	1-2s
Gemma 3 27B	Đa số đều gần như chính xác, cân nhắc vì quá chi tiết	4-5s
Gemini 2.0 Flash	Đa số đều gần như chính xác	~1s
Gemini 2.0 Flash lite	Đa số đều gần như chính xác	<1s
Gemini 2.5 Flash	Các kết quả gần như đúng hoàn toàn	3-6s

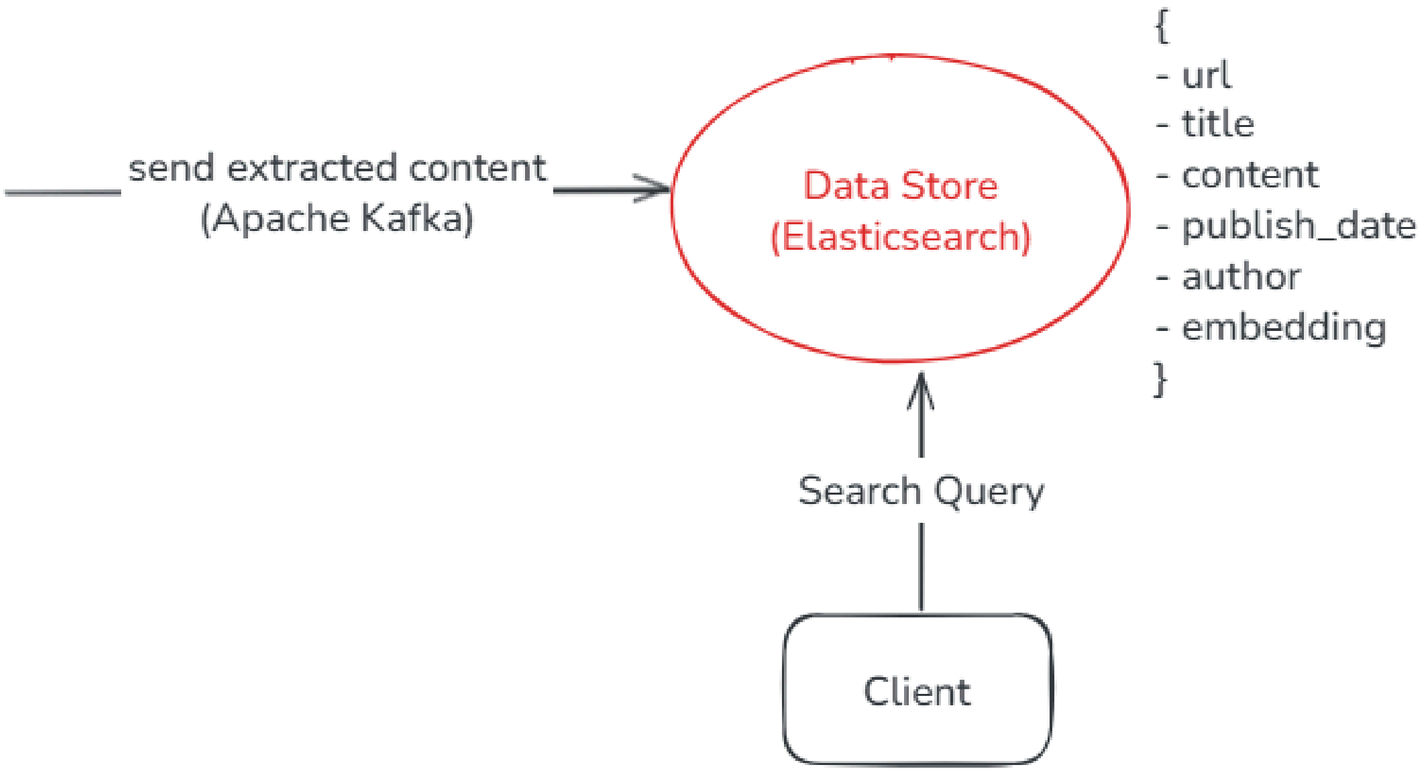
Parser Module

Phần cuối cùng của module là gửi các url mới đến Frontier:

- Trích xuất tất cả các url có trong HTML
- Loại bỏ đi những url là categories
- Lọc ra các url là trang báo đã thành công theo cache aside

Data Store Module

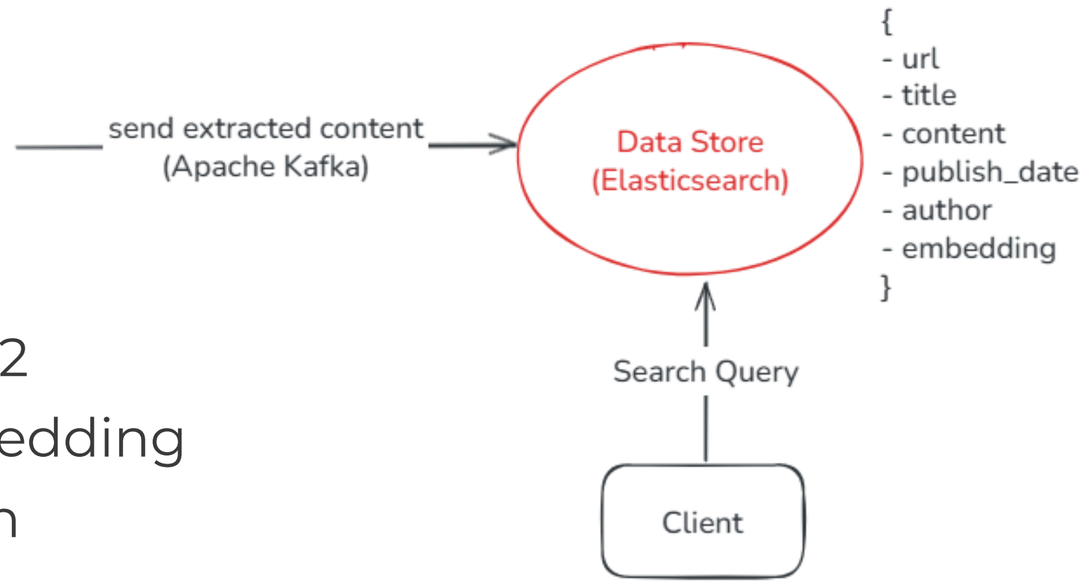
Quản lý việc lưu trữ và truy xuất dữ liệu đã xử lý



Thiết kế và triển khai hệ thống

Data Store Module

Công việc thực hiện:

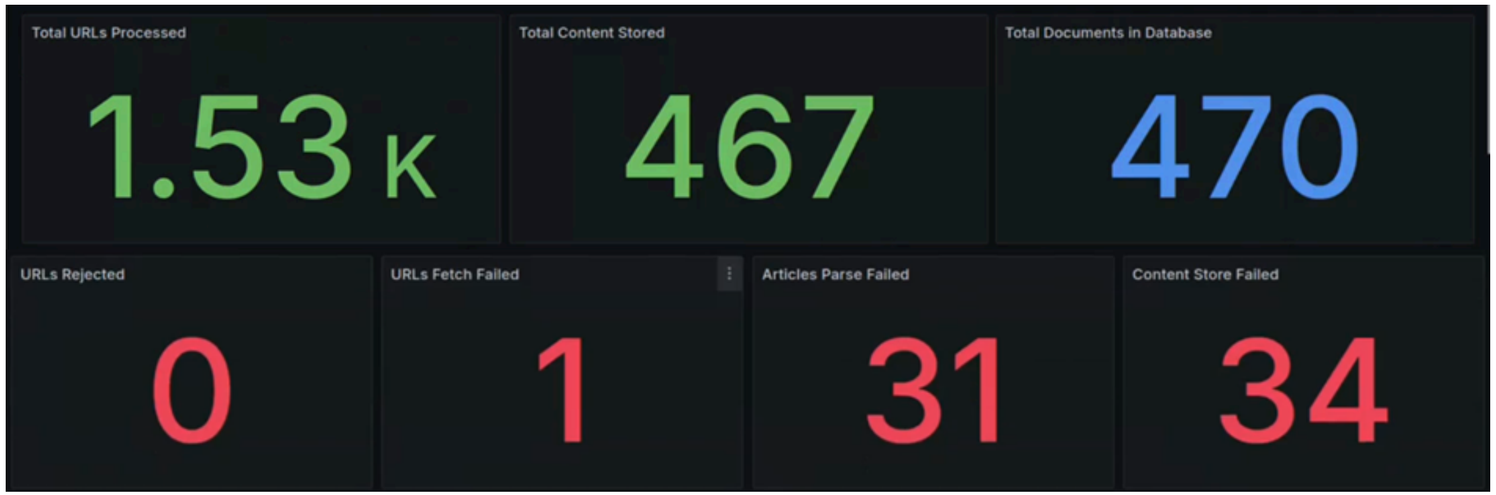


Sử dụng mô hình nomic-embed-text-v2
Xử lý nội dung → đưa vào mô hình embedding
→ trả về vector 768 chiều → Elasticsearch

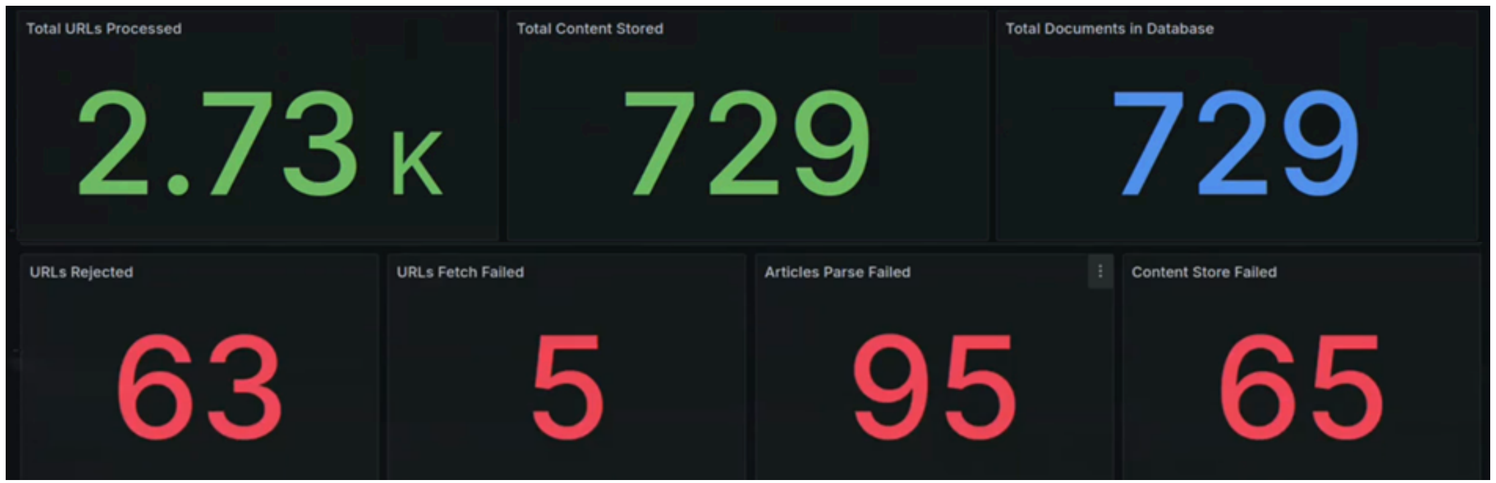
- **Keyword Search API**: query thông qua “match”, “bool”
- **Semantic Search API**: embedding truy vấn, tìm kiếm theo cosine similarity

KẾT QUẢ THỰC NGHIỆM

Thống kê số lượng thành công/lỗi của hệ thống thông qua Prometheus, Grafana



Kết quả sau 5 phút



Kết quả sau 10 phút

KẾT QUẢ THỰC NGHIỆM

Tiêu chí	Kết quả
Tốc độ	Tăng tốc độ thu thập dữ liệu lên nhiều lần so với phương pháp tuần tự.
Độ chính xác trong trích xuất	Độ chính xác trong việc trích xuất thông tin dao động mức 70-80% ở các lần chạy khác nhau
Độ chính xác của API tìm kiếm	Các API hoạt động ổn định, Semantic Search cho kết quả truy vấn khá khớp về mặt ngữ nghĩa

KẾT LUẬN

Quá trình nghiên cứu và triển khai, hệ thống đã đạt được các kết quả:

- Tự động thu thập và xử lý dữ liệu báo chí quy mô lớn
- Áp dụng AI để thay thế cho việc tự đặt các rule-based truyền thống
- Đảm bảo hiệu năng và khả năng mở rộng

Các điểm cần cải thiện:

- Tối ưu các bộ lọc đã có trong hệ thống
- Tăng cường độ chính xác của quá trình parser hỗ trợ bởi AI
- Cải thiện thêm về kết quả đầu ra của các API Search

TRÂN TRỌNG CẢM ƠN!