Method for implementing a complex modular system for analyzing information sources

Zhengbing Hu[1,†], Dmytro Uhryn[2,†], Artem Kalancha[2,*,†], Viktor Ivashko[2,†] and Viktor Ilin[2,†]

[1] *Scholl of Computer Science, Hubei University of Technology, Nanli Road, Hong-shan District, Wuchang 22, 430068 Wuhan, Hubei, China*

[2] *Yuriy Fedlovyvh Chernivtsi National University, Kotsiubynskoho Street 2, 58012 Chernivtsi, Ukraine*

Abstract

The paper presents a method for implementing a modular information system of the full cycle – from automated collection to deep analytics of text messages, specialized in open sources and text streams. The system combines text preprocessing with the calculation of a set of metrics, in particular lexical similarity, temporal-semantic influence, classification of hostile rhetoric, clustering and construction of a directed graph of sources. The system also includes an asynchronous processor that receives tasks from the queue and stably interacts with the API (using Circuit Breaker), and also stores the results in a document-oriented database. At the analytics level, TSI captures the direction and intensity of cross-channel impact even with low overall lexical similarity, while the hostile speech detection module evaluates rhetoric at the message and source levels. The summary interface generates brief automatic conclusions for each source.

Keywords

Timed Semantic Influence, cosine similarity, hostile rhetoric, source clustering, information influence graph, automated data collection.

1. Introduction

The actual informational space is described by fast changes of headlines, fragmentation of sources, and a surplus of sources which often share the same information in different ways, speeds, and contexts. The lack of editorial oversight in platforms like messaging apps leads to repeated posts, mutual citations, and paraphrases. It creates multiple copies of the same message, complicating the task of verifying content, tracing the original source, and identifying connections between sources. In this situation, it's necessary not only to gather large amounts of text from open sources, especially Telegram channels, but also to uncover hidden relationships, rhetorical similarities, and possible influences between sources over time.

As Natural Language Processing (NLP) methods have expanded greatly, it makes sense to combine NLP techniques [1] to extract and compute metrics in order to compare two or more sources. It should contribute for calculated overall conclusions about the behavior of sources and the structure of their connections. Using those algorithms into a specific combination, we can bring together collection, pre-processing, and analyze it deeply, while saving results at every stage for future visualization and comparison.

2. Problem Statement

The today's informational space changes rapidly. This is particularly evident on social platforms and messengers, where the lack of editorial control allows for the spread of all types of

AIT&AIS'2025: International Scientific Workshop on Applied Information Technologies and Artificial Intelligence Systems, December 18–19 2025, Chernivtsi, Ukraine

[1*] Corresponding author.

[†] These authors contributed equally.

drzbhu@gmail.com (Z. Hu); d.ugryn@chnu.edu.ua (D. Uhryn); kalancha.artem@chnu.edu.ua (A. Kalancha); v.ivashko@chnu.edu.ua (V. Ivashko); ilin.viktor@chnu.edu.ua (V. Ilin)

0000-0002-6140-3351 (Z. Hu); 0000-0003-4858-4511 (D. Uhryn); 0009-0004-1451-7470 (A. Kalancha); 0000-0002-9339-4648 (V. Ivashko); 0009-0009-8124-2709 (V. Ilin)

information – from news and analysis to disinformation, manipulation, and aggressive rhetoric. In this setting, we need to build effective tools that can automatically collect, process, and analyze messages from open sources.

The most challenging thingh is analyzing large volumes of text data in real time or close to real time. The issue is larger then just collecting messages. It is required to identify hidden connections between sources, evaluate their rhetoric, and identify coordinated activities or information influences that may not be clear with a superficial view. For complex analysis, both the content of the messages and the timing of their distribution are important factors.

3. Formulation of the purpose of the article

The main goal of current work is to create a method for building an information system. Proposed system should analyze open sources of text information. It will combine tools for collecting, pre-processing, analyzing data using natural language processing techniques, and visualizing results. The system should be scalable, stable, and precise in its calculations. Also it should allow for expansion of new sources and metrics in the future to count more factors of the information. The work looks at a range of technical and analytical methods, along with a modular structure approach in order to enable flexible, multi-stage analysis of text messages. The results from reproduced analysis can be applied in scientific research and practical tasks, like OSINT-analysis, society awareness of harmful sources etc. These tasks include identifying information influences between sources, forming information graphs, determining levels of hostility for each source, and classifying the main topics of sources.

4. Justification of the analysis of scientific research sources

A study of the MediaRank system demonstrates that ranking approximately 50 thousand news sources based on the citation/reputation graph, content features, popularity and “bottomline pressure” indicators gives global “quality-impact” of sources [2], but does not directly measure the temporal-semantic impact between sources at the message level.

In turn, the studied NELA ecosystem provides a toolkit and large datasets of news with features for comparing sources, in particular the analysis of content distribution networks (copying or republishing) between publications [3]. The analyzed methods build similarity and copying graphs, apply vector representations of nodes (Node2Vec) and compare with reliability and bias.

The scientific work Semantic “Echo” of Strategic Communications proposes a metric for measuring the echo (impact) of public messages in social networks through semantic similarity: the texts are encoded with sentence-embedding, then the cosine similarity between the generations of organizations and the tweet before/after the event is calculated [4]. Changes are compared with the basic background of activity.

Another way of detecting the impact between events (or messages in our case) was also analyzed. Hawkes processes are a class of stochastic point processes in which the occurrence of an event increases the probability of the occurrence of subsequent events in the near future [5]. They are described as self-exciting processes: each new event increases the intensity of the process for a certain time, after which this impact gradually decreases. The intensity depends on the base level and on the sum of the contributions of all previous events through the impact kernel.

5. Method of implementing a modular system for analyzing information sources

The first step in proposed method is to examine the input data from the chosen data source. Since the system will handle large amounts of text and produce general estimates for information sources, it’s crucial to define the structure, form, and analysis value of input data, which includes identifying stable fields, gaps, the text’s language, and its temporal patterns.

Next step is the system requirements formulation: includes determining a set of measurable features relevant to our area of research, such as lexical similarity, indicators of temporal distribution, and rhetorical classes at both the message and source levels. We also select methods to calculate these features. The non-functional requirements address the expected performance, processing speed, fault tolerance, security, and scalability.

Using the outlined requirements, we design a general architecture. The modules must work together so the final system is reliable and resilient against partial failures. Modularity is essential for both expansion – adding more complex models or increasing throughput – and for clear responsibilities: collection, processing, storage, and visualization are all handled separately. The core of the system is effectively represented by a small set of modules: *MISA = {API, UI, Scheduler, NLP, Preprocessing, Queue, Database}*, and the external data provider, Telegram API, acts as a separate integration component outside of *MISA (Modular information system analysis).*

The next step includes breaking down the key subsystems for pre-processing and natural language processing. It's important to establish clear relationships between metrics and their calculation methods as well as their dependencies on other metrics (of applicable). It includes understanding what influences text vector representations, where sensitivity to language or lemmatization lies, and how the parameters of methods and response impact a back-off of timing and precision.

Implementing the modules is a central point of the proposed method, accompanied by testing metrics against known estimates and fine-tuning parameters. Both standard checks, like consistency, reproducibility, and sensitivity to noise, and comparisons with basic approaches are useful to understand the advantages of each engineering approach.

The final step is the generation of short automatic conclusions for each source, which bring together key metrics and compare them with other sources. It is these summaries that turn massive numerical arrays into understandable solutions: who influences what, where informational waves appear, how rhetoric changes over the time, and which connections between sources are stable.

An information system for analyzing information sources is required meet several functional and non-functional requirements to ensure its effectiveness, scalability, and ability to conduct analytical research in real or near real time. The system must be designed to work with dynamic information flows, especially messages from open sources like Telegram channels [6], which are frequently used to share news, propaganda, or coordinated disinformation.

One basic requirement for the system is the ability to add new sources of information for processing and analysis. In the initial implementation stage, Telegram channels are the first type of platform we use, but the design should allow for future expansion to other platforms. Adding a new channel should be easy for the user and automatically trigger the collection of historical messages (within the available API), saving them for further processing.

Users should have a user-friendly interface to check the availability of data in the database for each source. The feature comes in the form of a timeline that displays whether data exists for certain periods. For instance, if messages for a specific channel have been collected only for the last three months, the system should show selected ranges and let users other select time ranges for deeper analysis.

Access to basic statistical information about each source is another important requirement. Required set of statistics should include the average message length, average word length, the number of messages by type (text, photo, video, audio), and a list of the most frequently used words. Moreover, the system should show the average daily activity of the channel, indicating how many messages are published per each hour. It help provide a basic understanding of the communication style of the source before applying more complex NLP methods on it.

To effectively monitor technical performance, the system should allow users or administrators to view current process activity. They should be able to see whether text processing is happening (through the Processing Handler module) or if NLP operations are being conducted.

Beyond analyzing individual sources, the system should also support aggregated analytics across all sources simultaneously. It includes constructing and viewing a cosine similarity matrix among allsources, allowing for the identification of thematic or stylistic connections between channels. A Timed Semantic Influence (TSI) matrix should also be generated to assess the temporal semantic influence of one source on another. All these metrics are illustrated in an interactive graph, enabling users to select a specific node (channel) for a more detailed analysis of connections and influences.

The system (see Figure 1: System Asrchitecture Module diagram.) should operate in a mode of periodic data updates, at least daily or weekly. The task scheduler should ensure automatic loading, processing, and analysis of new messages without user intervention. This keeps the system current, allowing for the timely detection of new information trends or changes in source behavior.

The system should automatically generate short summaries for each source based on the analytics. Summaries can cover source classification (e.g., news, propaganda, persona), main topics, trends in changing of style or rhetoric, and shifts in influence graph positions, whuch allows users to quickly understand the content and potential threats from a particular source without needing to review and analyze the raw messages.

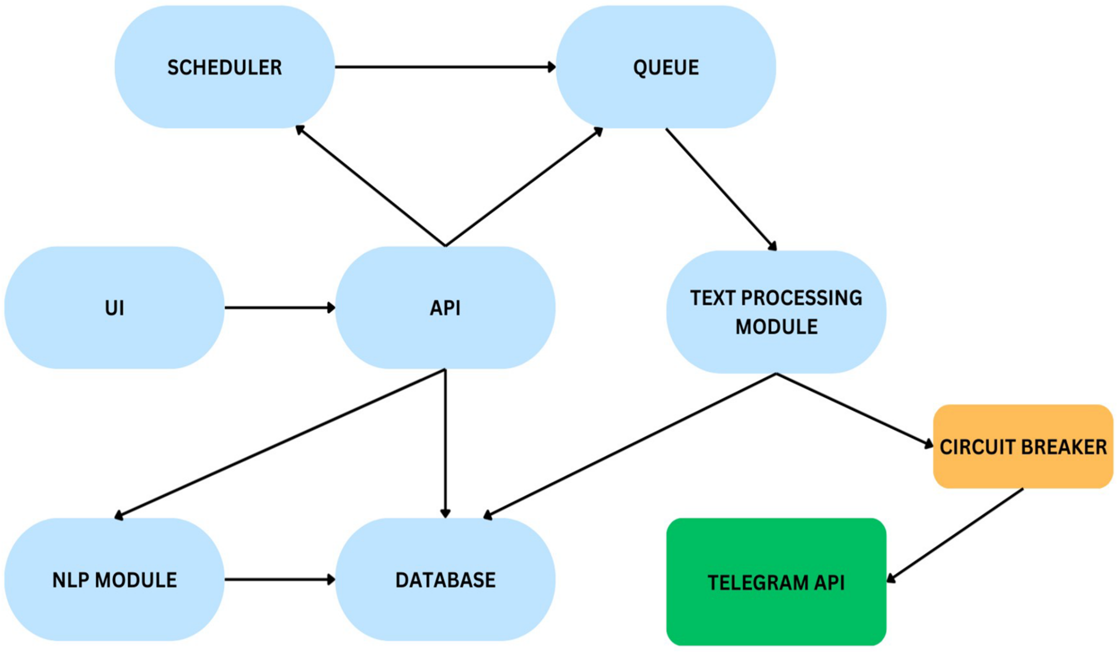

Figure 1: System asrchitecture module diagram.

UI (User Interface). UI acts as the main point of interaction between the user and the analytical system. It provides access to key functions: viewing a list of information sources, displaying statistical characteristics of each of them, building activity graphs, analyzing the structure of messages and the results of NLP processing. UI also allows you to work with analytical tables, such as similarity matrices or influence graphs between sources, providing representative visualization and filtering of information. Special attention is paid to the intuitiveness of the interface so that even a non-specialist user can easily navigate the results.

API. A connecting component that connects the user, the administrative interface and all system modules between them. The API is used to call the main functions: adding new sources, starting processing, obtaining statistics and analytical results. In addition, the API performs the basic logic of pre-processing queries, namely: aggregating statistical data based on information from the database, performing preliminary filtering and calling the functions of the corresponding subsystems.

Task queue. The system uses the Kafka message broker, which acts as a task queue. Queue allows you to scale information processing by dividing large tasks into smaller subtasks and ensuring their stable execution [7]. Kafka guarantees the preservation of each message, maintenance of the queue, redelivery in case of failures, and load balancing between executions.

This ensures reliable delivery of tasks to the processor function, even in the event of high load or temporary unavailability of individual system components.

Data processing module. The module is responsible for performing tasks of preprocessing text messages [8]. Its main functions include: parsing messages, cleaning text, extracting key fields, filtering irrelevant information, and forming a standard format for NLP analysis. The processed data is stored in the database in a structured form, which allows for easy sampling, aggregation, and analysis in the future. The module interacts with the Kafka queue, accepting tasks for processing and sending the results further to the system.

Let $S = \{s_1, \ldots, s_N\}$ – be a set of sources, and for each s we have a sequence of messages $M_s = \{(t_k, x_k)\}$ where t_k is the publication time, x_k is the raw message text.

Before saving, encoding is applied: a dictionary lemma V and an injective mapping $c\colon V \longrightarrow N$, where V is the word and N is the code. Each prepared message is represented by a code sequence $x'_k = [c(w_1), \ldots, c(w_n)]$, and the inverse mapping c^{-1} allows us to decrypt the text. As a result, two collections are supported: *Dictionary (w, c(w))* and *Messages* (s, t_k, x'_k).

The task scheduler is a module that automates the launching of data processing tasks. It regularly launches tasks to update information from sources, such as daily or weekly. The scheduler identifies which sources need data updates, generates tasks, and adds them to the Kafka queue. The scheduler has flexible configuration options; it supports setting periods, analysis types, and sources. It also helps avoid re-running areas that have already been processed.

Natural Language Processing Module. The module is the central part of the analytical core of the system. It accepts tasks via API and runs text analysis according to specified parameters. The module supports configuration of the analysis type: Cosine Similarity [9], TSI, hostile language detection [10], clustering [11, 12], etc. The processing results are stored in the database along with the corresponding time stamps and launch parameters, which allows not to duplicate calculations and speed up repeated queries. The module is scalable in terms of new metrics or processing steps and allows parallel processing of individual sources or periods.

Database. The system uses the document-oriented database MongoDB. Mongo allows you to store messages in JSON document format, which provides flexibility when storing data of different structures (for example, messages with text, images or links to media files). The main advantages of MongoDB in the context of Big Data are scalability and a convenient data aggregation mechanism used in the formation of statistics and analytics [13], which allows us to quickly execute queries on a large number of records without losing system performance.

6. Preprocessing module

The Preprocessing module, or Processing Handler (see Figure 2: Text Preprocessing Workflow), is the basic part for converting raw messages from platforms into structured data that can be further analyzed in our system. It operates as an asynchronous functional unit that takes parameters from the Kafka queue, processes them according to specified logic, and stores the results in the database. The module is designed to work reliably with unstable external data sources and considers the unique aspects of using the Telegram API or any other social platform in the future.

Unloading data from the Telegram API begins with initializing a request to retrieve a block of messages from the chosen channel for a specific time period. Since the Telegram API does not guarantee stable connections, the system employs the Circuit Breaker [14] pattern. Proposed approach helps avoid overload during failures by temporarily blocking requests after a series of unsuccessful attempts, allowing the service to recover before trying again.

Additionally, the Telegram API periodically requires reauthentication of the session. If access needs a confirmation code, such as an SMS code, the system automatically generates a message and sends a notification to the administrator's central mail, prompting them to log in and enter the code.

Calculation of basic message metrics. For each received message, basic characteristics are immediately calculated: number of characters, number of words, presence of URL links, presence of media, whether the message is a repost. Secondary characteristics are also stored, such as message structure, previous tags, original author ID (in case of repost), which may be useful in further analysis.

Determination of message language. At this step, the system automatically determines the language of the message – Ukrainian or Russian. It's important for subsequent processing steps, in particular for translation and lemmatization. Both simple rules (for example, character frequency) and machine learning libraries can be used to determine the language [15].

Text tokenization. The message is broken down into individual tokens (words or characters) according to linguistic features. Tokenization is the first step in preparing the text for analysis, allowing you to work with the text at the word level.

Text cleaning. Stop words (the most common service words), numbers, unnecessary punctuation marks, URL links, emojis, special characters, and other information that does not carry a linguistic load are removed from the text. Cleaning improves the quality of subsequent lemmatization and semantic analysis.

Translation of Russian-language words. If the message is written in Russian, each word is checked in the local cached dictionary of matches. If a corresponding word already exists, the word is automatically replaced with a Ukrainian analogue. If the word is not found, it is translated using a third-party translation tool (for example, via the Google Translate API or another open-source tool), and the result is stored in the dictionary for reuse. Message language normalization provides a significant increase in productivity and precise of final results.

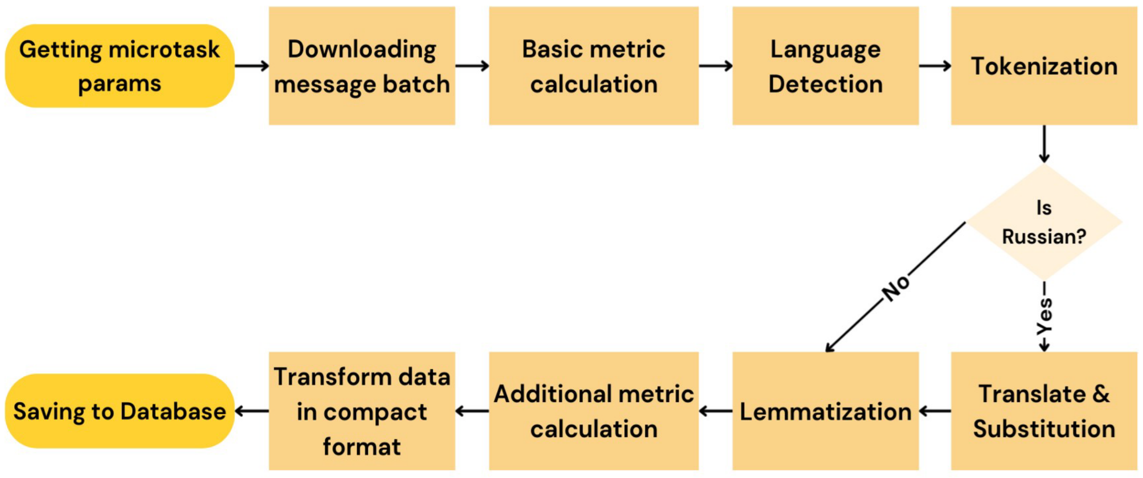

Figure 2: Text preprocessing workflow.

Lemmatization. After translation, the text undergoes lemmatization – the process of reducing words to their original (dictionary) form.

Calculation of additional metrics. Based on the cleaned and lemmatized text, additional characteristics are calculated that can be used at subsequent stages of NLP analysis: word frequency, syntactic features, number of unique words, share of links, etc. These metrics are stored in a separate field of the document, which allows them to be used selectively.

Conversion to a compact format. After processing, the message is converted to a compact format for storage: duplicates are reduced, repeated elements are aggregated, auxiliary fields are reduced. This saves space in the database and speeds up subsequent selections.

Saving to the database. The final processed object is written to the document-oriented MongoDB database. Both raw data and calculated metrics, language, creation time, message source, and technical information are stored.

Our previous analysis represent that the average message length is about 234 characters, with variations from 132 to 366, depending on the channel [16]. After thorough data cleaning, the average length dropped to 220 characters, reducing noise and improving the quality of further processing. On average, one message consists of 34 tokens, but that number decreased to 27 after removing stop words, marking a 21% reduction.

Several methods were tested for optimizing message storage: token arrays, strings, and numeric encoding. The least efficient format was the array, which took up 8.03 MB. Switching to strings reduced it to 5.33 MB, achieving a savings of around 17%.

The best results came from token encoding, where each word is assigned a numeric identifier. In this method, the average document size shrank from 440 to 249 bytes, and the collection size decreased to 2.76 MB – almost half the size of the string format. The additional cost of maintaining the dictionary is minimal, and the encoding/decoding time is less than a second for 12,000 messages.

7. Natural Language Processing Module

The Natural Language Processing Module (NLP module) (see Figure 3: Submodules of NLPmodule) is responsible for conducting a entire analytical analysis of text data previously prepared by the Processing Handler. The main function of the module is to extract high-level information from stored raw messages: from measuring lexical similarity to constructing influence graphs and analyzing hostility of sources. Both classical text processing algorithms and own developed approaches are used, in particular Timed Semantic Influence. Each analytical step is implemented as a separate submodule, and its results are stored in the database, which allows you to avoid repeated calculations and use the data to compute more complex metrics.

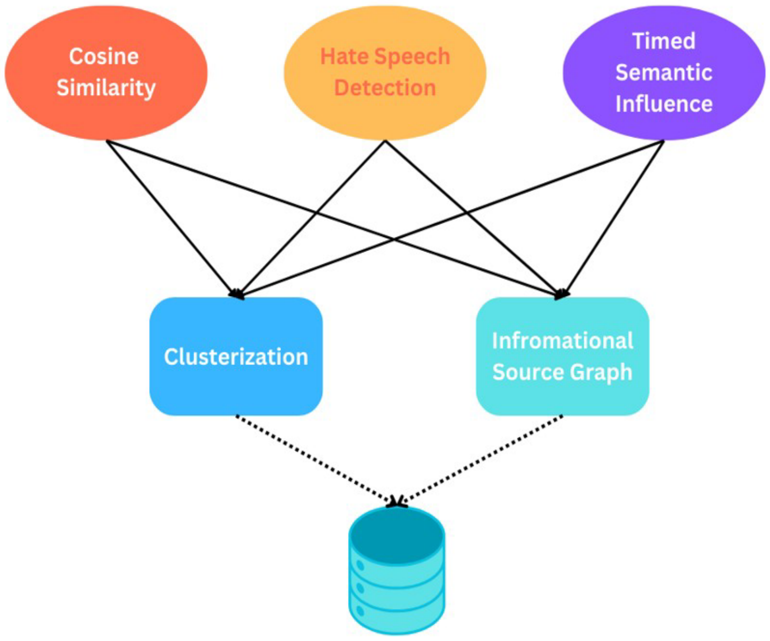

Figure 3: Submodules of NLP module.

Cosine Similarity Module. This submodule generates a vector representation of each information source based on its lexical profile. The frequency of lemmas in messages is calculated, after which a multidimensional vector z is created (see Formula 1), which represents the topic, style, and vocabulary of the source. Based on the constructed vectors, a cosine similarity matrix is calculated (see Formula 2), which allows us to determine how similar the sources are to each other in terms of vocabulary – the first basic metric that demonstrates the potential thematic or semantic proximity of channels and is further used in clustering or building influence graphs.

$$z(s)=\frac{1}{n_s}\sum_{k=1}^{n_s} v(x_k)\in\mathbb{R}^d. \quad (1)$$

Then for a pair of sources s1, s2 the cosine similarity is equal to:

$$cs(s_i, s_j) = \frac{z(s_i)^T z(s_j)}{|z(s_i)_2||z(s_j)_2|} \in [-1, 1], \tag{2}$$

where $z(s_i)$ is the corresponding vector for channel s_i.

TSI – Timed Semantic Influence. The TSI algorithm is an innovative approach that allows you to detect the temporal influence of one source on another. Its goal is to find informational waves, i.e. similar messages that appear in different sources in a short period of time. If a certain source consistently publishes key topics earlier than others, and its vocabulary is subsequently repeated in other sources, which indicates its influence. Even if the overall similarity between channels is low (by Cosine Similarity), TSI allows you to detect hidden connections between sources. The result of the work is an influence matrix, which shows the direction and intensity of information influence between pairs of channels.

$$TSI = \frac{sm(s_i, s_j)}{1 + \alpha \times \Delta t}, \tag{3}$$

where α – is the time influence coefficient, Δt is the difference between the publication times of 2 messages, *sm* is the cosine similarity function.

TSI determines the influence between channels by the appearance of semantically similar messages in close time windows. Unlike Hawkes, TSI works not only with time, but also with content, which reduces the number of false connections and integrates well into the NLP module. At the same time, its parameters are set empirically, and it is not a full stochastic model.

Advantages of TSI over Hawkes:

1. Semantically determined influence: TSI selects only pairs of posts similar in content in a given time window, so the connection is less sensitive to noisy activity matches.
2. Easy integration into your system: TSI is calculated on already prepared vectors/embeddings.

Disadvantages of TSI over Hawkes:

1. Parameters (similarity threshold, decay shape, window width) are set empirically. Without additional statistical tests, it is more difficult to make formal conclusions about causality.
2. TSI is not a full stochastic model of event generation. It is more of a content- and time-based impact estimator.

Hostile Language Detection. The submodule classifies messages based on hostility using a specially prepared dataset. The training set includes examples of messages marked as either hostile or not, which trains the model. After training, the model processes all messages and assigns each a hostility score. The average hostility level for each source is then calculated, serving as a metric for further analysis.

Clusterization. Once vector representations (lexical profile, hostility, TSI impact) are created for all sources, they are used for cluster computation. This submodule is developed to group sources with similar rhetorical or thematic characteristics. Standard clustering methods like DBSCAN or KMeans are used, facilitating work in multidimensional spaces. The resulting clusters identify information coalitions and groups of sources that work together or promote similar narratives in informational space.

Source Graph. The final step of analysis requires creating a graph that shows connections between sources. This submodule combines all previously calculated metrics: cosine similarity, TSI, hostility level, and statistical activity, to create a directed graph. In this graph, nodes represent information sources, and edges indicate presence and direction of influence between them. The strength of an edge reflects the relationship's intensity, based on TSI, hostile rhetoric, or thematic closeness. The graph visualizes the information landscape, highlighting isolated or overly influential sources and potential centers of coordination.

All analysis results, whether intermediate (vectors, metrics, estimates) or final (matrices, clusters, graphs), are stored in a database by source, pair of sources and data ranges, which allows for reuse in visualizations, new queries, graphing, or comparisons over time without repeating calculations.

8. Conclusions

The proposed system provides a complete cycle for gathering, processing, and analyzing text messages from information sources, particularly Telegram channels. The architecture emphasizes fault tolerance through the Circuit Breaker mechanism and ensures reporting at all processing stages. It includes automatic alerts about re-authentication needs. The modular design enables system scaling, addition of new sources, or updates to individual components without refactoring the entire system architecture of analysis, which makes the system responsive to changes in the technical and analytical environment. The NLP module analyzes text in key areas: building vector models, calculating cosine similarity, determining TSI for influence detection, assessing hostility levels, clustering channels, and creating a connection graph. The results are stored in the database for future use.

Compared to other systems, this one analyzes messages in messengers and indentify sources in a multi-metric space, including the time-semantic influence coefficient. It combines top practices from various systems with its own analysis methods.

The scientific innovation is based on the introduction of the time-semantic influence (TSI) metric, which links the timing of messages with their semantic similarity. By that, it enables the detection of directional connections between sources, even with low overall lexical similarity, leading to an influence matrix for building an interaction graph. A method for implementing a modular system for analyzing and evaluating relationships between informational sources, taking into account non-functional requirements for the system, was proposed. Furthermore, multi-metric NLP analysis is consolidated in a single module: the system integrates lexical similarity (cosine), TSI, rhetoric classification, and clustering as separate submodules while reusing intermediate results.

Practical value includes efficient source analytics. Users receive brief automatic summaries for each channel (type, topics, position dynamics in the graph), saving time on primary analytics. The combination of TSI and lexical metrics aids in detecting coordination and information waves, revealing hidden connections and tracking early narrative "waves" between channels. The system supports scalable workflows, by using of the architecture with Kafka and asynchronous processing, ensuring stable performance under heavy loads and allowing for regular updates without administrator intervention. Metric enhancement is flexible, following a modular scheme that simplifies the addition of new indicators, like additional rhetoric classifiers or network metrics, without modifying the entire system. It reuses intermediate representations in similarity matrices and influence graphs. The proposed tool developed by proposed method is actually useful for different scenarios like: including OSINT, editoral offices, and research groups, helping to identify primary sources, monitor risks for society, and map the informational space.

Acknowledgements

The authors would like to express their sincere gratitude to Yuriy Fedkovych Chernivtsi National University for providing the opportunities, resources, and supportive environment that made this research possible. We also extend our thanks to colleagues and collaborators who contributed valuable insights and assistance throughout the research and preparation of this work.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT and Grammarly in order to: Grammar and spelling check and as a smart Search Engine to find related works based on context of conversation. After using these tools/services, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, D. Roth, Recent advances in natural language processing via large pre-trained language models: A survey, ACM Computing Surveys 56 (2024) 1–40. doi:10.1145/3605943.

[2] J. Ye, S. Skiena, MediaRank: Computational ranking of online news sources, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '19, Association for Computing Machinery, New York, NY, 2019, pp. 2469–2477. doi:10.1145/3292500.3330709

[3] B. D. Horne, W. Dron, S. Khedr, S. Adalı, Assessing the news landscape: A multi-module toolkit for evaluating the credibility of news, in: Proceedings of The Web Conference 2018, WWW '18, IW3C2, Geneva, Switzerland, 2018, pp. 235–238. doi:10.1145/3184558.3186987.

[4] T. J. B. Cann, B. Dennes, T. Coan, S. O'Neill, H. T. P. Williams, Using semantic similarity and text embedding to measure the social media echo of strategic communications, arXiv preprint arXiv:2303.16694 (2023). doi:10.48550/arXiv.2303.16694.

[5] J. Worrall, R. Browning, P. Wu, K. Mengersen, Fifty years later: New directions in Hawkes processes, SORT 46 (1) (2022) 3–38. doi:10.2436/20.8080.02.116.

[6] Telegram, Telegram APIs (Bot API, TDLib, MTProto), Online documentation, URL: https://core.telegram.org/.

[7] S. Serbout, A. El Malki, C. Pautasso, U. Zdun, API rate limit adoption – a pattern collection, in: Proceedings of the 28th European Conference on Pattern Languages of Programs, EuroPLoP '23, ACM, New York, NY, 2023, pp. 1–20. doi:10.1145/3628034.3628039.

[8] J. Camacho-Collados, On the role of text preprocessing in neural network architectures: An evaluation study on text categorization and sentiment analysis, arXiv preprint arXiv:1707.01780 (2018). doi:10.48550/arXiv.1707.01780.

[9] P. Gupta, M. Buitelaar, Modelling text similarity: A survey, arXiv preprint arXiv:2307.14725 (2023). doi:10.5120/11638-7118.

[10] M. Saroar Jahan, M. Oussalah, A systematic review of hate speech automatic detection using natural language processing, Neurocomputing 546 (2023). doi:10.1016/j.neucom.2023.126232.

[11] M. E. Celebi, Y. A. Aslandogan, A survey of density-based clustering algorithms, Knowledge-Based Systems 212 (2020). doi:10.1007/s11704-019-9059-3.

[12] J. Singh, D. Singh, A comprehensive review of clustering techniques in artificial intelligence for knowledge discovery: Taxonomy, challenges, applications and future prospects, Applied Artificial Intelligence 62 (2024). doi:10.1016/j.aei.2024.102799.

[13] M. Rathore, S. Bagui, MongoDB: Meeting the dynamic needs of modern applications, Encyclopedia 4 (2024) 1433–1453. doi:10.3390/encyclopedia4040093.

[14] J. A. Valdivia, A. Lora-Gonzalez, X. Limon, K. C. Verdin, Patterns related to microservice architecture: A multivocal literature review, Programming and Computer Software 46 (8) (2024) 594–608. doi:10.1134/S0361768820080253.

[15] E. Chavez, M. Garcia, J. Favela, Fast and Accurate Language Detection in Short Texts using Contextual Entropy, Research in Computing Science 90 (90) (2015) 351–358. doi:10.13053/rcs-90-1-27.

[16] V. Lytvyn, A. Kalancha, D. Uhryn, M. Talakh, Improvement of text data storage methods, SISN 15 (2024) 102–114. doi:10.23939/sisn2024.15.102.