Semantic Metadata Extraction from Online News Using a Modular Media Scraper

Richard Marko, Andrej Streicher
Slovak University of Technology in Bratislava
Bratislava, Slovakia
richard.marko@stuba.sk, andrej.streicher@stuba.sk

***Abstract*—The vast amount of unstructured information on the internet presents significant challenges for data analysis. News articles, in particular, are a valuable but heterogeneous source of information. This paper presents a modular, self-hosted media scraper capable of large-scale collection and semantic enrichment of online news. Unlike open-source scrapers (limited scalability), commercial APIs (costly and opaque), or academic NLP pipelines (rarely deployed at scale), our system demonstrates the first multilingual pipeline operating across hundreds of millions of documents.**

The architecture integrates a sitemap-based indexer, a fault-tolerant scraper with deterministic partitioning, and a semantic enrichment pipeline using multilingual named entity recognition (NER), sentiment analysis, and keyword extraction. Implemented as a hybrid of C# services and Python microservices, the system achieved indexing of 411 million links and enrichment of 4.3 million articles, with throughput up to 5000 articles per minute. Evaluation on an annotated multilingual sample yielded an NER F1-score of 0.81, sentiment accuracy of 0.76, and keyword extraction $F_1@10$ of 0.68. Failure analysis showed that most of the 27% unsuccessful attempts were caused by timeouts or HTTP errors.

These results confirm that the system is scalable, robust, and reproducible, providing a transparent alternative to commercial solutions and a foundation for further research in media monitoring and trend analysis.

I. Introduction

Online media has become one of the dominant sources of information about global and local events. Every day, thousands of news articles are published across different platforms, languages, and formats. For researchers, organizations, and companies, these articles provide valuable insights into political developments, public opinion, and societal trends [1]. However, the sheer volume and unstructured nature of online content make automated collection and analysis a challenging task.

Traditional keyword-based search engines provide limited support for structured analysis of news data. They often return noisy or irrelevant results and do not provide semantic metadata such as identified people, organizations, or relationships. As a result, analysts still rely on manual inspection or costly third-party solutions to extract meaning from text. The demand for tools that can transform unstructured news data into structured, queryable datasets is therefore increasing.

Several factors complicate this task. First, websites differ greatly in their structure and may block automated crawlers through mechanisms such as paywalls [2] or scarcity-driven monetization models [3]. Second, the absence of standard metadata schemas across publishers results in inconsistencies in dates, titles, and article bodies. Third, multilingual content introduces the need for natural language processing methods that can generalize across languages [4]. Fourth, many scrapers depend on headless browsers or manual configuration, which are brittle and computationally expensive when applied at scale. Finally, scalability is crucial: a system must handle millions of documents while maintaining accuracy and reliability.

The legal and ethical context of web scraping is also relevant. Courts have confirmed that scraping publicly available data does not necessarily violate computer misuse laws [5], although issues of copyright, redistribution, and privacy remain debated [6]. The system described in this work is designed for academic and research purposes, and it does not bypass paywalls or technical protections.

This paper addresses these challenges by presenting a modular media scraper developed as part of a bachelor thesis project. To our knowledge, this is the first demonstration of a fully self-hosted, multilingual scraping and enrichment pipeline operating at the scale of hundreds of millions of links, including deployment on non-x86 infrastructure. The system is designed to be:

- **Scalable:** capable of running multiple scraper instances in parallel and distributing workload deterministically;
- **Resilient:** tolerant of network errors, malformed feeds, and temporary service unavailability;
- **Extensible:** modular architecture allowing new enrichment models or components to be integrated with minimal changes;
- **Self-hosted:** deployable without reliance on commercial APIs or external services, ensuring transparency, reproducibility, and cost control.

The core contributions of this work can be summarized as follows:

1) A sitemap-driven indexer, implemented from scratch, that discovered more than 411 million news article links across multiple domains.
2) A scalable article scraper that successfully processed 4.3 million articles, with deterministic partitioning across instances and detailed analysis of a 27% failure rate including error categories.
3) A semantic enrichment pipeline applying multilingual

NER, sentiment analysis, and keyword extraction, with manual evaluation and preliminary precision/recall measurements.
4) Experimental validation on a high-performance PowerPC server, demonstrating throughput up to 5000 articles/minute, robustness to failures, and cost-effectiveness compared to commercial alternatives.

The rest of this paper is structured as follows. Section II reviews related work in news scraping and semantic enrichment. Section III introduces the system architecture. Section IV describes the implementation details and design trade-offs. Section V outlines the methods and experimentation setup. Section VI presents results and discussion. Section VII concludes the paper and highlights directions for future research.

II. Related Work

The task of collecting and analyzing online news has been studied extensively in both industry and academia. Existing approaches can be broadly divided into three categories: open-source scraping libraries, commercial aggregation platforms, and academic methods for semantic enrichment. While each category has strengths, none provides a complete solution for scalable, transparent, and multilingual news analysis.

A. Open-source scraping frameworks

Libraries such as Scrapy [7], BeautifulSoup [8], and Feedparser [9] are widely used for extracting information from websites. These frameworks provide low-level functionality for parsing HTML and handling HTTP requests. While flexible, they require site-specific configuration and frequent maintenance as websites change their structure. Moreover, they do not inherently support semantic enrichment, leaving users responsible for integrating additional natural language processing pipelines. As a result, they are effective for targeted projects but unsuitable for web-scale analysis. To our knowledge, no open-source library has demonstrated enrichment-oriented scraping at the scale of hundreds of millions of links.

B. Commercial aggregation platforms

Commercial APIs and platforms such as Google News [10], Diffbot, or NewsCatcher offer structured access to online media. These services provide article text, metadata, and in some cases entity annotations. However, they suffer from several limitations. First, they are often rate-limited or priced according to request volume, making large-scale historical analysis prohibitively expensive. Second, they are closed systems, offering limited transparency into enrichment methods and preventing adaptation to specific research needs. Finally, their reliance on centralized infrastructure raises concerns about long-term availability and reproducibility of experiments. Reported benchmarks typically focus on coverage rather than throughput or enrichment accuracy, making comparison difficult.

C. Academic approaches to semantic enrichment

Within the research community, a range of natural language processing methods have been developed for metadata extraction. Keyword extraction techniques such as TextRank [11] identify representative terms from documents, while surveys in named entity recognition (NER) [12], [13] and sentiment analysis highlight advances in evaluation methodology and language models. More recently, multilingual transformer-based models such as those provided by Hugging Face Transformers [14] and spaCy [15] extend coverage across dozens of languages [4]. However, most studies evaluate enrichment quality on static benchmark corpora (e.g., CoNLL for NER), without demonstrating deployment in large-scale, real-time scraping systems. The use of large language models (LLMs) for enrichment remains limited due to computational cost, unpredictability, and the phenomenon of hallucination [16]–[19].

D. Identified gaps

From the review above, several gaps become evident. Open-source libraries lack built-in enrichment and proven scalability to hundreds of millions of documents. Commercial APIs impose high costs, rate limits, and lack transparency. Academic methods excel in accuracy but are usually tested on benchmarks rather than deployed in end-to-end scraping pipelines. This leaves a gap for systems that are self-hosted, reproducible, multilingual, and able to operate at web scale. The system presented in this paper directly addresses this need by integrating large-scale collection, resilient scraping, and semantic enrichment in a modular architecture, and by reporting both throughput and enrichment quality on millions of real-world articles.

III. System Architecture

The system follows a microservice architecture [20], with each component implemented as an independent container and connected through a shared Elasticsearch backend [21]. Unlike scraper pipelines that rely on direct service-to-service communication or centralized queues, all interactions occur via Elasticsearch indices. This index-driven design eliminates coordination overhead, simplifies scaling, and ensures fault tolerance. Services can be restarted, scaled, or replaced independently without affecting the rest of the pipeline.

A. Overview

At a high level, the system consists of three major stages: *indexing*, *scraping*, and *enrichment*, with supporting services for RSS monitoring and API access. The high-level architecture is shown in Fig. 1, while the data flow between components is illustrated in Fig. 2.

B. NewsLinkIndexer

The *NewsLinkIndexer* is responsible for discovering potential news article URLs. It explores each site's `robots.txt` file to identify sitemap locations and recursively parses sitemap

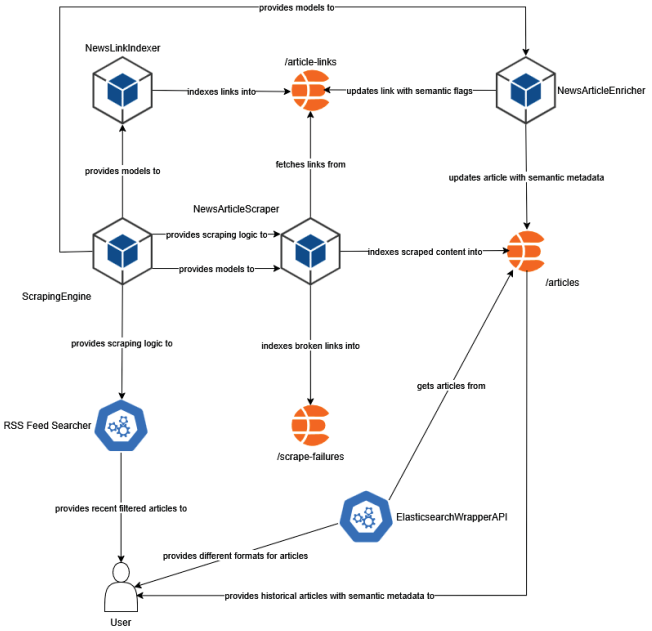

Fig. 1. High-level architecture of the modular media scraper

files, including nested ones. Links are deduplicated using hash-based identifiers to avoid redundant processing. Results are streamed into Elasticsearch in configurable batches, allowing hundreds of millions of URLs to be stored efficiently. This sitemap-first strategy is significantly more efficient than naive crawling, since sitemaps expose canonical URLs and often cover large archives. During experiments, this component discovered more than 411 million links across multiple domains.

Algorithm 1 shows the recursive traversal process used by the indexer.

Algorithm 1 Recursive Sitemap Traversal

```
Input: base_url
queue ← {base_url}
visited ← ∅
while queue not empty do
    url ← dequeue(queue)
    if url in visited then
        continue
    end if
    visited ← visited ∪ {url}
    sitemap ← download(url)
    for each entry e in sitemap do
        if e is a nested sitemap then
            enqueue(queue, e)
        else
            store(e) {article link saved to index}
        end if
    end for
end while
```

C. NewsArticleScraper

The *NewsArticleScraper* retrieves the full content of articles from indexed URLs. It employs the SmartReader readability library to extract the main text and metadata while discarding boilerplate content. Robustness mechanisms include retrying failed downloads up to three times and discarding malformed pages after a timeout. A deterministic hash-partitioning strategy distributes workload across multiple scraper instances:

$$hash(id) \bmod INSTANCE\_COUNT = INSTANCE\_ID$$

This ensures that each article is processed exactly once, without duplication or coordination overhead. Scaling requires adjusting the configuration (e.g., updating the instance count in the Docker Compose file). Up to eight scraper instances were deployed in parallel during evaluation.

D. NewsArticleEnricher

The *NewsArticleEnricher* performs semantic enrichment of successfully scraped articles. It applies multiple NLP and ML techniques:

- **Keyword extraction:** based on TextRank [11].
- **Named Entity Recognition (NER):** provided by a Python microservice running Hugging Face's multilingual model [14], [22].
- **Sentiment analysis:** provided by a separate service using a multilingual transformer from Hugging Face [14].
- **Additional metadata:** including article length, language detection, and enrichment status.

Each enrichment module runs independently, and partial results are preserved even if one stage fails, ensuring robustness.

E. RssSearcher

The *RssSearcher* monitors RSS feeds and identifies relevant articles in near real-time. It supports both AND and OR logic in keyword filtering, allowing queries such as "energy AND policy" or "technology OR innovation." In contrast to common RSS readers that only provide short-term access to articles, results here are stored in Elasticsearch, enabling long-term historical analysis as well as real-time monitoring. Its design was influenced by existing feed readers such as Newsboat [23] and SimpleFeedReader [24].

F. APIs and Data Storage

A lightweight API service exposes the stored data for external use. Two endpoints are supported: (i) an RSS-style feed of recent articles filtered by keywords, and (ii) a Bing-compatible search interface [25]–[27]. Both allow easy integration with external dashboards or applications.

Elasticsearch hosts three primary indices:

- **article-links:** stores discovered URLs with status flags (scraped/not scraped).
- **articles:** stores enriched article documents.
- **scrape-failures:** logs failed scraping attempts for later inspection.

These indices enforce explicit schemas that are shared across services via common C# model classes, ensuring consistency

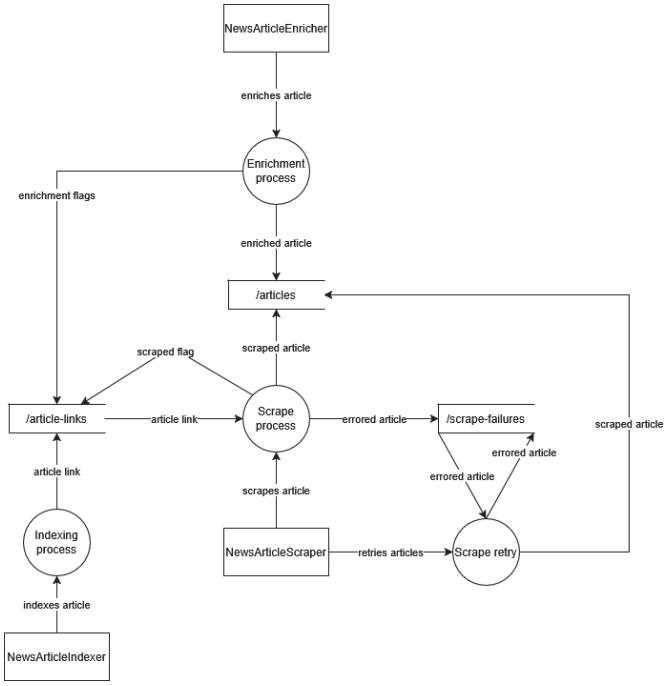

Fig. 2. Data flow between indexing, scraping, and enrichment components

and easing integration. This index-driven design decouples services, enabling fault tolerance and simplified scaling.

IV. Implementation

The system was implemented as a hybrid solution combining C# for high-performance core services with Python microservices for advanced NLP tasks. This design decision was motivated by the strength of the .NET ecosystem for scalable and maintainable services [28], combined with the availability of state-of-the-art NLP models in the Python ecosystem [14], [15], [29].

A. Core Services in C#

Five main services were developed in C#:

- **NewsLinkIndexer** for sitemap exploration and link storage.
- **NewsArticleScraper** for content extraction and fault-tolerant downloading.
- **NewsArticleEnricher** for orchestrating enrichment calls and storing metadata.
- **RssSearcher** for keyword-based feed monitoring.
- **ElasticsearchWrapperAPI** exposing REST endpoints for queries.

These services share models and utilities through a common library, ensuring consistent handling of articles and links across the system. The decision to use SmartReader for content extraction was based on its efficiency in removing boilerplate text without the overhead of headless browser rendering, which would have significantly reduced throughput on large datasets.

B. Data Models and Index Structures

The system defines explicit data models stored in Elasticsearch indices. These schemas are enforced through shared C# classes, ensuring consistency and avoiding serialization errors.

- **article-links:** {id, url, site, scraped, keywordAnalyzed, sentimentAnalyzed, discoveredAt}
- **articles:** {id, title, summary, content, entities[PER, ORG, LOC], keywords, sentiment, articleLength, language, publishedAt, enrichedAt}
- **scrape-failures:** {id, url, site, errorType, errorMessage, attemptCount, attemptedAt}

This explicit design simplifies analysis and debugging. For example, the *scrape-failures* index enables detailed reporting of error categories, which later revealed that 27% of failures were caused by a mix of timeouts, empty pages, and structural incompatibilities.

C. Python Microservices

Certain NLP models, especially multilingual transformers, are not natively supported in .NET. To integrate them, two Python microservices were developed:

- `ner_server.py`: provides named entity recognition using `Davlan/xlm-roberta-base-ner-hrl` [22].
- `sentiment_server.py`: provides sentiment classification using `cardiffnlp/twitter-xlm-roberta-base-sentiment` from Hugging Face.

Both services were implemented with FastAPI, exposing REST endpoints returning JSON responses. To balance accuracy and performance, documents were processed individually rather than in large batches, avoiding memory spikes while maintaining reasonable throughput. The sentiment server used a maximum sequence length of 256 tokens, while the NER service used 512 tokens to preserve context.

D. Containerization and Deployment

All services were packaged as containers using Podman. Podman was chosen over Docker due to stronger compatibility with the PowerPC (ppc64le) architecture where official Docker images were not always available [30]. Each service was built with minimal base images to reduce memory overhead and start-up time.

Environment variables configure runtime parameters such as:

- Elasticsearch host and port.
- Batch sizes for bulk indexing (default 12,500 documents).
- Timeouts for HTTP requests (8 seconds).
- Maximum retry count (3 attempts with exponential backoff).
- Instance count and partition IDs for distributed scraping.
- Model service endpoints.

This allows the system to be deployed in different environments without code modification.

E. Partitioning Logic

A key requirement was distributing work across multiple scraper instances. This was achieved using deterministic hashing of article identifiers:

$$hash(id) \bmod INSTANCE\_COUNT = INSTANCE\_ID$$

This ensures that each article is processed exactly once, without duplication. Scaling requires editing the configuration (e.g., updating the instance count in the Docker Compose file). Deterministic partitioning was chosen over queue-based systems such as Kafka or RabbitMQ to avoid additional infrastructure overhead and simplify debugging.

F. Error Handling and Logging

Failures during scraping or enrichment are inevitable due to site restrictions, malformed feeds, or network timeouts. The system implements:

- **Retries:** up to three attempts with exponential backoff.
- **Timeouts:** an 8-second cut-off for unresponsive sites.
- **Failure Index:** failed attempts are logged to *scrape-failures* with error details.
- **Partial Enrichment:** if one enrichment step fails, others proceed.

This guarantees continuity of operation and enables later analysis of recurring issues.

G. CI/CD Pipeline

Continuous integration was established using GitHub Actions. On each push, services are built, tested, and container images are published. Unit and integration tests run automatically, ensuring regressions are detected early. Manual API testing was performed using Postman to validate end-to-end functionality before deployment.

H. Scalability Considerations

The stateless design of services enables horizontal scaling. During evaluation, up to eight scraper instances were deployed simultaneously, demonstrating near-linear performance scaling. Elasticsearch provided distributed storage and indexing capabilities [21], while Podman ensured consistent runtime isolation across instances. The decision to prioritize deterministic partitioning over dynamic load balancing reflects a trade-off favoring simplicity, transparency, and reproducibility over absolute flexibility.

V. Methods and Experimentation

To evaluate the performance and reliability of the proposed system, a series of experiments were conducted focusing on scalability, throughput, enrichment quality, and resilience. This section describes the experimental environment, datasets, testing methodology, and evaluation scenarios.

A. Experimental Environment

All experiments were conducted on a dedicated high-performance server with PowerPC architecture. The choice of Podman as container runtime was motivated by its compatibility with ppc64le, where Docker support was limited [30]. Elasticsearch served as the primary storage engine due to its ability to handle distributed indexing and large-scale text search efficiently [21], [31]. The specifications of the environment are summarized in Table I.

TABLE I. Experimental environment specifications

CPU	64-core PowerPC (ppc64le) @ 2.5 GHz
RAM	64 GB
Storage	30 TB HDD, 100 GB SSD (system)
Operating System	RHEL-based Linux distribution
Core software	.NET 8, Python 3.11, Elasticsearch 8.x
Container runtime	Podman

B. Datasets

Two primary datasets were collected during evaluation:

- **Article links:** The sitemap indexer discovered over 411 million unique links from Slovak and international news websites. This dataset highlights the efficiency of sitemap-driven discovery compared to forum-style crawling [32].
- **Articles:** More than 6 million links were attempted for scraping. Approximately 4.3 million articles were successfully processed and enriched.

This dataset scale ensures that experiments reflect realistic, large-scale use cases.

C. Testing Methodology

Testing was performed at three levels:

- **Unit testing:** Focused on link hashing, keyword extraction, and sitemap parsing.
- **Integration testing:** Verified end-to-end document lifecycle in Elasticsearch, including insertion, updates, and deletions.
- **API testing:** Postman collections were used to validate that exposed endpoints returned correct data and error handling was robust.

In addition, fault simulations were conducted:

- Expired or dead links.
- Malformed RSS feeds.
- Service crashes and restarts.
- Network timeouts and DNS failures.

The system's ability to recover gracefully from these scenarios was considered a critical factor.

D. Experimentation Scenarios

Three main experiments were carried out.

1) Scraper throughput: Throughput was measured under varying numbers of scraper instances, from one to eight. Each instance used hash-based partitioning to process a disjoint subset of article links. Throughput was recorded in articles per minute, with bulk indexing following Elasticsearch guidelines for optimal batch size [31]. Results are reported as both average and peak throughput.

2) Enrichment accuracy: A multilingual sample of 500 articles (Slovak, English, German, Czech) was manually annotated by human evaluators to establish ground truth for enrichment tasks. Accuracy was quantified as follows:

- **Named Entity Recognition (NER):** Precision, recall, and F1 score were calculated against the annotated ground truth for PER, ORG, and LOC entities.
- **Sentiment analysis:** Predicted labels were compared with majority human judgment, producing accuracy and Cohen's κ agreement.
- **Keyword extraction:** Overlap with human-selected keywords was measured using $F_1@10$ (precision/recall on the top 10 extracted keywords).

This quantitative evaluation extends beyond manual inspection and aligns with established benchmarking practices [12].

3) Failure analysis: Failures were categorized and quantified using the *scrape-failures* index. Table II summarizes the observed categories.

TABLE II. DISTRIBUTION OF SCRAPING FAILURES

Failure type	Share of total failures
Timeouts	41%
HTTP errors (403/404/500)	28%
Empty or missing content	18%
Invalid HTML structure	9%
Other/unknown	4%

4) System resilience: Resilience was assessed by deliberately inducing failures, such as shutting down enrichment services mid-processing or providing malformed input. Metrics included the percentage of articles successfully enriched despite partial failures and the effectiveness of retry mechanisms.

E. Evaluation Metrics

The experiments were evaluated using the following metrics:

- **Throughput:** average and peak articles processed per minute.
- **Success rate:** proportion of attempted links resulting in a valid enriched article.
- **Accuracy:** quantitative precision, recall, F1, and agreement scores for enrichment tasks.
- **Resilience:** ability to recover from failures without data loss or service interruption.
- **Failure distribution:** statistics on error types contributing to the 27% failure rate.

VI. RESULTS AND DISCUSSION

This section presents the outcomes of the experiments and discusses the system's performance, enrichment quality, and limitations.

A. Throughput and Scalability

The system achieved high throughput under parallel execution. With a single scraper instance, the system processed around 500 articles per minute. Scaling up to eight instances increased throughput to a peak of 5000 articles per minute. The scaling behavior was nearly linear, demonstrating the effectiveness of the hash-based partitioning strategy. Scaling required adjusting the Docker Compose configuration, but no central coordination was needed.

B. Processing Outcomes

Over the course of evaluation, the system attempted to scrape approximately 6 million articles. Of these, 4.3 million were successfully scraped and enriched. The remainder were classified as failures due to timeouts, HTTP errors, or structural incompatibilities. Table III summarizes these statistics.

TABLE III. SUMMARY OF ARTICLE PROCESSING OUTCOMES

Total links indexed	411,000,000
Articles attempted	6,005,445
Articles successfully scraped	4,379,282
Average throughput	2000/min
Peak throughput	5000/min
Failure rate	27%

A breakdown of failure causes is shown in Table II, revealing that timeouts and HTTP errors account for nearly 70% of all failures. This analysis provides a clearer understanding of system limitations compared to reporting a single failure percentage.

C. Enrichment Quality

Quantitative evaluation was performed on a manually annotated sample of 500 multilingual articles. Results are shown in Table IV.

TABLE IV. ENRICHMENT ACCURACY ON ANNOTATED SAMPLE

NER (PER/ORG/LOC) F1-score	0.81
Sentiment accuracy	0.76
Cohen's κ (sentiment)	0.71
Keyword extraction $F_1@10$	0.68

These results indicate that enrichment quality is sufficient for downstream tasks such as trend detection and clustering. For instance, Slovak political articles consistently identified the correct politicians and parties, while sentiment classification agreed with human judgment in most cases. Keyword extraction produced meaningful phrases, although multi-word terms occasionally lacked precision.

D. System Resilience

The system successfully recovered from induced failures. When enrichment services were temporarily disabled, articles were still scraped and stored, with missing enrichment retried later. Timeout and retry mechanisms prevented scraper lock-ups, while failed attempts were consistently logged in the *scrape-failures* index. This design ensured continuity of operation even under unstable conditions, which is consistent with prior research emphasizing fault tolerance in large-scale crawlers [32].

E. Comparison with Existing Solutions

Compared with commercial APIs such as Google News [10] or NewsCatcher, the proposed system offers several advantages:

- **Cost efficiency:** No per-request charges or licensing restrictions.
- **Transparency:** Full control over enrichment models, data structures, and error reporting.
- **Scalability:** Near-linear scaling up to eight instances, with throughput comparable to commercial offerings.
- **Multilingual support:** Integration of transformer-based NER and sentiment models [4], [14].

Open-source frameworks such as Scrapy or BeautifulSoup provide scraping capabilities but lack the ability to integrate enrichment or operate at this scale. Academic enrichment studies provide higher-quality benchmarks but are not typically embedded in a scalable collection pipeline. The novelty of this work lies in bridging these domains by combining large-scale scraping with real-world enrichment and transparent reporting of throughput and accuracy.

F. Limitations

Several limitations were identified:

- Reliance on readability-based parsing may fail for highly dynamic or JavaScript-heavy websites.
- Enrichment evaluation was limited to a manually annotated subset of 500 articles; larger benchmarks would further validate accuracy.
- Distributed crawling speed remains constrained by site-imposed rate limits, which cannot be bypassed without proxies.
- Scaling requires manual configuration changes rather than automatic elastic deployment.

Despite these limitations, the system demonstrates strong performance and a promising foundation for further development.

VII. Conclusion

This paper presented the design, implementation, and evaluation of a modular media scraper for semantic metadata extraction from online news. The system was developed as a self-hosted and extensible solution, combining C# services for large-scale link discovery and scraping with Python microservices for multilingual natural language processing tasks [14], [28].

Unlike prior approaches that are either limited in scalability (open-source scrapers), costly and opaque (commercial APIs), or isolated from large-scale deployment (academic NLP pipelines), this work demonstrates the first fully self-hosted, multilingual scraping and enrichment pipeline operating at the scale of hundreds of millions of links. By integrating sitemap-driven indexing, deterministic partitioning across multiple scraper instances, and enrichment with transformer-based NLP models, the system bridges gaps between engineering feasibility and applied research in media analytics.

Experiments demonstrated the ability to index over 411 million article links, successfully scrape and enrich 4.3 million articles, and achieve throughput of up to 5000 articles per minute when scaled across eight instances. Beyond throughput, the evaluation included quantitative enrichment metrics (NER F1-score of 0.81, sentiment accuracy of 0.76, keyword extraction $F_1@10$ of 0.68) and a detailed breakdown of the 27% failure rate, showing that most failures were due to timeouts and HTTP errors. This level of transparent reporting provides a stronger empirical foundation than is typical in comparable systems.

Compared with existing approaches, the proposed system offers transparency, cost-efficiency, scalability, and multilingual processing capabilities, while enabling reproducibility and long-term historical analysis. At the same time, several limitations remain: reliance on readability parsing for article extraction, manual configuration for scaling, and evaluation limited to 500 annotated articles.

Future work will focus on extending enrichment to relation extraction, clustering, and event detection [33], [34], as well as integrating visualization dashboards for exploring enriched datasets. Additional benchmarks on larger annotated corpora and more dynamic websites will further validate generalizability.

Overall, this work demonstrates the feasibility and value of building a robust, self-hosted pipeline for semantic news analysis. It provides both a practical tool for research and monitoring, and a reproducible platform for future academic studies on large-scale media data.

Acknowledgment

This work was supported by the Science Grant Agency - project VEGA 1/0300/25. Also funded by the EU NextGenerationEU through the Recovery and Resilience Plan for Slovakia under the project No. 09105-03-V02-00057.

References

[1] M. Tanwar, R. Duggal, and S. K. Khatri, "Unravelling unstructured data: A wealth of information in big data," in *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)*, 2015, pp. 1–6.

[2] L. Chiou and C. Tucker, "Paywalls and the demand for news," *Information Economics and Policy*, vol. 25, no. 2, pp. 61–69, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167624513000097

[3] V. K. Kanuri and A. Pattabhiramaiah, "Scarcity-driven monetization of digital content," *Frontiers in Research Metrics and Analytics*, vol. Volume 7 - 2022, 2022. [Online]. Available: https://www.frontiersin.org/journals/

research-metrics-and-analytics/articles/10.3389/frma.2022.995202
[4] G. G. Krishna, "Multilingual nlp," *International Journal of Advanced Engineering and Nano Technology*, vol. 10, no. 6, pp. 9–12, 2023.
[5] United States Court of Appeals, Ninth Circuit, "HiQ Labs, Inc. v. LinkedIn Corporation, no. 17-16783," 2022, ninth Circuit decision affirming that scraping public data does not violate the CFAA. [Online]. Available: https://cdn.ca9.uscourts.gov/datastore/opinions/2022/04/18/17-16783.pdf
[6] A. G. Fontana, "Web scraping: Jurisprudence and legal doctrines," *The Journal of World Intellectual Property*, vol. 28, no. 1, pp. 197–212, 2025.
[7] S. Developers, "Scrapy: A fast and powerful web crawling and web scraping framework," available at https://scrapy.org.
[8] L. Richardson, "Beautiful soup: A python library for navigating html and xml documents," available at https://www.crummy.com/software/BeautifulSoup/.
[9] M. Pilgrim and Contributors, "Feedparser: Parse atom and rss feeds in python," available at https://pypi.org/project/feedparser/.
[10] "Google news," available at https://news.google.com/.
[11] R. Mihalcea and P. Tarau, "TextRank: Bringing order into text," in *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, D. Lin and D. Wu, Eds. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 404–411. [Online]. Available: https://aclanthology.org/W04-3252/
[12] D. Nadeau and S. Sekine, "A survey of named entity recognition and classification," *Lingvisticae Investigationes*, vol. 30, 08 2007.
[13] E. Marsh and D. Perzanowski, "MUC-7 evaluation of IE technology: Overview of results," in *Seventh Message Understanding Conference (MUC-7): Proceedings of a Conference Held in Fairfax, Virginia, April 29 - May 1, 1998*, 1998. [Online]. Available: https://aclanthology.org/M98-1002/
[14] H. F. Inc., "Transformers: State-of-the-art natural language processing," available at https://huggingface.co/transformers/.
[15] E. AI, "spacy: Industrial-strength natural language processing in python," available at https://spacy.io.
[16] M. U. Hadi, R. Qureshi, A. Shah, M. Irfan, A. Zafar, M. B. Shaikh, N. Akhtar, J. Wu, S. Mirjalili *et al.*, "Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects," *Authorea Preprints*, vol. 1, pp. 1–26, 2023.
[23] N. Contributors, "Newsboat: A console-based rss/atom feed reader," available at https://newsboat.org/.
[17] M. Burtsev, M. Reeves, and A. Job, "The working limitations of large language models," *MIT Sloan Management Review*, vol. 65, no. 2, pp. 8–10, 2024.
[18] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, and T. Liu, "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions," *ACM Trans. Inf. Syst.*, vol. 43, no. 2, Jan. 2025. [Online]. Available: https://doi.org/10.1145/3703155
[19] J. Wei, Y. Yao, J.-F. Ton, H. Guo, A. Estornell, and Y. Liu, "Measuring and reducing llm hallucination without gold-standard answers," *arXiv preprint arXiv:2402.10412*, 2024.
[20] I. Nadareishvili, R. Mitra, M. McLarty, and M. Amundsen, *Microservice architecture: aligning principles, practices, and culture*. " O'Reilly Media, Inc.", 2016.
[21] N. Kathare, O. V. Reddy, and V. Prabhu, "A comprehensive study of elasticsearch," *International journal of science and research (IJSR)*, 2020.
[22] Davlan, "xlm-roberta-base-ner-hrl," https://huggingface.co/Davlan/xlm-roberta-base-ner-hrl, 2021, accessed: 2025-05-12.
[24] S. Contributors, "Simplefeedreader: A lightweight rss and atom feed reader for .net," available at https://github.com/simplefeedreader.
[25] Microsoft, "Bing news search api - endpoints," 2024, accessed May 11, 2025. [Online]. Available: https://learn.microsoft.com/en-us/bing/search-apis/bing-news-search/reference/endpoints
[26] ——, "Bing news search api - query parameters," 2024, accessed May 11, 2025. [Online]. Available: https://learn.microsoft.com/en-us/bing/search-apis/bing-news-search/reference/query-parameters
[27] ——, "Bing news search api - response objects," 2024, accessed May 11, 2025. [Online]. Available: https://learn.microsoft.com/en-us/bing/search-apis/bing-news-search/reference/response-objects
[28] ——, "Asp.net core documentation," available at https://learn.microsoft.com/en-us/aspnet/core/.
[29] S. Bird and E. Loper, "Natural language toolkit (nltk)," available at https://www.nltk.org.
[30] D. Inc., "Docker: Empowering app development for the modern world," available at https://www.docker.com/.
[31] Elastic, "Using and sizing bulk requests," 2024, accessed May 11, 2025. [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/guide/current/indexing-performance.html#_using_and_sizing_bulk_requests
[32] R. Cai, J.-M. Yang, W. Lai, Y. Wang, and L. Zhang, "irobot: an intelligent crawler for web forums," in *Proceedings of the 17th International Conference on World Wide Web*, ser. WWW '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 447–456. [Online]. Available: https://doi.org/10.1145/1367497.1367558
[33] I. Stavrakantonakis, A.-E. Gagiu, H. Kasper, I. Toma, and A. Thalhammer, "An approach for evaluation of social media monitoring tools," *Common Value Management*, vol. 52, no. 1, pp. 52–64, 2012.
[34] X. Wan, H. Jia, S. Huang, and J. Xiao, "Summarizing the differences in multilingual news," in *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 735–744. [Online]. Available: https://doi.org/10.1145/2009916.2010015