FeedFetcher: A Resilient Web Feed Downloader for Corpus Construction

Ondřej Herman[1,2], **Jan Kraus**[1], **Vít Suchomel**[1,2]
[1]Lexical Computing, Brno, Czech Republic
[2]Natural Language Processing Centre, Masaryk University, Brno, Czech Republic
firstname.lastname@sketchengine.eu

Abstract

Building large-scale, timestamped monitor corpora requires robust and efficient tools for continuous web data acquisition. We present FeedFetcher, an open-source, lightweight yet resilient downloader designed to collect linguistic data from RSS/Atom web feeds. The tool enables continuous corpus updates by harvesting newly published web content with minimal downtime and high data integrity. Implemented in Rust for performance, memory safety, and scalable concurrency, FeedFetcher supports thousands of simultaneous connections while maintaining server politeness. The software is available under the GPL-3.0 license on https://github.com/ondra/feed_fetcher. In our setup, the entire workflow integrates FeedFetcher with downstream text-processing pipelines for tokenization, lemmatization, corpus compilation and deployment. The system is currently used to update monitor corpora in 64 languages, producing approximately two billion tokens per month. These corpora are available in Sketch Engine. We also describe methods for discovering new web feeds, combining manual exploration with automated extraction from large-scale web crawls to expand linguistic coverage. We demonstrate the system's applicability through a time-based analysis of word-frequency change, showing how long-term accumulation of timestamped data supports the study of lexical dynamics and language evolution.

Keywords: Web Feed, Timestamped Data, Monitor Corpus, Web Corpus, Sketch Engine

1. Introduction

To build large-scale, time-stamped monitor corpora, a robust and efficient tool for downloading content from the web is essential. We have developed FeedFetcher, a specialized downloader designed to gather linguistic data from web feeds.

1.1. Web Feeds: RSS, Atom, JSON Feed

A web feed is a data format used by websites to publish frequently updated content, such as news articles or blog posts. The predominant standards are RSS (Really Simple Syndication) and Atom, both of which are XML-based. While RSS is the older and more established format, its specification can be ambiguous, particularly regarding aspects like character escaping and date formatting. Atom was developed later to address these inconsistencies. In practice, many websites provide both, and for the purpose of content discovery, they are largely interchangeable. Another, JSON-based, format is described by the JSON Feed standard, but its usage is not common in practice.

A feed consists of a list of entries, each corresponding to a specific web page. These entries typically include the page's URL, a title, publication and update timestamps, and often a brief summary. Our system leverages these feeds as a starting point for discovering and downloading full-text articles for corpus inclusion.

1.2. Advantages and Limitations of Feed-Based Corpus Construction

Feed-based corpus construction offers several advantages. Because RSS and Atom feeds are updated regularly and contain publication date, they allow continuous collection of recent text from trusted sources without crawling the entire web. Vertical crawling based on selected feeds yields more uniform and higher-quality corpora, reduces the amount of unusable pages and duplicates, and eases metadata extraction (publication date, author, title). The dynamic nature of feeds allows researchers to build monitor corpora that capture linguistic change, track neologisms and update lexica in real time.

However, reliance on RSS/Atom feeds also imposes limitations. The available feeds cover only specific domains – typically news or blogs – so corpora built from them may have limited genre diversity. Many feeds provide only headlines and short summaries; the full article must still be downloaded and parsed, and content extraction remains challenging. Large-scale feed aggregation requires infrastructure to de-duplicate articles and manage high volumes of data.

Another limitation of this approach to compiling monitor corpora lies in the reliance on timestamps provided in web feeds. We assume that the publishing dates reported by feed metadata accurately reflect the actual time of publication of the feed content, which is likely true in the vast majority of cases. However, we have not systematically evaluated the

Proceedings of the Fifteenth Language Resources and Evaluation Conference (LREC 2026), pages 7014–7022
11–16 May 2026.

reliability of these timestamps, and occasional inaccuracies or inconsistencies may affect the temporal precision of the resulting corpora.

Finally, because feeds can change or disappear, long-term reproducibility may be difficult. Despite these challenges, the literature shows that RSS/Atom feeds are a valuable tool for constructing dynamic, domain-specific corpora and for monitoring linguistic trends.

2. Related Work

This section provides an overview of different existing projects with comparable objectives.

In early work, Fairon (2006) presented Corporator, a command-line program that takes RSS news feeds as input, downloads all referenced web pages, filters HTML content and advertisements, removes menus and boilerplate, and converts text to UTF-8. The paper argues that RSS feeds provide high-quality, regularly updated links from a small number of carefully selected sites, enabling *vertical crawling* rather than indiscriminate web crawling. Corporator schedules regular downloads so the corpus grows over time. It supports multiple languages and integrates with Unitex for concordance searches.

Later work expanded into the creation of interface GlossaNet 2 (Fairon et al., 2008), a free online service where users define a dynamic corpus by selecting RSS feeds from a predefined pool (or by adding their own).

Alzahrani (2013) constructed the ArNeCo corpus by exploiting Google News RSS feeds. The feeds provided article titles and links that were crawled to retrieve full texts. The resulting corpus contains about eleven thousand Arabic news articles and over six million words across six domains; metadata such as publication date, author and section are preserved. The authors argue that RSS feeds allow the collection of contemporary text, helping to “address the recency problem” of older Arabic corpora like Gigaword.

Building on this idea, Al-Thubaity and Alhoshan (2019) introduced ARARSS, an open-source system that automatically constructs and updates Arabic corpora from RSS feeds. Users supply feed URLs and criteria such as target language and topic; the system downloads linked pages, removes duplicates, extracts metadata (location, time and topic) and stores the texts in a database. At publication, ARARSS had built a corpus containing over 28 million words. The authors note that feed-based construction reduces the number of unusable pages compared with broad crawling and yields a better balanced corpus.

Other researchers have created large-scale news corpora by aggregating thousands of feeds. The IJS NewsFeed pipeline by Novak and Trampuš (2012) periodically crawls a list of RSS feeds and a subset of Google News, extracts article links and downloads the full HTML pages. The pipeline cleans HTML into clear text, performs language identification, and semantically enriches the articles using the Enrycher system. Cleaned and enriched versions of articles are cached and distributed via a simple API. This feed-driven approach processes tens of thousands of articles per day and serves as the backbone of the JSI NewsFeed corpus by Krek et al. (2017), which processes roughly 80,000 feeds and yields 350–600 thousand deduplicated news articles per day; the English portion alone contains tens of billions of words.

Oxford University Press’s New Monitor Corpus (Digiteum, 2017) uses a similar strategy; the earlier system harvested 26 million news articles (approx. nine million tokens) via RSS feeds, and the subsequent "Super Corpus Platform" processes 2 million documents per month using Event Registry news aggregator and Azure cloud services.

Logoscope (Falk et al., 2018) is a semi-automatic tool for detecting and documenting French neologisms. It retrieves articles from multiple French newspapers via RSS feeds every day and processes them to keep only journalistic content; unknown word candidates are ranked using exclusion lists and machine-learning features. The platform monitors an average of 430 sources per day.

NeoCrawler by Kerremans et al. (2012), designed to study the life cycle of neologisms, also uses selected RSS and Atom feeds to compile dynamic corpora. A Manager component retrieves feeds at regular intervals, strips boilerplate and duplicates, assembles the corpus, performs tokenisation and tagging and returns the results to the user.

Bulgarian Neologism Detection system by Stoyanova et al. (2016) monitors top-news RSS feeds and downloads linked pages; metadata such as author and publication date are extracted and stored alongside the text before linguistic annotation.

NeoN by Tomaszewska et al. (2025) processes daily RSS feed data documents to detect Polish neologisms. The system employs multi-layer filtering, context-aware lemmatisation and an LLM-based precision-boosting in the final stage of candidate filtering while allowing researchers review candidate words through a web interface.

Tools originally developed as web crawlers have integrated RSS retrieval modules to track updates from news sites. News-please by Hamborg et al. (2017) is a generic open-source news site-harvester and extractor. To find articles from a given news outlet, it supports four crawling modes: analysing RSS feeds for recent articles, recursively

following internal links, parsing sitemaps and a combined "automatic" mode. Users can run two instances in parallel: an automatic mode to retrieve all historical articles and an RSS mode to ingest newly published articles. The extraction phase combines several state-of-the-art libraries (Newspaper, readability, etc.) to obtain title, lead, main text, publication date, author and main image; the results are stored in JSON or Elasticsearch.

3. FeedFetcher: A Resilient Web Feed Downloader for Corpus Construction

This section details its design principles, architecture, and implementation.

3.1. Design Goals

We set out to create FeedFetcher to help us create and maintain continuously updated corpora from Web texts.

Simplicity and reliability were our main aims, as the crawler needs to provide data for long periods of time with minimal downtime.

3.2. Architectural Design

FeedFetcher is designed as a batch-oriented system rather than a continuously running daemon. This design choice facilitates monitoring, simplifies error recovery, and allows for predictable resource allocation.

The process is divided into two distinct stages: an *update* stage and a *fetch* stage, which are executed periodically.

1. **Update Stage:** The cycle begins with a user-provided list of seed feed URLs. The *update* program downloads these feeds, parses them, and adds any previously unseen article URLs to a central download plan.

2. **Fetch Stage:** Subsequently, the *fetch* program reads this plan and downloads the full HTML content of the target pages.

The separation into the two stages made development significantly easier compared to a monolithic architecture. Each of the stages is self-contained, and can be debugged in isolation.

The entire batch process operates under a configurable timeout (e.g., one hour). If the process exceeds this limit, it terminates gracefully, persisting its current state. This mechanism prevents the crawler from getting stuck in an unresponsive state, a common issue with long-running network processes.

3.2.1. Robustness and Atomicity

To enhance robustness, each run begins by moving the existing plan file to a backup location. This ensures that a failed run does not corrupt the state and prevents concurrent executions. Similarly, output data is written to a temporary file and renamed to its final destination only upon successful completion, guaranteeing atomic updates. In case of power failure or other unexpected system downtime, running *update* or *fetch* is always safe after recovery.

The atomic-write mechanism ensures that the corpus compilation process, which follows the fetcher, never ingests a partially written or corrupt plan. This guarantees that each corpus update is clean and internally consistent. In case of failure, the fetcher can be safely restarted from an earlier plan version.

Failed downloads are automatically retried in subsequent runs, up to a configurable limit (typically three attempts), to handle transient network errors or temporary server unavailability.

3.2.2. Parallelism and Server Politeness

The fetcher is highly concurrent, capable of downloading from a large number of servers simultaneously (a configurable parameter; we have experimented with values up to 8,000 without notable issues). However, to avoid overloading any single server, all requests to the same host are strictly serialized and rate-limited with a configurable delay. Requests are pipelined over a single connection where supported by the server to improve efficiency.

The download order for pages from a single website is randomized in each run. This strategy ensures a diverse selection of content is retrieved when dealing with high-volume sources (e.g., aggregated blog platforms) that might otherwise not be fully processed within a single batch timeout.

3.2.3. Simplicity

FeedFetcher is designed to be simple to use. Both of the *update* and *fetch* programs use only a small amount of parameters, with sane and tested defaults, which should not be necessary to change for the common use cases.

The complete state is stored inside a single, human-readable[1], plan file. The list of feeds is stored in a single text file, with one URL per line, which can be modified at any time. This allows new

[1]We are deeply convinced that easily readable data is essential for effective operation, enabling quick understanding of ongoing processes and rapid inspection of any unexpected behaviour.

feeds to be added or feeds obsolete for any reason to be removed whenever needed.

FeedFetcher only uses standard Rust dependencies, so it is simple to compile and run it on many platforms supported by Rust.

3.3. The Download Plan Format

The state of the downloader is maintained in a single, tab-separated values (TSV) file, referred to as the plan. Each line represents a target web page and its status. The plan contains the following columns:

`age` An integer counter indicating the number of update cycles since the URL was last seen in its feed. A value of zero means it is still active. This is used to phase out stale entries.

`status` The download state, such as `new` (recently discovered), `ok` (successfully downloaded), or an error code for failed attempts.

`retries` The number of download attempts made for the URL.

`seen` A timestamp marking when the URL was first discovered by the fetcher.

`published` The publication timestamp as provided by the feed.

`feed` The URL of the source web feed.

`url` The target URL of the web page to be downloaded.

`title` The title of the web page as provided by the feed.

Given its potential size, the plan file is compressed using Zstandard (zstd) to manage disk space, especially as historical backups of the plan are retained for every run. Entries are not removed from the plan during a run. Deletion is handled as a separate maintenance step, where entries with a non-zero `age` are considered safe to remove, as they are no longer present in the live feeds.

3.4. Implementation

FeedFetcher is implemented in Rust, a choice motivated by the language's guarantees of performance, memory safety, robust concurrency, and portability. It leverages the Tokio asynchronous runtime, which enables efficient handling of thousands of concurrent network connections. The TSV plan files are parsed using a high-speed, custom parser written in Rust.

The final output is a zstd-compressed stream of JSON Lines, where each line contains the full HTML body of a downloaded page along with its associated metadata. Text extraction and processing is handled asynchronously in a subsequent processing step.

3.4.1. Evolution from a Python-Based Predecessor

An earlier prototype of the downloader was developed in Python using an SQLite database for state management. This implementation faced significant performance bottlenecks, primarily due to database locking contention under concurrent load, leading to prolonged run times of several hours. In addition, the database files were very large relative to the actual payload size, with opaque internal structure and I/O intensive access patterns, which proved difficult to debug and to reason about.

An attempt to migrate to asynchronous Python did not fully resolve these issues. Database contention worsened, and the complexities of Python's async ecosystem, particularly around task cancelation and exception handling in long-running network tasks, proved challenging. The loosely-typed nature of the language also made the program susceptible to runtime errors from unexpected edge cases common in web crawling.

The rewrite in Rust directly addressed these challenges, resulting in a more performant, stable, and maintainable system. The new version always reads and writes the state files in a sequential order, providing great performance even on traditional hard-disk storage arrays.

3.4.2. Availability

FeedFetcher is available under the GPL-3.0 license and can be accessed from its GitHub repository: https://github.com/ondra/feed_fetcher. Installation instructions and tips for running the tool are provided in the readme file there.

4. Finding Web Feeds to Fetch

Since 2021, we have used FeedFetcher to regularly download web texts from web feeds and update our monitor corpora. The project now covers 64 languages (as of March 2026). New high-quality feeds are added both manually and, more recently, through an automated acquisition process.

The main limitation in expanding the size and diversity of the resulting corpora lies not in the scalability of the collection and processing software, but in the limited number of web feeds currently known to us.

4.1. Manual Feed Exploration

As a first step, the RSS feeds are collected manually. The extent to which this process is feasible

Language	Feeds active before	Web pages searched (millions)	Unique feeds extracted (millions)	Filtered feeds yielding nice text	Feeds active after	Active feed gain
Arabic	213	143	10.4	294	491	131 %
Czech + Slovak	870	149	3.8	1,697	2,370	172 %
English	6,023	591	20.7	8,538	13,629	126 %
Estonian	318	47	1.6	229	497	56 %
French	229	148	4.5	6,384	5,682	2,381 %
German	468	192	5.1	8,199	7,966	1,602 %
Irish	8	9	0.2	14	16	100 %
Italian	716	77	2.6	2,186	2,631	267 %
Portuguese	432	90	2.7	2,118	2,467	471 %
Spanish	307	123	3.7	18,737	16,515	5,279 %

Table 1: Gain of active feeds using the automated web feed exploration.

largely depends on the language in question. Unfortunately, comprehensive lists of RSS feeds are rarely available, except for English, where dedicated websites provide such lists that can be downloaded using simple scripts. Manual collection, however, offers an important advantage – It allows for greater control over the content that will ultimately be downloaded and compiled into corpora.

The process typically begins with a predefined set of topics. A researcher or external annotator uses these topics as a starting point to search for relevant websites through one or more search engines. In some countries (e.g., the Czech Republic), national search engines are available, and these may yield distinct results, making it worthwhile to experiment with different search strategies.

Each website identified through this process is then examined manually to determine whether it provides an RSS feed. In straightforward cases, an RSS icon or a link labeled *RSS* is clearly visible on the webpage. In other cases, the feed may only be discoverable in the page's source code. Browser extensions designed for detecting RSS feeds can assist researchers in locating feeds more efficiently. Once identified, the feed URLs are copied into a spreadsheet for further processing.

4.2. Acquiring Feeds Automatically

Using a manual approach to identifying web feeds, we collected a fair number of sources. However, most feeds available online remained undiscovered. English was the most represented language with 8,047 active feeds, while Irish had only eight.

As a part of a previous project, we had also crawled a large number of general, non-timestamped web pages. From this dataset, we extracted feed links found in HTML `<head>` sections using a regular expression matching the usual feed content types[2].

To filter out low-quality or irrelevant links, we retained only those appearing at least 20 times in the HTML data and located either within the national top-level domains (TLDs) of the target languages or within websites contributing at least 20 pages of high-quality text to the general corpus. Links containing other language codes or foreign TLDs were excluded. The remaining links were expanded with common feed paths (`/rss`, `/rss.xml`, `/feed`, etc.).

All filtered links were processed by both stages of FeedFetcher: *update* and *fetch*. The retrieved data passed through the same text-processing pipeline used for building production monitor corpora, including the extraction of paragraphs of quality text and near-paragraph de-duplication. Feeds contributing at least one cleaned text were kept.

The resulting feeds were added to the production list and integrated into monitor corpora for 11 selected languages, ranging from low-resourced (Irish) to high-volume (English, Spanish). Feed counts and relative gains are summarized in Table 1. For this evaluation, a feed is considered active if it produced nice text between June and October 2025. Here, "nice text" denotes non-short, near-deduplicated paragraphs in the target language.

Inspection of the new feeds revealed many small sources such as personal blogs – in contrast to manually identified feeds, mostly originating from newswires. Although the increase in active feeds ranged from 56 % (Estonian) to 5,279 % (Spanish), the corresponding growth in the text volume was smaller. Nevertheless, the diversity and representativeness of the monitor corpora has improved through the inclusion of these additional sources.

[2] `application/rss` and `application/atom`

	Lemma	Trend	Frequency	Sample
1	agentic	3.08	36,624	
2	stablecoin	2.36	50,792	
3	stillness	1.80	13,550	
4	completeness	1.73	27,079	
5	perplexity	1.60	27,316	
6	legality	1.54	39,236	
7	measurable	1.54	59,927	
8	imago	1.54	11,771	
9	reshape	1.48	140,552	
10	slop	1.48	12,315	

	Lemma	Trend	Frequency	Sample
11	tokenized	1.48	13,130	
12	doomsday	1.43	18,313	
13	midterm	1.43	27,803	
14	guesswork	1.38	14,896	
15	structured	1.33	117,165	
16	reinvention	1.33	14,062	
17	performative	1.33	13,309	
18	fragmented	1.28	34,994	
19	tokenization	1.28	13,600	
20	scalable	1.23	103,176	

Figure 1: Top trending lemmas in the English Trends corpus in 2025 calculated by Sketch Engine.

5. Production Deployment

This section describes the setup of our FeedFetcher based system for continuous text collection from web feeds. We also provide an example of a time-based word analysis to illustrate how long-term accumulation of timestamped data can help reveal word use trends.

Corpora built with this setup, along with many examples of their use for studying linguistic trends and language change, are described in (Herman et al., 2025).

5.1. Deployment Setup

The system was designed for robustness and modularity, enabling quick issue mitigation, reproducible processing, and flexible modification or replacement of individual components within data processing pipelines. At present, we collect data in 64 languages from sources yielding roughly 2 billion tokens in 4 million documents per month.

The recommended infrastructure setup for FeedFetcher supports data acquisition and processing in over 50 languages and scales to corpora of up to 100 billion tokens for the largest languages. It runs on a Unix-like system with 32 CPU cores and 512 GB RAM. Data processing is orchestrated by scheduled tasks (`cron`):

- feed fetching: every 4 hours (6 times per day),
- text processing: once per day,
- corpus compilation: twice per week,
- deployment to production servers: twice per week (after compilation),
- data backup and cleanup: once per day.

5.1.1. Text processing

Processing follows the procedure described in (Kilgarriff et al., 2014). Data are organized by year and language, with one web page corresponding to a single document. Paragraphs are extracted from the original HTML using jusText (Pomikálek, 2011), which filters out short or low-quality text and non-target-language content. Language identification at document and paragraph levels is performed with CLD2.[3] Sensitive strings (e.g., email addresses, credit card numbers) are masked. Metadata such as topic, genre, and geographic scope are assigned based on prior annotation of feeds or web domains.

Text normalization converts characters to canonical Unicode form and standardizes whitespace and symbols. For space-separating scripts, tokenization uses Utok (Rychlý and Špalek, 2022) with language-specific abbreviation lists. Lemmatization and morphological tagging are handled by language-specific NLP tools. Sentence boundaries are marked either by the analyzer output or heuristic rules. Duplicate and near-duplicate paragraphs are removed using Onion (Pomikálek, 2011). Final corpora are indexed twice a week by the Manatee corpus management system (Rychlý, 2007) and

[3] https://github.com/CLD2Owners/cld2

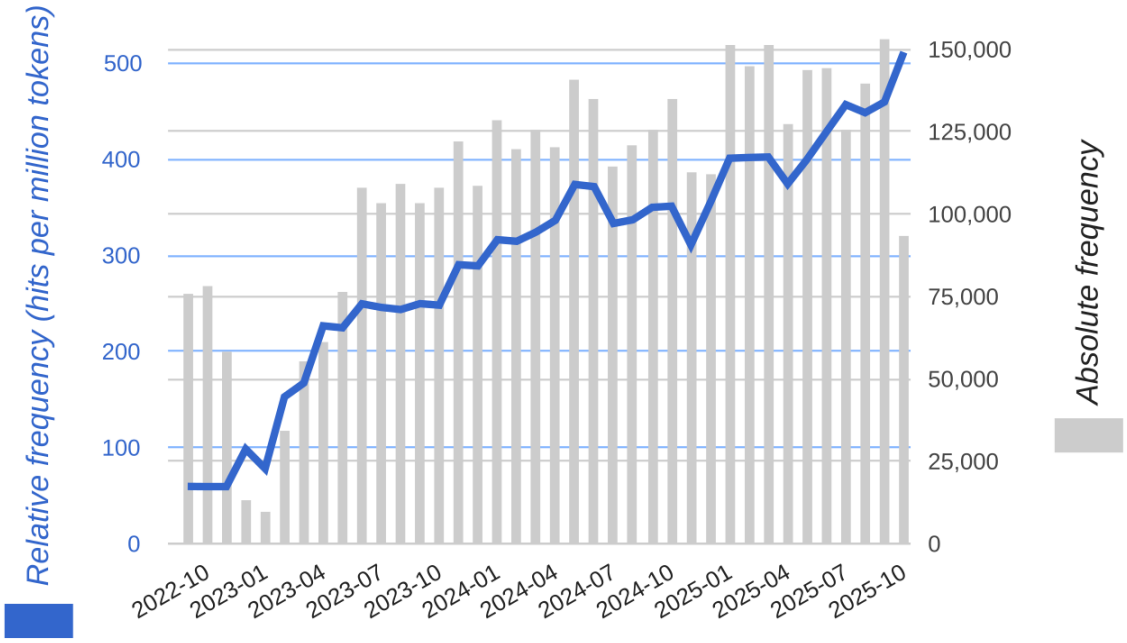

Figure 2: Timeline of the concordance of word "AI" in the English Trends monitor corpus based on timestamped web texts obtained using FeedFetcher – as displayed in Sketch Engine.

deployed to Sketch Engine (Kilgarriff et al., 2014) production servers.

5.1.2. Reproducibility

The workflow is fully automated and managed via text production recipes in `Makefiles`, ensuring efficient reprocessing whenever tools or data change. Each step is executed through `cron`, enabling automatic recovery after outages. If a processing tool is replaced (e.g., when a new morphological analyzer is deployed), removing all its derived data triggers automatic regeneration of all downstream results through the dependency mechanism of `make`.

With this setup, the entire set of corpora can be regenerated from the backup of FeedFetcher output simply by running `make`.

5.2. Trend Corpora available in Sketch Engine

The texts downloaded by FeedFetcher are processed and loaded into Sketch Engine, available to all of its users. Sketch Engine is a corpus management and text analysis system, which indexes the data and makes it accessible for large-scale linguistic research.

Rich Metadata

The system preserves the metadata extracted by FeedFetcher. Each document is annotated with its source feed, website URL, publication timestamp, and genre. This allows users to filter the data and compare different subcorpora. Sketch Engine provides a wide range of features to analyze the collected text, including:

Linguistic search and Concordance: Users can search for complex grammatical patterns using the Corpus Query Language (CQL) and inspect the matching occurrences in their original context.

Word Sketch: A tool that processes typical collocations to provide a summary of a word's grammatical and collocational behavior.

Trends: Uses the publication timestamps to identify words and senses that increase or decrease in frequency over time.

Frequency wordlists: Generates lists of words, lemmas, or PoS tags sorted by frequency.

Keyword and terminology extraction: Finds words and multiword terms that are typical for a specific text or time period by comparing it to a reference corpus.

5.3. Diachronic analysis

Regularly updated corpus data enriched with metadata, such as publication dates, makes it possible

to trace how the frequency of words and phrases changes over time. This approach enables researchers to investigate temporal dynamics in language use, including the emergence of new vocabulary, semantic shifts, or the decline of outdated expressions. Such analyses reveal how linguistic trends correspond with broader social, political, or technological developments. Notably, sudden increases in word frequency often coincide with specific real-world events, offering valuable insights for sociolinguistic and discourse-oriented research.

Figure 1 shows the most salient trending lemmas in the English Trends corpus during the year 2025, extracted by the Trends feature of Sketch Engine.

Figure 2 shows the frequency timeline for the term *AI* (artificial intelligence) in the English Trends corpus. The usage started to rise sharply in late 2022. This aligns with the public introduction of conversational AI systems such as ChatGPT, which entered everyday discourse and media coverage at this point. In the visualization, the blue line represents relative frequency (occurrences per million words), while the grey bars show absolute frequency counts.

6. Conclusion and Future Work

This paper introduced FeedFetcher, an open-source web feed downloader designed for corpus construction, that enables the creation of large, timestamped monitor corpora. The tool was developed with a strong emphasis on robustness and scalability, both of which have been validated through our extensive deployment experience.

Using the tool, we continuously download web pages distributed via web feeds and update monitor corpora that enable research into linguistic trends (including neologisms) and language change over time, across 64 languages, as of March 2026.

Future work will focus on expanding language coverage and increasing feed diversity by applying the automated discovery procedures experimentally carried out and described in this paper. These efforts aim to further enhance the size, representativeness, and analytical potential of the resulting corpora.

7. Ethical Considerations and Limitations

We develop and use FeedFetcher to collect timestamped web pages published through public feeds, operating in good faith that the authors intend their material to be publicly accessible online. Analyses supported by the resulting corpora are conducted on aggregated data while individual texts are not redistributed in full. We respect authors' rights and are prepared to remove specific documents from the resulting corpora upon request; however, no such requests have been received to date.

8. Bibliographical References

Abdulrahman Al-Thubaity and Mohammed Alhoshan. 2019. Ararss: A system for constructing and updating arabic textual resources. *Journal of King Saud University – Computer and Information Sciences*, 31(3):353–365.

Digiteum. 2017. New monitoring corpus and super corpus platform: Case study. Digiteum blog (online).

Cédrick Fairon. 2006. Corporator: A tool for creating rss-based specialized corpora. In *Proceedings of the 12th Annual Conference of the European Chapter of the Association for Computational Linguistics*.

Cédrick Fairon, Kévin Macé, and Hubert Naets. 2008. Glossanet 2: A linguistic search engine for rss-based corpora. In *Proceedings of the 4th Web as Corpus Workshop*, pages 34–39, Marrakech.

Ingrid Falk, Delphine Bernhard, and Christophe Gérard. 2018. The logoscope: a semi-automatic tool for detecting and documenting french new words.

Felix Hamborg, Norman Meuschke, Corinna Breitinger, and Bela Gipp. 2017. news-please: A generic news crawler and extractor. In *Proceedings of the 15th International Symposium of Information Science*, pages 218–223.

Daphne Kerremans, Susanne Stegmayr, and Hans-Jörg Schmid. 2012. The neocrawler: Identifying and retrieving neologisms from the internet and monitoring ongoing change. In K. Allan and J. A. Robinson, editors, *Current Methods in Historical Semantics*, pages 59–96. De Gruyter Mouton, Berlin.

Adam Kilgarriff, Vít Baisa, Jan Bušta, Miloš Jakubíček, Vojtěch Kovář, Jan Michelfeit, Pavel Rychlý, and Vít Suchomel. 2014. The sketch engine: ten years on. *Lexicography*, 1(1):7–36.

Blaž Novak and Mitja Trampuš. 2012. Internals of an aggregated web news feed. In *Proceedings of the Slovenian KDD Conference (SiKDD)*.

Jan Pomikálek. 2011. *Removing boilerplate and duplicate content from web corpora*. Ph.D. thesis, Masaryk university, Faculty of informatics, Brno, Czech Republic.

Pavel Rychlý. 2007. Manatee/bonito–a modular corpus manager. *RASLAN 2007 Recent Advances in Slavonic Natural Language Processing*, page 65.

Pavel Rychlý and Samuel Špalek. 2022. Utok: The fast rule-based tokenizer. In *Proceedings of the Sixteenth Workshop on Recent Advances in Slavonic Natural Language Processing (RASLAN)*, pages 149–154.

Ivelina Stoyanova, Svetlozara Leseva, Martin Yalamov, and Svetla Koeva. 2016. An online system for neologism detection in bulgarian.

Aleksandra Tomaszewska, Dariusz Czerski, Bartosz Żuk, and Maciej Ogrodniczuk. 2025. Neon: A tool for automated detection, linguistic and llm-driven analysis of neologisms in polish. In *International Conference on Computational Science (ICCS)*.

9. Language Resource References

Sultan Alzahrani. 2013. Building, profiling, analysing and publishing an arabic news corpus based on google news rss feeds. In *Arabian Forum on Information Retrieval (AIRS)*.

Ondřej Herman, Miloš Jakubíček, Jan Kraus, and Vít Suchomel. 2025. From word of the year to word of the week: Daily-updated monitor corpora for 25 languages. In *Electronic lexicography in the 21st century. Proceedings of the eLex 2025 conference*.

Simon Krek, Ondřej Herman, Jan Bušta, Miloš Jakubíček, and Blaž Novak. 2017. Jsi newsfeed corpus. In *The 9th International Corpus Linguistics Conference*. University of Birmingham.